Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

Linnéa Bråten

# Programmeringsoppgaver i matematikk 1T

## En kvalitativ innholdsanalyse av programmeringsoppgaver i lærebøker for matematikk 1T

Masteroppgave i matematikk, MLREAL
Veileder: Alexander Schmeding
Juni 2024

NTNU
Kunnskap for en bedre verden

Linnéa Bråten

# Programmeringsoppgaver i matematikk 1T

## En kvalitativ innholdsanalyse av programmeringsoppgaver i lærebøker for matematikk 1T

Masteroppgave i matematikk, MLREAL
Veileder: Alexander Schmeding
Juni 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

NTNU
Kunnskap for en bedre verden

# Sammendrag

I 2020 kom programmering inn i læreplanen for matematikk, blant flere fag. I tillegg fikk matematikk hovedansvar for programmeringsopplæringen. Ifølge Flø (2021, s. 3) handler programmering i matematikkfaget om at elevene skal lære å bruke algoritmisk tenking som en problemløsningsstrategi. Ettersom valget av lærebøker er en kritisk faktor for hva lærere underviser og hvordan de underviser (Kongelf, 2015, s. 84) er det interessant å undersøke programmeringsoppgavene i ulike lærebøker. I tillegg er programmering i matematikkfaget ganske nytt, og mange lærere mangler nyttig kompetanse i programmering. Dermed kan man anta at lærere ikke endrer programmeringsoppgavene i bøkene i stor grad, men bruker lærebokoppgavene slik de står.

Formålet med studien er å bidra til mer kunnskap om programmering i matematikk 1T og å kartlegge styrker og svakheter i programmeringsoppgaver. Studiens forskningsspørsmål er:

(1) Hva karakteriserer programmeringsoppgaver i lærebøker i matematikk 1T og hvordan knyttes oppgavene til PRIMM og UMC?
(2) Hvilke temaer knyttes programmeringsoppgavene i matematikk 1T til og på hvilke måter knyttes programmering og matematikk sammen?

Analysen har bestått av 63 programmeringsoppgaver fra tre lærebøker der jeg har brukt et analyseverktøy fra Bråting og Kilhamn (2022) som jeg har videreutviklet. Analysen består av tre delanalyser der jeg har undersøkt handlinger, programmeringsbegreper og matematiske begreper og koblingen mellom programmering og matematikk. Resultatene og diskusjonen viser at handlingene «forme og skape» og «følge en regel» er de mest fremtredende handlingene, og som oftest finnes kun en handling.

Programmeringsoppgavene inneholdt lite av programmeringsbegrepet «feilsøke» og de typiske matematiske begrepene i programmeringsoppgavene var «funksjon», «likning», «tilnærming» og «aritmetikk». Av de matematiske begrepene kan «tilnærming» og «mønster» knyttes til «big ideas» i matematikken. De fleste programmeringsoppgavene lot elevene utforske nye matematiske ideer, hovedsamling gjennom numerisk tilnærming, eller bruke kjent matematikk som kontekst for oppgaven. Det var få programmeringsoppgaver der elevene kunne omformulere programmeringsideer til matematisk notasjon. Det samme gjelder oppgaver uten matematisk innhold.

# Abstract

In 2020, programming was incorporated into the mathematics curriculum, among other subjects. Additionally, mathematics was given primary responsibility for the teaching of programming. According to Fløe (2021, p. 3), programming in mathematics involves teaching students to use computational thinking as a problem-solving strategy. Given that the choice of textbooks is a critical factor in what and how teachers teach, it is interesting to examine the programming tasks in various textbooks (Kongelf, 2015, p. 84). Furthermore, programming in mathematics is quite new, and many teachers lack adequate programming skills. Thus, it can be assumed that teachers do not change the programming tasks in the books much but use the textbook tasks as they are.

The purpose of this study is to contribute to the knowledge of programming in Mathematics 1T and to identify the strengths and weaknesses of the programming tasks. The research questions of the study are:

(1) What characterizes programming tasks in Mathematics 1T textbooks, and how are these tasks related to PRIMM and UMC?
(2) What themes are the programming tasks in Mathematics 1T related to, and in what ways are programming and mathematics connected?

The analysis consists of 63 programming tasks from three textbooks, where I have used an analysis tool from Bråting and Kilhamn (2022), which I have further developed. The analysis consists of three sub-analyses where I have examined actions, computational concepts, mathematical concepts, and the connection between programming and mathematics. The results and discussion show that the actions "form and create" and "follow a procedure" are the most prominent actions, and often only one action was observed.

The programming tasks contained little of the computational concept "debugging," and the typical mathematical concepts in programming tasks were "function," "equation," "approximation," and "arithmetic". Among the mathematical concepts, "approximation" and "pattern" could be linked to "big ideas" in mathematics. Most programming tasks allowed students to explore new mathematical ideas, mainly through numerical approximation, or used known mathematics as the context for the task. There were few programming tasks where students were asked to reformulate computational ideas using mathematical notation. The same applies to exercises without mathematical content.

# Forord

Innleveringen av denne masteroppgaven symboliserer avslutningen av fem år som lektorstudent ved NTNU. I løpet av disse årene har jeg tilegnet meg kunnskap og erfaring i læreryrket og lært mye om temaer jeg ikke kunne fra før, blant annet programmering. Denne oppgaven har gitt meg mulighet til å utforske mer om programmering i matematikkfaget, som jeg vet jeg får god nytte av i arbeidslivet.

Jeg vil først av alt takke min veileder Alexander Schmeding for all hjelpen jeg har fått. Du har gitt meg gode råd, konstruktive tilbakemeldinger og har vært god å diskutere med. Videre ønsker jeg å takke min samboer som har lettet på arbeidsoppgavene hjemme de siste ukene før innlevering slik at jeg kunne fokusere fullt ut på oppgaven.

Sist, men ikke minst, takk til fantastiske medstudenter. Dere har gjort hverdagen skikkelig bra, både på skolen og på fritiden. Takk for god støtte, gode samtaler og gode lattere. Studiet hadde ikke vært det samme uten dere.

*Linnéa Bråten*

# Innhold

# Figurer

# Tabeller

# 1. Innledning

I kunnskapsløftet 2020 (LK20) kom det frem at elever skal lære å programmere i matematikk, naturfag, musikk og kunst og håndverk, hvor matematikk har hovedansvar for programmeringsopplæringen (Flø, 2021, s. 3). Ifølge Utdanningsdirektoratet (2019a) innebærer programmering i matematikkfaget at elevene skal lære å bruke algoritmisk tenking (AT) som en problemløsningsstrategi. Elevene skal i tillegg vurdere når det er hensiktsmessig å benytte ulike digitale hjelpemidler, inkludert programmering. Argumentene for programmering i skolen knyttes til nødvendige ferdigheter for det 21. århundre, fremtidige behov for kompetanse i næringslivet og evne til å forstå hvordan et stadig mer digitalisert samfunn fungerer (Wing, 2006, s. 34).

Programmering og algoritmisk tenking er ikke nye begrep innenfor matematikken. I 1980 presenterte Papert (1980, s. 182) begrepet «computational thinking», som på norsk omtales som algoritmisk tenking. Papert var også med på å utvikle det første programmeringsspråket for barn, kalt LOGO (Solomon et al., 2020, s. 1). Programmering i skolen slo imidlertid ikke gjennom slik Papert ønsket, og programmering ble dermed en aktivitet for spesielt interesserte. I 2006 kom en ny bølge med programmering da Wing (2006) reintroduserte begrepet algoritmisk tenking. Wing (2006, s. 33) gir ikke en kort og konkret definisjon av algoritmisk tenking. Likevel hevder hun at algoritmisk tenking er en problemløsningsstrategi som blant annet innebærer å bruke abstraksjon og dekomponering når man skal løse komplekse problemer. Det finnes flere definisjoner på algoritmisk tenking, men som vi vil se i denne undersøkelsen, innebærer alle definisjonene de samme prinsippene.

I flere europeiske land har programmering kommet inn i læreplaner og ifølge Sevik (2016, s. 22) er det økt oppmerksomhet på AT når landene velger å ta programmering inn i skolen. Hvor programmering hører hjemme i skolefagene, er imidlertid ikke tydelig. Noen land har lagt programmering til i matematikken, mens andre har tatt det med som et eget fag. Blant annet har England valgt å ta med programmering som del av et eget IKT-fag, og det samme har Danmark. Finland har derimot valgt å ta med programmering i flere fag, blant annet i matematikkfaget, på samme måte som Norge. Sverige har også tatt med programmering i flere fag, men i matematikken knytter de programmering spesifikt til algebra (Bråting & Kilhamn, 2022, s. 595).

Selv om programmering i den norske skolen ikke er en helt ny ide, er den ny i LK20. Dermed hadde de fleste lærere og elever ingen kompetanse om programmering før 2020. Lærere måtte raskt lære seg programmering for å kunne undervise elevene. Ifølge Maugesten et al. (2021, s. 5) er det tidkrevende for lærere å lære programmering fra bunn av. Man kan derfor anta at flere lærere fortsatt ikke har full kontroll på temaet. Dette kan føre til at lærere i større grad enn før lener seg på lærebøkene. Fra forskningen fremgår det at matematikklærere ofte anvender lærebøker som sin primære ressurs i undervisning (Pepin et al., 2013, s. 686; Flø, 2021, s. 3). Lærebøkene oppfattes til å reflektere de synspunktene som kommer til uttrykk i læreplanen. Det kan være en grunn til at matematikklærere primært bruker lærebøkene. Ettersom matematikklærere bruker lærebøkene mye, er valget av lærebok en kritisk faktor, som definerer hva lærere underviser, hvordan de underviser og hvordan elevene deres lærer (Kongelf, 2015, s. 84).

## 1.1 Forskningsspørsmål og formål med studien

Ettersom lærebøker kan ha stor innflytelse på hva som undervises i matematikkfaget, er det interessant å undersøke hvor godt lærebøkene kan bidra til utvikling av kompetanse og ferdigheter i programmering. Det er også interessant å undersøke hvordan programmeringsoppgavene kan knyttes til matematikk og algoritmisk tenking. Min interesse for temaet kommer fra debatten om programmering i skolen burde integreres i matematikkfaget eller være et eget fag. Ifølge Sevik (2016, s. 25-26) anbefalte en ekspertgruppe, satt ned av Utdanningsforbundet, at programmering ikke burde bli inkludert i eksisterende fag, men heller bli etablert som et eget fag. Ekspertgruppen hevdet videre at dersom programmering blir innlemmet i eksisterende fag så kunne det føre til at temaet ble nedprioritert. Ifølge Dolonen et al. (2019, s. 26) ble programmering lagt til i eksisterende fag fordi Utdanningsdirektoratet ikke ønsket å etablere et eget skolefag, samt at de andre nordiske landene har samme modell.

At programmering ble knyttet til matematikk har bakgrunn i at programmering er en naturlig del av problemløsning i matematikk (Dolonen et al., 2019, s. 26). Programmering blir også sett på som nyttig for at elevene skal utvikle seg som algoritmiske tenkere. Samtidig vil programmering som problemløsning overlappe med kritisk tenking. Debatten har gjort meg spesielt interessert i sammenhengen mellom programmering og matematikk.

I undersøkelsesprosessen til masteroppgaven kom jeg over en studie av Bråting og Kilhamn (2022). De gjennomførte en tekstbokanalyse der de undersøkte programmeringsoppgavene i svenske lærebøker i matematikk for barneskolen. Deres forskningsspørsmål inkluderte hva som karakteriserte programmeringsoppgavene i lærebøkene og hvordan programmering og matematikk ble knyttes sammen i programmeringsoppgavene. Ettersom Bråting og Kilhamn (2022) undersøkte noe som jeg var interessert i, valgte jeg å ta utgangspunkt i deres analyseverktøy for å undersøke et lignende tema. For at analyseverktøyet kunne brukes for et høyere trinn, måtte jeg gjøre noen endringer på det. Jeg tilpasset analyseverktøyet så det kunne brukes for å undersøke lærebøker i matematikk 1T.

Matematikk 1T har ett kompetansemål som handler om programmering. Det er: «Formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsningsstrategier, digitale verktøy og programmering» (Kunnskapsdepartementet, 2020). Ettersom programmering ikke blir knyttet til et spesifikt tema i kompetansemålet, tenkte jeg at det ville være spennende å undersøke hvilke matematiske temaer programmering blir knyttet til. Videre tenkte jeg at det ville være spennende å se om programmeringsoppgavene i disse temaene er knyttet til noen sentrale ideer («big ideas») i matematikken. Gjennom lektorstudiet har jeg blitt kjent med to undervisningsmetoder for å lære programmering, PRIMM og UMC. Disse metodene har blitt skrevet om av henholdsvis Flø (2021) og Maugesten et al. (2021) i Tangenten, som er et tidsskrift for matematikkundervisning. Vi kan dermed anta at lærebokforfattere er kjent med metodene. Derfor er det interessant å undersøke om programmeringsoppgavene i boka blir knyttet til disse metodene. Etter jeg videreutviklet og skjerpet forskningsspørsmålene til Bråting og Kilhamn (2022) kom jeg frem til de to følgende forskningsspørsmålene:

(1) Hva karakteriserer programmeringsoppgaver i lærebøker i matematikk 1T

og hvordan knyttes oppgavene til PRIMM og UMC?

(2) Hvilke temaer knyttes programmeringsoppgavene i matematikk 1T til og på hvilke måter knyttes programmering og matematikk sammen?

For å svare på forskningsspørsmålene har jeg analysert programmeringsoppgavene i tre lærebøker for matematikk 1T. Ettersom denne masteroppgaven kun undersøker programmeringsoppgaver, vil «oppgaver» alltid henvise til «programmeringsoppgaver». Hvis jeg henviser til andre typer oppgaver, vil det bli tydelig presisert.

Formålet med studien er å bidra til mer kunnskap om programmering i matematikk 1T og å kartlegge styrker og svakheter i oppgaver. Økt kunnskap om karakteristikkene til programmeringsoppgaver kan bidra til at lærere blir bedre rustet til å gjennomføre eventuelle justeringer og endringer i oppgaven. Dette kan videre bidra til å øke elevers læringsutbytte i matematikk. Formålet er også å få kjennskap til matematiske temaer som kan være hensiktsmessige å knytte programmering til. I denne masteroppgaven undersøkes hva lærebokforfatterne mener programmering kan passe til. Det er ikke noe fasit på hva programmering burde knyttes til, men studien vil kunne gi innsikt i muligheter. Ved å se på tre ulike tilnærminger til programmering i matematikk 1T, kan lærere gjøre seg opp egne meninger om hvilke temaer og oppgaver som knytter programmering og matematikk sammen på en hensiktsmessig måte. Ved å reflektere over hvilke oppgaver som knytter programmering og matematikk sammen på en hensiktsmessig måte, vil man kunne ta et mer gjennomtenkt valg om hvilke oppgaver man vil inkludere eller sette søkelys på i undervisningen. Slik kan man bidra til å øke elevers læringsutbytte i matematikk og man vil bidra til at elevene kan nå visse kompetansemål. Spesielt kompetansemålet knyttet til programmering.

Det finnes flere undersøkelser som har gjennomført tekstbokanalyser av programmeringsoppgaver i matematikk, men mange av undersøkelsene ser på oppgaver på barne- og ungdomstrinn (Bråting & Kilhamn, 2022; Engen, 2022; Stokkenes, 2022). Dermed vil det det være informerende å gjennomføre en undersøkelse på et høyere trinn. Da jeg ikke fant noen studier med like forskningsspørsmål som denne studien, vil undersøkelsen kunne bidra til å gi ny kunnskap på temaet. Momentene nevnt hittil er, i tillegg til å være formålet for studien, også knyttet til profesjonsrelevans da det er kunnskap som er nyttig å ta med videre i arbeidslivet.

Som diskutert i de forrige avsnittene kan denne masteroppgaven bidra til at lærere får mulighet til å øke elevers læringsutbytte i matematikk, samtidig som det gir relevante ferdigheter for fremtiden. Dette har relevans for bærekraftig utvikling i lys av FNs bærekraftsmål: «God utdanning», som sier at vi skal «sikre inkluderende, rettferdig og god utdanning og fremme muligheter for livslang læring for alle» (FN, u.å.). Selv om programmering er nytt i LK20, har elevene rett til en god kvalitet på utdanningen. Derfor er det viktig at både vi som lektorstudenter og ferdigutdannede lærere setter oss inn i teori om læring av programmering. Samtidig burde vi forholde oss noe kritisk til oppgaver i lærebøkene. Hvis lærere setter seg godt nok inn i programmering kan en bidra til at elevene får mulighet til å oppnå relevant digital kompetanse for fremtiden.

Jeg vil videre presentere kort hvordan oppgaven er bygd opp. I kapittel 2 skal jeg presentere teori innen programmering i matematikk, før jeg presenterer det teoretiske rammeverket jeg har tatt utgangspunkt i for studien. I kapittel 3 presenteres metode for

datainnsamling og i kapittel 4 presenteres analyseprosessen og lærebokanalysens resultater. Resultatene diskuteres opp mot teori i kapittel 5, hvor jeg også vurderer rammeverket og peker på studiens begrensninger. Kapittel 6 inneholder oppgavens konklusjon.

# 2. Teori

I dette kapittelet vil jeg presentere relevant teori brukt til å svare på forskningsspørsmålene. To sentrale begreper er programmering og algoritmisk tenking (AT). Ettersom rammeverket jeg skal ta utgangspunkt i er laget for blokkbasert programmering og lærebøker for matematikk 1T bruker tekstbasert programmering, så vil jeg presentere forskjellen mellom de to programmeringstypene. Det kan også finnes oppgaver med analog programmering i lærebøkene, så derfor vil jeg også presentere analog programmering. For å få mer innsikt i programmeringens plass i skolen vil jeg presentere programmering fra et historisk blikk før jeg går videre til en begrunnelse av hvorfor programmering burde være en del av matematikkfaget og hvordan programmering er knyttet til LK20 og kompetansemålene i 1T. Deretter vil jeg ta for meg AT og se hvordan det knyttes til kjerneelementer i læreplanen for matematikk 1T og hva som er forskjellen mellom AT og programmering. Ettersom studien er en innholdsanalyse av lærebøker, vil jeg også presentere teori om lærebokens rolle i matematikk. I det første forskningsspørsmålet skal jeg undersøke hvordan programmeringsoppgavene kan knyttes til PRIMM og UMC. Dermed vil jeg presentere de to didaktiske modellene, PRIMM og UMC, hver for seg. For å kunne svare på forskningsspørsmålet om hvordan programmeringsoppgavene knyttes til matematikk, vil det være nyttig å se på sentrale ideer i matematikk og derfor vil jeg redegjøre hva sentrale ideer er. Mot slutten av kapittelet presenterer jeg rammeverket fra Bråting og Kilhamn (2022) som er utgangspunktet for analyseverktøyet i oppgaven og jeg vil redegjøre for de to rammeverkene analyseverktøyet er bygd opp av: Benton et al. (2016) sitt rammeverk for handling; 5E og Brennan og Resnick (2012) sitt rammeverk for AT. Helt til slutt vil jeg begrunne valget av rammeverket for denne studien.

## 2.1 Blokkbasert programmering, tekstbasert programmering og analog programmering

Programmering er et begrep som defineres på flere ulike måter. En vanlig definisjon på programmering er at datamaskinen instrueres til å utføre ulike oppgaver. Slike instruksjoner kalles ofte for algoritmer (Lindsø, 2020). Ifølge Sevik (2016) er programmering mer enn bare å skrive programkode. Det er også prosessen med å komme frem til denne koden. Altså, prosessen fra å identifisere 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 koden.

Det finnes tre ulike typer programmering som er relevant for utdanning: blokkbasert-, tekstbasert- og analog programmering. Blokkbasert programmering, heretter kalt BBP, er en måte å programmere på ved hjelp av blokker der man for å gjøre en bestemt handling drar passende blokker inn i et program. Med tekstbasert programmering, heretter kalt TBP, må du derimot skrive kode med tegn for å utføre handlinger. Analog programmering innebærer derimot å programmere uten en datamaskin (Statped, 2021). Det kan blant annet være å skrive pseudokode, altså en tekstlig og uformell beskrivelse av et program, der teksten til en viss grad er bygget opp slik programkoden ville vært skrevet. Et annet eksempel kan være å bruke omgivelsene og seg selv for å lære hva ulike programmeringsbegreper betyr. Det finnes mange eksempler på hvordan man kan lære barn og ungdom både programmeringsbegreper og prinsipper fra informatikk analogt.

Noen eksempler kan man finne på csunplugged.org. På barneskolen brukes ofte BBP i undervisning, men på ungdomsskolen begynner overgangen til TBP (Statped, 2023). Eksempler i lærebøker på barne- og ungdomsskolen bruker ofte Scratch som programmeringsmiljø, men det finnes mange miljøer der man kan kode BBP. Mange av de eksisterende blokkbaserte programmeringsmiljøene har en del like trekk. Blant annet Scratch, Alice, Lego Mindstorms og Blockly bruker en blokkstruktur der blokker av kode kan settes sammen akkurat som legobrikker (Moors et al., 2018, s. 58). På samme måte som legobrikker, kontrollerer programmeringsmiljøene at bare kommandoer som passer sammen kan settes sammen. De fleste miljøer har noen trekk som skiller de, men det finnes noen nøkkeltrekk som er delt mellom de fleste blokkbaserte programmeringsmiljøene. Disse nøkkeltrekkene avdekkes når man sammenligner BBP og TBP. Det som skiller BBP fra TBP er at de fjerner behovet for å memorere syntaks, de har en blokkaktig struktur, de bruker farger og former aktivt for å skille mellom konsepter og de har et forenklet språk som minner mer om menneskelig språk (Moors et al., 2018, s. 58). Det som oftest blir sett på som det vanskeligste med å lære TBP er nivået av fokus som trengs for å holde styr på syntaks. «Studenter må først bli mestere på syntaks, før de kan bli problemløsere» (Moors et al., 2018, s. 58). Den største forskjellen mellom BBP og TBP og analog programmering er at BBP og TBP krever en datamaskin mens analog programmering kan gjøres uten en datamaskin.

## 2.2 Historisk blikk på programmering i skolen og i matematikk

At programmering kan være nyttig for å lære matematikk er ikke en ny ide. I 1966 lagde Seymour Papert, Wallace Feurzeig, Daniel Bobrow og Cynthia Solomon et programmeringsspråk kalt LOGO, som var det første programmeringsspråket spesifikt designet for barn (Solomon et al., 2020, s. 1). LOGO var mer enn bare et programmeringsspråk, det var et læringsmiljø hvor barn kunne utforske matematiske ideer og lage prosjekter med deres eget design.

Papert (1980, s. 6) hevder at å programmere en datamaskin betyr å kommunisere med den på et språk som både den og den menneskelige brukeren kan «forstå», og å lære språk er en av tingene barn gjør best. Med få unntak så lærer alle barn å snakke. Hvorfor skal da ikke et barn lære å «snakke» med en datamaskin? En måte man kan kommunisere med en datamaskin er å «snakke» med Turtle. Turtle er en datamaskinstyrt virtuell skilpadde som eksisterer i LOGO miljøet. Skilpadden er laget for å være lett å programmere med og god å tenke med. Skilpadden brukes ofte for å tegne figurer da man kan gi kommandoer som «forward», «right», «pendown» og «penup». Når barn jobber med sånne elektroniske tegneblokker så lærer de et språk for å snakke om former og kombinasjoner av former, om hastigheter og endringsrate og om prosesser og prosedyrer. Elevene lærer å snakke matematikk og de utvikler seg selv som matematikere (Papert, 1980, s. 11-13).

Etter utgivelsen av Papert sin bok «Mindstorms», i 1980, og innføringen av rimelige personlige datamaskiner ble LOGO og Logo-miljøet godt kjent for lærere. Logo-miljøet handlet om at elever, individuelt eller i gruppe, skulle arbeide med et prosjekt etter eget valg. Det skulle bli satt av tid til å reflektere over tankegang og arbeidsprosess og læreren skulle heller være en hjelper enn en autoritet. At mange ble kjent med Logo over

en kort tidsperiode førte til at lærere tok i bruk LOGO uten å ha vært på lærerforberedelsesworkshops i regi av LOGO-utviklerne. Dette førte til at lærere så på LOGO på en annerledes måte enn det utviklerne hadde i tankene. Lærerne så på det som en måte å myndiggjøre barn fremfor at det handlet om å lære matematikk. Det var ulike meninger blant lærere om denne arbeidsmåten, men noen likte det godt og begynte å prate og skrive om «the Logo philosophy». Dessverre handlet det ikke om å gjøre matematikk versus å lære om matematikk, men nettopp om elevsentrert læring (Solomon et al., 2020, s. 55-56).

Samtidig som at mange lærere så på LOGO som å myndiggjøre barn fremfor at det handlet om å lære matematikk, kom det internasjonal forskning som gjorde at programmering i skolen mistet sin kraft. Det ble gjort en rekke undersøkelser utover 1980-tallet og starten av 90-tallet som undersøkte overføringsverdien av det å lære programmering. Undersøkelsene viste at det var vanskelig å finne støtte for at de som lærte å programmere gjennomgikk noen kognitiv trening som var bedre enn annet skolearbeid. Samtidig var det vanskelig å finne støtte for at programmering kunne bedre elevers problemløsning ettersom det er vanskelig å avgrense og definere. I tillegg fant man ikke støtte for at programmeringskompetanse ga fordeler i andre emner (Dolonen et al., 2019, s. 6). På bakgrunn av de kritiske undersøkelsene, ble ikke programmering i skolen så utbredt som Papert hadde ønsket.

### 2.2.1 Norsk historie

I Norge ble det på sytti- og åttitallet gjennomført en rekke forsøk med programmering i skolen. Blant annet kom EDB (elektronisk databehandling) inn som valgfag på videregående. Dette medførte ikke den revolusjonen i pedagogikken som enkelte hadde sett for seg og programmering ble lenge en aktivitet for spesielt interesserte (Sevik, 2016, s. 8). Programmering kom litt tilbake da mønsterplanen for grunnskolen M87. kom. Da ble datalære et av hovedemnene i matematikken. I mønsterplanen M87 står det at «elevene bør møte datateknologien i matematikkopplæringen i skolen først og fremst gjennom formulering av løsningsmetoder som det passer å bruke datamaskin til. Datalære i matematikk tar utgangspunkt i algoritmebegrepet og knyttes nær til begrepet problemløsning» (Kirke- og undervisningsdepartementet, 1991, s. 203). Videre står det at elevene i 7.-9. trinn skal tolke data og trene i å trekke ut informasjon, jobbe med kjente algoritmer fra elevenes nærmiljø og fra andre emner i matematikk og innføre algoritmer som passer for bruk av datamaskin til løsning av kjente oppgaver. Blant annet til tilnærmingsmetoder, simulering og algoritmer for bruk av lommeregner (Kirke- og undervisningsdepartementet, 1991, s. 203). Det finnes lite programmering i læreplanene i matematikk mellom M87 og LK20. Dette gjelder læreplanen for den videregående skolen, Reform 94 (R94) (Utdanningsdirektoratet, 2011), læreplanen for grunnskolen, Reform 97 (L97) (KUF, 1996) og læreplanverket for Kunnskapsløftet 2006 (LK06) (Kunnskapsdepartementet, 2006). I neste avsnitt vil jeg presentere nyere historie og se på hvorfor programmering kom inn i Kunnskapsløftet 2020.

## 2.3 Programmering i matematikkfaget

En stor grunn til å lære barn og unge programmering har et opphav i ønsket om å utdanne flere til et arbeidsliv som blir stadig mer avhengig av teknologisk kompetanse

(Sevik, 2016, s. 11). I perioden mellom 2011 og 2013 ble det uttrykt kritikk for hvordan digitale ferdigheter ble omhandlet i læreplaner, både i utlandet og i Norge. Sentrale personer fra IKT bransjen samt politikere mente at datidens læreplaner ikke gjenspeilte fremtidens behov for IKT kompetanse og spesielt produksjon og sikring av datatjenester gjennom programmering (Dolonen et al., 2019, s. 15). I 2016 ble en ekspertgruppe satt ned av Utdanningsdirektoratet ferdig med rapporten «Teknologi og programmering for alle» (Sanne et al., 2016). Gruppa foreslo at det burde opprettes et nytt obligatorisk teknologifag i grunnskolen. Faget burde omfatte teknologi og programmering, være praktisk og AT skulle være sentralt i faget. Ekspertgruppa diskuterte om programmering burde være et eget fag eller bli integrert i eksisterende fag og kom frem til at de eksisterende fagene ikke alene kunne dekke hele spekteret av teknologisk kunnskap. De mente at en fornyelse av eksisterende fag er nødvendig, men ikke tilstrekkelig (Dolonen et al., 2019, s. 19-20). Ekspertgruppen hevdet at dersom teknologi og programmering blir integrert i eksisterende fag så kan det føre til at temaet blir systematisk nedprioritert. Det kan skje både fordi det ikke passer godt nok inn i fagets kultur, men også fordi lærere ikke opplever å ha nok kompetanse (Sevik, 2016, s. 25). Forfatterne av rapporten fra Senter for IKT stilte seg også bak anbefalingen fra ekspertgruppa om at programmering burde innføres som eget obligatorisk fag (Sevik, 2016, s. 26).
Til tross for resultatene i begge rapportene valgte Utdanningsdirektoratet at programmering skulle integreres i de obligatoriske fagene matematikk, naturfag, musikk og kunst- og håndverk fra høsten 2020, samtidig som at det fortsatt ville være et valgfag i ungdomsskolen. Dolonen et al. (2019, s. 26) finner få argumenter for å innlemme programmering i andre fag, men hevder at Utdanningsforbundet har nevnt at de ikke ønsket å etablere et eget skolefag, samt at de andre nordiske landene hadde samme modell. Ifølge Dolonen et al. (2019, s. 25-26) er hovedargumentasjonen for å få inn programmering i skolen at:

- Programmering kan bidra til økte digitale ferdigheter og forståelse som gjør at man kan ta mer informerte beslutninger knyttet til digitalisering av sektorer.
- Programmering som problemløsning overlapper med kritisk tenkning.
- Programmering er en naturlig del av problemløsning i STEM- fag (vitenskap, teknologi, ingeniørfag og matematikk) og yrker.

I neste avsnitt vil jeg presentere noe av det som ble nytt i matematikk 1T etter at LK20 trådte i kraft.

### 2.3.1 Fagfornyelsen og matematikk 1T

I matematikk 1T er kompetansemålene nøye utformet rundt fagets seks kjerneelementer (Kunnskapsdepartementet, 2020). Kjerneelementene er:

- utforsking og problemløsing
- modellering og anvendelser
- resonnering og argumentasjon
- representasjon og kommunikasjon
- abstraksjon og generalisering
- matematiske kunnskapsområder

En tydelig trend i de nye læreplanene er reduksjonen av temaer per år sammenlignet med tidligere læreplaner. Dette er gjort med den hensikt å sikre progresjon og tilstrekkelig tid til å utforske sammenhenger i faget. I matematikk 1T, som er et teoretisk matematikkfag, vektlegges spesielt teoretiske verktøy, problemløsning og resonnering, som er fundamentale for videre studier (Utdanningsdirektoratet, 2019b). Sammenlignet med tidligere læreplaner er programmering nå spesifikt nevnt i et av kompetansemålene til matematikk 1T. Ifølge Kunnskapsdepartementet (2020) er et av målene for opplæringen at elevene skal kunne:

> «Formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsningsstrategier, digitale verktøy og programmering».

Utdanningsdirektoratet (2019b) hevder at programmering kan gi rom for kreative måter å løse problemer på. Flere fag inneholder kompetansemål knyttet til programmering, men opplæringen i programmering er lagt til matematikkfaget. I matematikkfaget er programmering nevnt spesifikt fra 5. trinn, men det er også et mål å forberede yngre elever for tenkemåten som ligger til grunn for programmering gjennom både analoge og digitale aktiviteter. Dette inkluderer aktiviteter som å lage og følge regler og trinnvise instruksjoner i lek og spill allerede fra 2. trinn (Matematikksenteret, u.å.-b). Senere i skolegangen skal elevene utforske hvordan algoritmer kan utvikles, testes og forbedres, og de skal bruke programmering til å utforske matematiske egenskaper og sammenhenger (Utdanningsdirektoratet, 2019b). Videre hevder Utdanningsdirektoratet (2019b) at digitale verktøy, inkludert programmering, spiller en viktig rolle i å gjøre abstrakte matematiske konsepter mer håndgripelige for elever, og at de kan bidra til en dypere forståelse av matematiske områder.

I tabell 1 finnes en oversikt over kompetansemål i matematikk som er knyttet til programmering og som elevene burde beherske før de tar matematikk 1T.

| | |
|---|---|
| 2. trinn | Lage og følge regler og trinnvise instruksjoner i lek og spill. |
| 3. trinn | Lage og følge regler og trinnvise instruksjoner i lek og spill knyttet til koordinatsystemet. |
| 4. trinn | Lage algoritmer og uttrykke dem ved bruk av variabler, vilkår og løkker. |
| 5. trinn | Lage og programmere algoritmer med bruk av variabler, vilkår og løkker. |
| 6. trinn | Bruke variabler, løkker, vilkår og funksjoner i programmering til å utforske geometriske figurer og mønster. |
| 7. trinn | Bruke programmering til å utforske data i tabeller og datasett. |
| 8. trinn | Utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering. |
| 9. trinn | Simulere utfall i tilfeldige forsøk og beregne sannsynligheten for at noe skal inntreffe, ved å bruke programmering. |
| 10. trinn | Utforske matematiske egenskaper og sammenhenger ved å bruke programmering. |

*Tabell 1 Kompetansemål knyttet til programmering. (Kunnskapsdepartementet, 2019).*

AT (algoritmisk tenking) er også et nytt begrep i læreplanen og er fremhevet fordi det er en viktig problemløsningsstrategi. AT innebærer en systematisk tilnærming til

problemløsning, både i formuleringen av problemene og i forslaget av mulige løsninger. AT handler også om å gi elever effektive verktøy og begreper for å løse problemer, blant annet digitale verktøy (Utdanningsdirektoratet, 2019b). Ifølge Utdanningsdirektoratet (2019a) så finnes det noen viktige nøkkelbegrep som inngår i AT, i tillegg til typiske arbeidsmåter den algoritmiske tenkeren bruker for å løse problemer. Dette er illustrert i figur 1. I denne masteroppgaven har jeg tatt utgangspunkt i et annet rammeverk enn Utdanningsdirektoratet (2019a) sin modell for AT, som blir presentert i neste delkapittel. Der vil jeg presentere mer om AT og se på sammenhengen mellom nøkkelbegrepene fra figur 1 og begrepene i rammeverket jeg har tatt utgangspunkt i.


*Figur 1 Den algoritmiske tenkeren (Utdanningsdirektoratet, 2019a).*

## 2.4 Rammeverk for algoritmisk tenking (AT) for denne studien

AT er et begrep som er oversatt fra det engelske begrepet «computational thinking» (Gjøvik & Torkildsen, 2019, s. 32). Ifølge Wing (2006) så er AT en grunnleggende ferdighet for alle, ikke bare for de som jobber innenfor datateknologi. Hun mener at AT burde bli lagt til som en del av hvert barns analytiske evne, ved siden av lesing, skriving og regning. AT innebærer å løse problemer, designe systemer og forstå menneskelig atferd ved å dra nytte av begreper som er grunnleggende for datavitenskap (Gjøvik & Torkildsen, 2019, s. 32). Videre hevder Gjøvik og Torkildsen (2019, s. 32-33) at det er fem begreper som inngår i AT:

(a) **Abstraksjon:** Handler om å kunne trekke ut essensen av flere eksempler eller tilfeller og se bort fra irrelevante opplysninger.

(b) **Algoritmebehandling:** Handler om å følge og forklare trinnvise instruksjoner.

(c) **Generalisering:** Handler om å gjenkjenne mønstre og sammenhenger og lage allmenne regler og metoder som fungerer på en hel klasse av eksempler.

(d) **Automatisering:** Handler om å kunne implementere løsningen av problemer i et programmeringsspråk eller å gjøre det menneskelige bidraget minimalt.

(e) **Dekomponering:** Handler om å kunne bryte opp et problem i mindre bestanddeler og håndtere hovedproblemet i mindre biter.

Det er verdt å merke seg at begrepet algoritmebehandling er oversatt fra det engelske begrepet «algoritmic thinking», så man må være bevisst på at det ikke fullstendig overensstemmelse mellom norske og utenlandske termer. I tillegg vil også finne forskjellige definisjoner i forskjellig litteratur. Begrepene som inngår i AT er også relatert til et sett med holdninger og ferdigheter og inkluderer å kunne lage beregninger, teste og debugge, samarbeid og kreativitet og evnen til å håndtere åpne problemer. Samtidig vil AT også ha potensiale som et middel for kreativ problemløsning og innovative tilnærminger innen flere disipliner (Bocconi et al., 2022, s. 27). Det er viktig å huske at AT er en måte mennesker, ikke datamaskiner, tenker på. AT skal ikke få oss til å tenke som en datamaskin (Wing, 2006), men få oss til å tenke som en informatiker (Utdanningsdirektoratet, 2019a). Gjennom vår intelligens og med hjelp av datamaskiner så håndterer vi problemer som før var veldig tungvint uten datamaskinen. Samtidig bygger vi systemer med funksjonalitet begrenset kun av vår fantasi (Wing, 2006).

Som nevnt tidligere er rammeverket til Gjøvik og Torkildsen (2019) annerledes enn Utdanningsdirektoratet (2019a) sine nøkkelbegrep (figur 1). Allikevel kan man hevde at de omhandler de samme prinsippene. Derfor vil jeg nå sammenligne de to modellene. I begge modellene finner vi igjen dekomponering og abstraksjon. Algoritmer kan knyttes til algoritmebehandling ettersom algoritmebehandling handler om å følge og forklare trinnvise instruksjoner. Evaluering kan også knyttes til algoritmebehandling ved at det knyttes til algoritmisk resonnering som er en undergruppe av algoritmebehandling. Algoritmisk resonnering inngår i definisjonene av kreativ og imitativ resonnering, og imitativ resonnering går ut på å resonnere enten ved løsninger eller løsningsmetoder man kjenner fra før av (Gjøvik & Torkildsen, 2019, s. 33). Mønstre kan lett knyttes til generalisering og logikk kan knyttes til dekomponering da det vil være nødvendig å bryte opp et problem i mindre deler for å kunne analysere og forutse ulike utfall. Automatisering handler om å implementere løsningen av problemer i et programmeringsspråk, så man kan se på automatisering som det som skiller AT fra å være en ren problemløsningsmetode som kan brukes i flere grener av matematikk til å være en problemløsningsmetode knyttet til programmering.

### 2.4.1 Sammenheng mellom AT og kjerneelementer i 1T

Hvis man ser på kjerneelementene for matematikk 1T, ser man at de fleste av disse kjerneelementene kan knyttes til begrepene innenfor algoritmisk tenking (AT). Videre kommer jeg til å sammenligne AT begrepene med kjerneelementene fra Kunnskapsdepartementet. Alle disse kjerneelementene baserer seg på læreplanen fra

Kunnskapsdepartementet (2020). Det første kjerneelementet i 1T består av utforsking og problemløsning. Her står det spesifikt at AT er viktig i prosessen med å utvikle strategier og fremgangsmåter for å løse problemer og at det innebærer å bryte ned et problem i delproblemer som kan løses systematisk. I tillegg står det at det innebærer å vurdere om delproblemene best kan løses med eller uten digitale verktøy. Det andre kjerneelementet er modellering og anvendelser. Dette kan knyttes til begrepet abstraksjon fra AT ved at elevene må trekke ut essensen fra en modell og se bort ifra irrelevante opplysninger i modellen. Det tredje kjerneelementet er resonnering og argumentasjon. Det kan knyttes til algoritmebehandling, da algoritmebehandling innebærer algoritmisk resonnering (Gjøvik & Torkildsen, 2019, s. 33). Det fjerde kjerneelementet er representasjon og kommunikasjon. Under kjerneelementet står det at elevene må forklare og begrunne valg av representasjonsform og dette kan knyttet til AT ettersom en representasjonsform kan være et program elevene har skrevet selv eller et program elevene må forklare. Det femte kjerneelementet er abstraksjon og generalisering og er nøyaktig det samme som to av begrepene i AT. Det siste kjerneelementet er matematiske kunnskapsområder og er ikke direkte knyttet til AT. I følge Gjøvik og Torkildsen (2019, s. 34) så kan programmering være en relevant og fremtidsrettet måte å arbeide med matematikken på, på de fem første kjerneelementene. Altså kan programmering være et naturlig miljø for å implementere AT i matematikken (Gjøvik & Torkildsen, 2019, s. 33).

### 2.4.2 Sammenheng mellom AT og programmering

Programmering og AT har en toveis relasjon. Programmering (og koding) gir mulighet for undervisning og læring av AT, det gjør AT begrepene mer konkrete. Mens AT gir programmering en ny oppgradert rolle i undervisning. Skillet mellom AT og programmering er i prinsippet ganske subtilt. AT krever ikke nødvendigvis programmering, selv om å representere en løsning til et problem gjennom et program gir en perfekt måte å evaluere løsningen. En datamaskin vil kunne utføre instruksjonene og dermed gi elevene mulighet til å forbedre løsningen slik at den er veldig presis, for eksempel kan man finne en numerisk tilnærming av den deriverte med tilstrekkelig nøyaktighet for skolen. Så AT handler ikke nødvendigvis om programmering, men om problemløsning som fremmer læring (Bocconi et al., 2022, s. 27).

AT kan dermed forstås i tilknytning til programmeringsferdigheter eller i tilknytning til tankeprosesser som er essensielle for problemløsning. Mye av tidligere forskning om AT har handlet om å lære programmering for å tilegne seg ferdigheter innenfor AT. Ofte er vektleggingen av disse to synene kontekstavhengige. I studier som handler om bruk av datamaskiner, ser man at begrepene innenfor AT handler om programmeringsferdigheter, mens i andre studier så blir tenkeferdigheter vektlagt (Bocconi et al., 2022, s. 27). Allikevel er det en generell konsensus at AT ikke bare er en problemløsningsprosess. Løsningen på problemet burde uttrykkes på en måte som tillater at en datamaskin kan utføre den (Bocconi et al., 2022, s. 25). I figur 2 ser vi en oversikt over begreper som er assosiert med hver av de to synene.

| Computation Thinking associated with generic problem solving | Computation Thinking associated with programming and computing |
|---|---|
| Abstraction | Algorithmic Thinking |
| Data Analysis | Algorithm Design |
| Data Collection | Automation |
| Data Representation | Boolean Logic |
| Decomposition | Computation |
| Efficiency | Computational Modelling |
| Evaluation | Conditionals |
| Generalisation | Data Types |
| Logics & Logical Thinking | Events |
| Modelling | Functions |
| Patterns & Pattern Recognition | Iteration |
| Repeating Patterns | Loops (Repetition) |
| Simulation | Modularisation |
| System Thinking | Parallelisation |
| Visualisation | Sequencing |
| | Testing & Debugging |
| | Threads (Parallel Execution) |

*Figur 2 Begreper knyttet til AT som problemløsning eller i tilknytning til programmeringsferdigheter (Bocconi et al., 2022, s. 27).*

Ved å sammenligne AT begrepene fra rammeverket brukt i oppgaven, altså rammeverket fra Gjøvik og Torkildsen (2019), med begrepene i figur 2, fremkommer det at alle de fem AT begrepene fra Gjøvik og Torkildsen (2019) sin modell finnes i figur 2 innenfor begge retningene. Abstraksjon, generalisering og dekomponering finnes under generisk problemløsning, mens algoritmebehandling og automatisering finnes under programmering og beregning. Vi finner også igjen nøkkelbegrepene fra Utdanningsdirektoratet (2019a) sin modell av den algoritmiske tenkeren (figur 1) innenfor begge retningene. Dette er med på å bekrefte at Utdanningsdirektoratet ser på AT som en problemløsningsmetode som er sterkt knyttet til programmering. Forskjellige land har ulike grunner for å inkludere AT i læreplanen og Bocconi et al. (2022, s. 32) har rapportert at Norge har med AT for å fremme kode- og programmeringsferdigheter og for å fremme problemløsningsferdigheter. Dette i tillegg til å fremme ansettelse i digital sektor og å fremme logisk tenking.

## 2.5 Lærebokens rolle i matematikk

En av hovedmotivasjonene for å undersøke innholdet i lærebøker er den sentrale rollen disse bøkene spiller i utformingen av matematikkundervisningen. Ifølge Kongelf (2015, s. 84) er valget av lærebok ikke bare et administrativt spørsmål, men en kritisk faktor som definerer hva lærere underviser, hvordan de underviser, og hvordan elevene deres lærer. Blant annet vil antall sider innenfor hvert emne være bestemmende for hvor mye tid læreren bruker på stoffet og for elevenes prestasjoner (Kongelf, 2015, s. 84). Samtidig understreker Utdanningsdirektoratet (2005, referert i Kongelf, 2015, s. 84) lærebøkenes viktige rolle når de påstår at «[…] hvis det generelt ønskes forandringer i norsk skole, må også lærebøkene forandres». Ved å utføre detaljerte analyser, inkludert vertikale og

mikronivåanalyser, kan man få innsikt i hvordan læreboken faktisk kan bidra til å utvikle den matematiske kompetansen til elevene (Svingen & Gilje, 2018). Det er også verdt å merke seg at lærebøkene er betydelig mer brukt i matematikk, enn i andre skolefag (Lepik et al., 2015, s. 130). Dette understreker hvorfor en innholdsanalyse av lærebøker i matematikk er nyttig. En forklaring av hva en innholdsanalyse innebærer kommer i metodedelen av oppgaven (kapittel 3.1).

Det er viktig å huske at lærebøkene blir omfattende brukt i matematikkundervisningen, men hvordan de blir brukt varierer betydelig fra klasse til klasse og fra lærer til lærer. Allikevel vil temaer som er inkludert i læreboken sannsynligvis bli presentert i undervisningen, mens de som ikke er inkludert, mest sannsynlig vil bli utelatt (Lepik et al., 2015, s. 132). I Norge er det vanlig at matematikkundervisningen består av at læreren i starten av timen introduserer materiale gjennom helklassediskusjon og at elevene deretter får jobbe med oppgaver fra boka (Lepik et al., 2015, s. 136). Sammenlignet med det internasjonale gjennomsnittet utgjør undervisningen hvor læreren forklarer til hele klassen en forholdsvis liten del i Norge. Norske elever arbeider derimot mye alene med oppgaver i lærebøkene og forklarer svarene sine lite (Kongelf, 2015, s. 84). Lepik et al. (2015, s. 126) hevder også at læreboka ofte er mer viktig enn læreplanen når lærere planlegger undervisningen.

Mye forskning viser at matematikklærere bruker lærebøkene hovedsakelig for oppgaver. I Lepik et al. (2015, s. 144) sin undersøkelse kom de fram til at 69% av de norske lærerne brukte læreboka som eneste kilde til oppgaver i halvparten eller alle undervisningstimene. Videre viser undersøkelsen at 88% av de norske lærerne bruker læreboka som eneste kilde til lekser i hver undervisningstime eller hver andre undervisningstime. Leseren bes være bevisst på at alle funnene om lærebøkenes rolle stammer fra før LK20 ble innført, og at det er mulig at praksisen har endret seg, spesielt med innføringen av dybdelæring i læreplanen.

## 2.6 To didaktiske modeller for læring og undervisning av programmering

Gjennom årene har det kommet mange læring- og undervisningsstrategier om programmering. To strategier er PRIMM og Use-Modify-Create (UMC), der UMC har påvirket utviklingen av PRIMM (Sentance et al., 2019, s. 5). Det finnes som nevnt flere undervisningsstrategier, men på bakgrunn av tidsrammen til masteroppgaven velger jeg å fokusere på PRIMM og UMC. Grunnen til at jeg velger akkurat PRIMM og UMC er fordi begge kan diskuteres opp mot Bråting og Kilhamn (2022) sine handlinger i programmeringsoppgaver. I tillegg kan begge knyttes til utvikling av AT (Sentance et al., 2018, s. 115; Matematikksenteret, u.å.-a). PRIMM og UMC er også kjent av en god del matematikklærere, ettersom det er publisert flere artikler om strategiene i Tangenten, som er et tidsskrift for matematikkundervisning. Mer om koblingen mellom handlingene og PRIMM og UMC kommer i diskusjonen av forskningsspørsmål 1 (kapittel 5.1). Videre vil jeg gi en kort presentasjon av de to undervisningsstrategiene.

### 2.6.1 PRIMM

PRIMM er en metode for å undervise eller lære programmering. PRIMM-modellen beskriver en sammensatt oppgavetype eller en undervisningssekvens med flere oppgaver som bygger på hverandre (Flø, 2021, s. 7). Metoden er hovedsakelig laget som en prosess som lærere kan bruke for å strukturere undervisning, men den kan også brukes når man lager oppgaver for å strukturere oppgavejobbingen til elevene (Sentance et al., 2019, s. 10). PRIMM står for predict, run, investigate, modify og make. Nedenfor gis en oversikt over hva de ulike trinnene i prosessen innebærer.

(1) Forutse (predict): Elevene skal få utdelt en kode der de skal forutse hva koden gjør.
(2) Kjør (run): Elevene kjører koden så de kan teste sine antagelser.
(3) Undersøk (investigate): Elevene undersøker koden gjennom å forklare, feilsøke eller lignende.
(4) Endre (modify): Elevene endrer koden slik at funksjonaliteten øker. Dette gjøres med en rekke ulike utvidelser med ulik vanskelighetsgrad.
(5) Lag (make): Elevene lager et helt nytt program som bruker samme strukturer, men som løser et nytt program.
(Sentance et al., 2019, s. 10-11).

Ifølge Flø (2021, s. 5-6) er PRIMM en metode som tilbyr stillasbygging rundt elevenes forståelse. Metoden vil også redusere elevenes opplevde vanskelighetsgrad av programmeringsoppgaver. Utviklerne av PRIMM fremmer også nyttigheten av samarbeid gjennom de ulike trinnene, spesielt i trinnene «forutse» og «undersøk» (Sentance et al., 2019, s. 13). PRIMM-metoden bygger på andre didaktiske rammeverk, blant annet UMC (Sentance et al., 2018, s. 114). I neste delkapittel vil jeg presentere UMC.

### 2.6.2 Use-Modify-Create (UMC)

UMC er en metode som kan brukes for å lære eller undervise programmering. Når elever følger rammeverket går de fra å bruke et program til å selv lage et program (Sentance et al., 2019). UMC står for Use (bruk), modify (endre) og create (lag) som representerer arbeidsmetodens tre faser (Maugesten et al., 2021, s. 3). Nedenfor gis en oversikt over hva de ulike fasene går ut på:

(1) Bruk: Her får elevene et kodeeksempel, der de skal undersøke hvilke funksjoner programmet har og hvordan det er oppbygd.
(2) Endre: Elevene må forstå koden før de endrer deler av den. De må utforske matematikken, samtidig som de lærer programmering. Her kan de endre mindre eller større deler av koden, alt ettersom hvor trygge de er på programmering.
(3) Lag: Elevene bruker kunnskapen de har lært om programmet de har brukt og endret, til å programmere et liknende program.
(Maugesten et al., 2021, s. 3).

I følge Sentance et al. (2018, s. 114-115) vil arbeidsmetoden UMC kunne føre til at elevene utvikler sin AT. Et annet positivt aspekt ved UMC er at det lar elevene utvikle seg fra en bruker til en produsent. Hvis elevene lærer seg å forstå og bruke andres program før de skriver sitt eget, så blir det mer naturlig for elevene å skrive gode strukturerte

program senere.

## 2.7 «Big ideas»

Lærere blir mer og mer oppfordret til å lære bort «big ideas» (heretter omtalt som sentrale ideer), men hvis man spør en gruppe matematikklærere for eksempler av sentrale ideer så kan man få ganske varierende svar. Noen vil si at en sentral ide er et tema, sånn som likninger eller en gren av matematikken, sånn som geometri og noen nevner kompetansemål fra læreplaner. Selv om alle disse er viktige, så er ingen av de robuste nok til å kvalifisere som en sentral ide i matematikken (Charles & Carmel, 2005, s. 9).

> En sentral ide er en ide som er sentral for læringen av matematikk. En sentral ide skal være en påstand, den skal være sentral for læring av matematikk og den skal koble sammen flere matematiske forståelser til en sammenhengende helhet. (Charles & Carmel, 2005, s. 10).

Et eksempel på en sentral ide er påstanden

(1) «numeriske beregninger kan tilnærmes ved å erstatte tall med andre tall som er nærme i verdi og enkle å regne med mentalt. Målinger kan tilnærmes ved å bruke kjente referenter som enheten i måleprosessen».

Dette er en sentral ide, som kan knyttes til temaet numerisk tilnærming (Charles & Carmel, 2005, s. 17). To av de matematiske forståelsene Charles og Carmel (2005, s. 17) hevder hører til denne sentrale ideen er:

(a) «Tallet som er brukt til å lage en tilnærming bestemmer om estimatet er høyere eller lavere enn det eksakte svaret» og
(b) «Referansebrøker som 1/2 (0,5) og 1/4 (0,25) kan brukes til å estimere beregninger som involverer brøker og desimaler».

Det må nevnes at denne sentrale ideen og de matematiske forståelsene den er koblet til er utviklet av Charles og Carmel (2005) og at det ikke er mulig å finne et sett med sentrale ideer som alle matematikere og matematikklærere er enige om. Det vil si at definisjonen på en sentral ide er ganske vag, og dermed krever det selvstendig tenking fra lærere som selv må bestemme hva de ser på som en sentral ide. For å se forslag på flere sentrale ideer i matematikk henvises det til artikkelen av Charles og Carmel (2005). Målet med (Charles & Carmel, 2005, s. 9) sin artikkel var ikke å definere et sett med sentrale ideer som de ville at leserne skulle være enig i, men at artikkelen deres skulle å sette i gang en diskusjon om de sentrale ideene i matematikken. (Charles & Carmel, 2005, s. 9)Selv om det finnes ulike tolkninger av hva en sentral ide er, har Toh og Yeo (2019, s. 9) funnet to fellestrekk som gjelder flere forfatteres forslag til sentrale ideer. Det første fellestrekket er at sentrale ideer handler om forbindelser mellom ulike matematiske ideer. Det andre fellestrekket handler om at sentrale ideer ikke hører til et spesifikt trinn eller kompetansemål, men at ideen er sentral for flere, om ikke alle, klassetrinn. Det fører til at lærere må revurdere deres forståelse av matematikken og matematikkens natur. Ettersom disse fellestrekkene finnes, vil det forhåpentligvis ikke

være så store forskjeller i hva ulike lærere ser på som sentrale ideer, siden å være bevisst på sentrale ideer kan bidra positivt til undervisning. Videre vil jeg presentere noe av argumentasjonen til Charles og Carmel (2005) om hvorfor sentrale ideer kan bidra positivt til undervisningen.

Ifølge Charles og Carmel (2005, s. 10) bør sentrale ideer danne grunnlaget for ens matematikkunnskap, undervisningspraksis og for læreplaner. Hvis man forankrer matematikkunnskapen på et relativt lite antall store ideer, gir det en solid forståelse av matematikken. Ofte forstås noe bedre når det er relatert eller koblet til andre kjente konsepter. Hvor mye vi forstår er knyttet til antall forbindelser og styrken til disse forbindelsene. Nettopp siden sentrale ideer er knyttet til andre ideer så kan forståelsen av sentrale ideer hjelpe elever med å utvikle en dyp forståelse av matematikk. For at elever skal kunne klare å knytte sammen og kommunisere sentrale ideer så er det utrolig viktig at de har et korrekt matematisk språk (Toh & Yeo, 2019, s. 8). Når elever forstår de sentrale ideene så oppfattes ikke lenger matematikken som en samling av isolerte begreper og ferdigheter, men matematikk blir en sammenhengende mengde ideer (Charles & Carmel, 2005, s. 10).

Ettersom lærere ofte følger lærebøkene og tar utgangspunkt i oppgavene fra boka (kapittel 2.6), så vil det være spennende å se om lærebokoppgavene deres er knyttet opp mot sentrale ideer i matematikken, eller om kunnskapen er isolert fra de andre delene av matematikken.

## 2.8 Rammeverk for studien

I denne delen av teorien vil jeg utforske de ulike rammeverkene som danner grunnlaget for analyseverktøyet. Først vil jeg presentere 5E-rammeverket (Benton et al., 2016), etterfulgt av Brennan og Resnick (2012) sitt rammeverk for AT. Deretter vil jeg se på Bråting og Kilhamn (2022) sin tilnærming, som integrerer begge disse rammeverkene. Til slutt vil jeg gi en begrunnelse for hvorfor jeg har valgt å bruke Bråting og Kilhamn (2022) sitt analytiske rammeverk som utgangspunkt for analyseverktøyet i min masteroppgave.

### 2.8.1 Benton et als rammeverk: 5E

5E er et rammeverk som ble utviklet under et prosjekt kalt ScrathMath. ScratchMath hadde som mål å bygge matematisk kunnskap gjennom programmering i Scratch gjennom en et toårig undervisningsopplegg for elever i alderen 9-11 år (Benton et al., 2016, s. 1). Når forskerne gikk gjennom læringsmaterialer for dette prosjektet, utviklet de rammeverket 5E. Målet med rammeverket var å gi pedagogiske strategier for å veilede utformingen av programmeringsaktiviteter i matematikk fra et konstruktivistisk perspektiv (Bråting & Kilhamn, 2022, s. 597). Rammeverket består av de fem begrepene; utforske (Explore), forklare (Explain), forutse (Envisage), dele (Exchange) og brobygging (brigdE). Resultatene fra Benton et al. (2017, s. 136) sine case-studier antyder at 5E er fleksibelt nok for at lærere kan tilpasse begrepene til sin egen undervisningsstil og erfaring, samtidig som de imøtekommer behovene til elevene. Lærere kan formidle sentrale AT ideer og matematiske ideer på ulike måter, samtidig som de holder seg tro mot målet for læringsaktiviteten. Rammeverket må ikke blandes eller

bli forvekslet med læringsmodellen 5E fra BSCS som består av de fem fasene; engasjere, undersøke, forklare, utvide og vurdere (Büyükkarci & Taşlidere, 2023). Rammeverket fra BSCS har ingen betydning for denne masteroppgaven. Nedenfor kommer en forklaring av hva de fem begrepene i rammeverket (fra ScratchMath) innebærer.

1) Utforske: Elevene skal få lov til å utforske ideer, prøve ting selv og feilsøke konseptuelle og tekniske feil når det er nødvendig. Oppgavene skal guide elevene mot å ta kontroll over deres egen læring og å søke etter årsakene bak ulike resultater.
2) Forklare: Elevene skal få forklare hva de har lært og hvorfor de har valgt den løsningsmetoden de har brukt.
3) Forutse: Elevene skal kunne forutse hva programmet gjør før de kjører det. I tillegg skal de sammenligne det de forutså med det faktiske resultatet.
4) Dele: Elevene skal kunne dele antagelser og ideer, de skal kunne argumentere for sine løsningsmetoder, sammenligne med andre og løse uenigheter i grupper.
5) Brobygging: Å finne koblinger mellom programmering og matematiske ideer. For å kunne utvikle eller oppdage disse koblingene må ideer bli satt i en ny kontekst og bli bygget opp på ny med et matematisk språk.

### 2.8.2 Brennan og Resnicks rammeverk for AT

Brennan og Resnick (2012, s. 3) har utviklet et rammeverk for AT som involverer tre dimensjoner. Disse er begreper, praksiser og perspektiver innenfor AT. Rammeverket ble laget etter at Brennan og Resnick (2012, s. 3) studerte aktiviteten i et nettsamfunn for Scratch i tillegg til å observere aktiviteten under Scratch workshops. De identifiserte syv begreper innenfor AT som de fant nyttige i en rekke Scratch prosjekter og som også kan overføres til andre typer programmering. Disse begrepene består av sekvenser, løkker, parallellisme, handlinger, vilkår, operatorer og data (Brennan & Resnick, 2012, s. 3).

Gjennom intervjuer og observasjoner av barna kom de frem til den andre dimensjonen, praksiser innenfor AT. Denne dimensjonen består av praksiser som barna engasjerte seg i når de programmerte. Praksisene kan bli delt inn i fire hoveddeler; være inkrementell og iterativ, teste og feilsøke, gjenbruke og remikse, samt abstrahere og modulere (Brennan & Resnick, 2012, s. 7).

Den siste dimensjonen de kom frem til er perspektiver innenfor AT. Det handler om de perspektivene barna danner om verden rundt dem og om seg selv. Denne dimensjonen består av å uttrykke seg, interaksjon med andre og å stille spørsmål om og med teknologi (Brennan & Resnick, 2012, s. 10-11).

### 2.8.3 Bråting og Kilhamns rammeverk

I 2020 gjennomførte Bråting og Kilhamn (2022, s. 595) en undersøkelse der de analyserte programmeringsoppgavene som finnes i lærebøker for barnetrinnet i Sverige. Målet var å få en karakterisering av programmeringsinnholdet i bøkene og diskutere hvordan innholdet kan påvirke elevenes muligheter til å lære matematikk. Forskningsspørsmålene deres var:

(1) Hva kjennetegner programmeringsinnholdet i svenske lærebøker i skolematematikk?
(2) På hvilke måter skaper programmeringsoppgavene en bro mellom programmering og matematikk?
(Bråting & Kilhamn, 2022, s. 595).

De brukte Brennan og Resnick (2012) sitt rammeverk for AT sammen med Benton et al. (2017) sitt rammeverk for handling; 5E som grunnlag for deres analytiske rammeverk (Bråting & Kilhamn, 2022, s. 595).

I starten av analysen prøvde Bråting og Kilhamn (2022, s. 599) å analysere lærebøkene ved å bruke de to rammeverkene AT og 5E individuelt, men de kom frem til at ingen av rammeverkene kunne forklare innholdet tilstrekkelig. Ettersom rammeverket for AT er et rammeverk for tenking og 5E er et rammeverk for designprinsipper, så er ingen av dem automatisk egnet for analyse av lærebøker (Bråting & Kilhamn, 2022, s. 598). På bakgrunn av dette konstruerte Bråting og Kilhamn (2022, s. 598-599) sitt eget analytiske verktøy som består av handlinger og begreper.

Bråting og Kilhamn (2022, s. s.599) kom fram til 6 handlinger som kan kategorisere innholdet i lærebøker. Disse er presentert og definert i listen nedenfor. I handling (c), (b) og (d) finnes spor av praksiser innenfor AT. Det er henholdsvis feilsøking, være inkrementell og være iterativ. Handling (b) og (c) er knyttet til begrepet utforske fra 5E og (e) og (f) er henholdsvis knyttet til begrepene forklare og forutse fra 5E. Den første handlingen (a) ble inspirert av forfatternes egne data.

a) Følge en regel: Å følge steg-for-steg instruksjoner, repetere eller fortsette på et mønster.
b) Finne regel: Finne ut prosedyren, regelen eller mønsteret som genererer et utfall, for eksempel et tallmønster.
c) Feilsøke: Å feilsøke («debugge») en kode.
d) Forme og skape: Gi instruksjoner, lage et mønster, skrive kode og å representere med symboler.
e) Forklare: Forklare med naturlig språk. Bruke ord for å beskrive en prosedyre, regel, et mønster eller et begrep.
f) Forestille seg: Forutse hva som vil skje. Reflektere over mulige utfall når verdier eller betingelser er endret.

I tillegg til disse handlingene kom de frem til to typer begreper som kan finnes i oppgaver; matematiske begreper og programmeringsbegreper (Bråting & Kilhamn, 2022). For å kategorisere begrepene brukte de deskriptiv analyse, der de så på ordlyden i den gitte konteksten. Deskriptiv betyr beskrivende og det at de gjennomførte en deskriptiv analyse betyr at formålet var å beskrive fordelingen av ulike faktorer (Stoltenberg, 2022), i dette tilfellet matematiske begreper og programmeringsbegreper. Bråting og Kilhamn (2022, s. 599) hevder at innholdet kan bli klassifisert som et matematisk begrep hvis oppgaven, eller tilhørende eksempler og forklaringer, er knyttet til tradisjonell skolematematikk og hvis innholdet formidler en viktig matematisk idé. I deres analyse kom de frem til noen matematiske begreper de fant i lærebokoppgavene. Disse er mønster, geometri, aritmetikk, rotasjon og koordinatsystem.

Programmeringsbegreper ble også kategorisert via en deskriptiv analyse og er: stegvise instruksjoner, algoritmer, kode, regel, løkker, vilkår og feilsøking (Bråting & Kilhamn, 2022, s. 600).

For å kunne svare på hvordan oppgavene knyttes til matematikk, så forskerne på kombinasjonen mellom handlinger og begreper og så etter hvordan matematikk og programmering ble knyttet sammen. Spesielt så de etter om elevene ble bedt om å omformulere programmeringsideer ved å bruke matematisk notasjon eller om oppgavene gjorde at de kunne utforske ukjente matematiske ideer (Bråting & Kilhamn, 2022, s. 600).

### 2.8.4 Begrunnelse av rammeverk

I denne masteroppgaven har jeg valgt å videreutvikle analyseverktøyet til Bråting og Kilhamn (2022). Hovedbegrunnelsen for å ta utgangspunkt i Bråting og Kilhamn (2022) sitt analyseverktøy er at Bråting og Kilhamn (2022) i sin studie har delvis samme forskningsspørsmål som det jeg ønsker å undersøke. I tillegg er rammeverket spesifikt utviklet for å undersøke programmeringsinnhold i matematikkoppgaver. Derfor ble det naturlig at jeg tok i bruk deres analyseverktøy for å analysere lærebokoppgavene. Selv om denne undersøkelsen nesten har de samme forskningsspørsmålene som undersøkelsen til Bråting og Kilhamn (2022), så er det noen betydelige forskjeller i utgangspunktet for analysen. Undersøkelsen til Bråting og Kilhamn (2022) baserer seg på lærebøker fra 1-6 trinn og forskerne ser dermed på oppgaver som er fra et blokkbasert programmeringsspråk. I tillegg er rammeverkene som analyseverktøyet er bygd opp av, basert på programmeringsspråket Scratch som bruker BBP (Bråting & Kilhamn, 2022, s. 598). Allikevel kan man argumentere for at analyseverktøyet også kan brukes for oppgaver som baserer seg på TBP. Det vil være relevant i oppgaver der TBP er brukt å følge en regel, finne regler, feilsøke, forme og skape, forklare og å forestille seg. Det kan hende at det er lettere for elever å for eksempel forklare et program i Scratch enn i et tekstbasert programmeringsspråk ettersom Scratch bruker farger og former aktivt for å skille mellom konsepter (Moors et al., 2018, s. 58). I tillegg kan det også være enklere å lage et program, når man kan flytte blokker av kode og slippe å bry seg om syntaks (Moors et al., 2018, s. 58). Allikevel er ikke dette så relevant for masteroppgaven da denne oppgaven ikke vil ta for seg forskjell i læring mellom TBP og BBP.

Analyseverktøyet ser også på hvordan oppgaver knyttes til AT. I følge Sun et al. (2024, s. 1070) så har programmering blant ulike strategier, vist seg å være en av de effektive måtene å forbedre elevers algoritmiske tenking på, og er dermed ikke spesifikk til hverken BBP eller TBP. Samtidig nevner Brennan og Resnick (2012, s. 3) spesifikt at begreper innenfor AT i BBP kan overføres til andre typer programmering.

Bråting og Kilhamn (2022) har også brukt et annet rammeverk for AT enn det jeg har presentert i denne oppgaven. Jeg vil allikevel hevde at man finner alle begrepene innenfor AT fra Gjøvik og Torkildsen (2019) i handlingene til Bråting og Kilhamn (2022). I tabell 2 finnes det en oversikt over begreper innenfor AT jeg mener hører til de ulike handlingene.

| Handling: | Begreper innenfor AT: | Begrunnelse: |
|---|---|---|
| Følge en regel | Algoritmebehandling | Handler om å følge og forklare trinnvise instruksjoner |
| | Generalisering | Handler om å gjenkjenne mønsteret for å fortsette på det |
| Finne regel | Abstraksjon | Handler om å trekke ut essensen og se bort fra irrelevante opplysninger |
| | Generalisering | Handler om å gjenkjenne mønster og finne regler som fungerer for en hel klasse av eksempler. |
| Feilsøke | Abstraksjon | Ved å se bort fra unødvendige detaljer kan man fokusere på de sentrale elementene i feilen. |
| | Dekomponering | Handler om å dele et større problem inn i mindre, håndterbare deler for å identifisere og løse feil i koden. |
| | Algoritmebehandling | Man må forstå stegene i algoritmen for å avgjøre hvor det finnes feil. |
| Forme og skape | Automatisering | Handler om implementering av løsninger ved bruk av programmeringsspråk eller automatiserte verktøy. |
| | Dekomponering | Man må kunne bryte opp problemet i mindre deler, slik at man kan håndtere en del av gangen. |
| | Algoritmebehandling | Man må omgjøre informasjonen fra oppgaven til tydelige definerte instrukser som datamaskinen forstår. |
| | Abstraksjon | Man må trekke ut essensen av oppgaven for å kunne omgjøre det til instrukser. |
| Forklare | Abstraksjon | Når man gir en kort forklaring på hva et program gjør, skjuler man unødvendige detaljer. |
| Forestille seg | Generalisere | Hvis man forstår sammenhengen i programmet, vil man kunne forstå hva som skjer hvis noen betingelser endres. |
| | Abstraksjon | Handler om å trekke ut essensen og se bort fra irrelevante opplysninger. |

*Tabell 2 Oversikt over hvilke AT-begreper som hører til handlingene fra Bråting og Kilhamn (2022) og en kort begrunnelse på hvorfor begrepene kan knyttes til handlingen.*

Ettersom Bråting og Kilhamn (2022) har laget analyseverktøyet for matematikk på barnetrinn så vil ikke deres matematiske begreper være representative for matematikk 1T. Derfor kommer jeg til å måtte endre begrepene gjennom analysen. Likevel kommer ikke dette til å føre til noen endringer på selve analyseverktøyet. Jeg kommer også til å undersøke oppgavene i lys av sentrale ideer. Det vil ikke endre analyseverktøyet, men heller være en utvidelse. En mer detaljert oversikt over forskjellene mellom Bråting og Kilhamn (2022) sitt analyseverktøy og mitt analyseverktøy blir beskrevet i analyseprosessen for hver delanalyse (4.1). I neste kapittel vil jeg ta for meg oppgavens metode.

# 3. Metode

For å svare på forskningsspørsmålene som er beskrevet i innledningen (1.1) har jeg videreutviklet analyseverktøyet til Bråting og Kilhamn (2022) og gjennomført en innholdsanalyse av tre lærebøker for matematikk 1T. I dette kapittelet vil jeg beskrive metoden jeg har brukt; innholdsanalyse, før jeg presenterer utvalget og forklarer hva som er analyseenhetene. Deretter vil jeg presentere kort analysens struktur og analysens koder. Til slutt vil jeg diskutere reliabilitet, validitet og etiske betraktninger.

## 3.1 Innholdsanalyse

I denne undersøkelsen har jeg systematisk gjennomgått programmeringsinnholdet i tre lærebøker for matematikk 1T med sikte om å finne relevant informasjon om de forholdene som er beskrevet i forskningsspørsmålene. Ettersom jeg har bearbeidet, systematisert og registrert de relevante oppgavene i lærebøkene, slik at oppgavene kan brukes som datagrunnlag i studien, kan studien karakteriseres som en innholdsanalyse (Grønmo, 2004, s. 187). En innholdsanalyse er en «ikke-påtrengende» metode der man genererer data uten at ikke-forskende deltakere er involvert (Tjora, 2018, s. 182). For å analysere oppgavene har jeg tatt utgangspunkt i analyseverktøyet til Bråting og Kilhamn (2022), men gjort noen forandringer. Målet er å undersøke hva som karakteriserer lærebøker i matematikk 1T, se hvordan oppgavene knyttes til PRIMM og UMC, se hvilke temaer programmering knyttes til og hvordan programmering og matematikk knyttes sammen. Selv om jeg har sett på tre læreverk fra tre forskjellige forlag, er ikke utgangspunktet for studiet å sammenligne bøkene. Jeg har likevel valgt å presentere resultatene slik at man ser hvilke forlag resultatene kommer fra. Slik kan man se de ulike måtene programmering er inkludert på, og hvilke konsekvenser dette har for hvordan matematikk og programmering knyttes sammen.

Innholdet i innholdsanalyser kan behandles som enten kvalitative eller kvantitative data. Derfor må man skille mellom kvalitativ og kvantitativ innholdsanalyse (Grønmo, 2004, s. 187). Videre hevder Grønmo (2004, s. 189-193) at både kvalitativ og kvantitativ innholdsanalyse bygger på en systematisk gjennomgang av innholdet i dokumenter, men at de skiller seg fra hverandre på flere andre måter. Blant annet vil utvalget av forskningsdataen i en kvalitativ innholdsanalyse til dels foregå under datainnsamlingen. Samtidig vil problemstillingen bli mer belyst jo mer innhold som studeres, analyseres og tolkes. Det innebærer at datainnsamlingen er lite forutsigbar og så krevende at den må utføres av forskeren selv. Datainnsamlingen kan ikke bygge på en detaljert plan som legges på forhånd, for planene blir fort endret i løpet av datainnsamlingen. Ved en kvantitativ innholdsanalyse vil derimot utvelgingen av tekstene velges på forhånd før datainnsamlingen starter. Innholdet skal bli vurdert i forhold til et strukturert skjema av koder som er utviklet før datainnsamlingen begynner, et såkalt kodeskjema. Med kode menes en forkortelse eller et symbol som brukes om et segment av ord i en setning eller et avsnitt for å klassifisere ordene (Grønmo, 2004, s. 246). Man skal altså ikke utvikle koder underveis i gjennomgangen av tekstene i en kvantitativ innholdsanalyse. Ettersom gjennomgangen av datasettet er mer strukturert ved en kvantitativ innholdsanalyse enn en kvalitativ innholdsanalyse kan datainnsamlingen gjøres av andre enn forskeren (Grønmo, 2004, s. 193)

I denne studien valgte jeg tre lærebøker før jeg begynte på analysen. Jeg tok

utgangspunkt i Bråting og Kilhamn (2022) sitt analyseverktøy og har brukt deres koder for å analysere programmeringsoppgavene. Samtidig var det deler av analysen der jeg endret kodene til Bråting og Kilhamn (2022) i visse delanalyser. Det medfører at jeg ifølge Grønmo (2004, s. 212) har benyttet meg av en kvalitativ innholdsanalyse med kvantitative trekk. Mer om analysens koder kommer i kapittel 3.5.

## 3.2 Utvalg av forskningsdata

For undersøkelsen har jeg valgt tre lærebøker i matematikk 1T. Alle bøkene følger læreplanen i matematikk 1T fra LK20 og inkluderer programmeringsoppgaver. De tre lærebøkene er «Matematikk 1T», «Sinus 1T» og «Mønster 1T». Oversikt over bøkene og deres forfattere og forlag finnes i tabell 3. Grunnen til at de tre lærebøkene ble valgt er fordi forlagene er topp tre på listen av de 50 største forlagene i Norge (Neraal, 2024). Det var også disse lærebøkene jeg hadde hørt om før og hadde tilgang til. Tre bøker virket også passende for omfanget til en masteroppgave. Ettersom navnet til boken fra Aschehoug, «Matematikk 1T», kan bli forvekslet med faget matematikk 1T, vil jeg fremover presentere boken som kun «Aschehoug». Samtidig, ettersom studien kun handler om lærebøker i matematikk 1T, vil jeg presentere «Sinus 1T» som «Sinus» og «Mønster 1T» som «Mønster».

| Navn: | Forfattere: | Forlag: |
|---|---|---|
| Matematikk 1T | Inger Christin Borge<br>John Engeseth<br>Hermod Haug<br>Odd Heir<br>Håvard Moe<br>Tea Toft Norderhaug<br>Sigrid Melander Vie | Aschehoug |
| Sinus 1T | Tore Oldervoll<br>Otto Svorstøl<br>Birte Vestergaard<br>Einar Gustafsson<br>Egil Reidar Osnes<br>Robin Bjørnetun Jacobsen<br>Terje Andreas Pedersen | Cappelen Damm |
| Mønster 1T | Tove Kalvø<br>Jens Christian Lothe Opdahl<br>Knut Skrindo<br>Øystein Johannes Weider | Gyldendal |

*Tabell 3 Oversikt over de tre lærebøkene som er utvalget i oppgaven, deres forfattere og forlaget som har gitt ut bøkene.*

## 3.3 Analyseenheter i utvalget

I studien til Bråting og Kilhamn (2022, s. 600) valgte de å ha programmeringsoppgaver i lærebøkene som analyseenheter. Ettersom jeg har tatt utgangspunkt i deres rammeverk, valgte jeg også å ha programmeringsoppgaver som analyseenheter. Oppgavene jeg har tolket som programmeringsoppgaver er de oppgavene som inneholder ordet «programmering», er tydelig knyttet til programmering eller har et programmeringssymbol ved siden av oppgaven. Ettersom flere av oppgavene var nært knyttet til eksempler i bøkene, så er all materiell i bøkene som er knyttet til programmeringsoppgavene blitt inkludert i analysen. Jeg velger å sette søkelys på oppgavene, men presenterer også eksempler fra bøkene der det er relevant for analysen. Deloppgaver er ikke analysert hver for seg, da jeg er interessert i å finne ut hva som karakteriserer oppgavene som en helhet. I lærebøkene Mønster og Aschehoug finnes en opplæringsmanual om programmering bakerst i boka. Oppgavene i opplæringsdelen er ikke tatt med som analyseenheter da de ikke er knyttet til et spesifikt kapittel i boka og ettersom opplæringsmanualen er tatt med hovedsakelig for at elevene skal få en opplæring i programmering, ikke matematikk.

## 3.4 Analysens struktur

Gjennom analysen fant jeg 63 analyseenheter til sammen i de tre lærebøkene. Analyseenhetene har blitt analysert gjennom tre delanalyser. I den første delanalysen har jeg undersøkt handlinger i oppgavene, i den andre delanalysen har jeg undersøkt programmeringsbegreper og matematiske begreper og i den siste delanalysen har jeg undersøkt koblingen mellom programmering og matematikk i oppgavene. Strukturen av delanalysene følger strukturen i studien til Bråting og Kilhamn (2022). En mer nøyaktig gjennomgang av selve analyseprosessen for hver delanalyse kommer i kapittel 4.1.

## 3.5 Analysens koder

I alle delanalysene har jeg brukt koder for å kategorisere data og analysen har blitt gjennomført både induktivt og deduktivt. En induktiv analyse innebærer at kodene blir utviklet fra datamaterialet og samlet i kategorier og en deduktiv analyse innebærer at kodene er definerte før analysen begynner (Tjora, 2018, s. 18). Hvilke delanalyser som er induktive og deduktive vil komme frem i analyseprosessen (kapittel 4.1) Som nevnt i kapittel 3.1 er kode en forkortelse som brukes om et segment av ord i en setning eller et avsnitt for å klassifisere ordene. En kode for et bestemt avsnitt kan for eksempel angi hvilket tema avsnittet dreier seg om (Grønmo, 2004, s. 246). For eksempel hvilke matematiske begrep oppgavene handler om. I analysen har jeg brukt både deskriptive og fortolkende koder. Deskriptive koder er rent beskrivende karakteristikker av det faktiske og eksplisitte innholdet i teksten, mens fortolkende koder gir et uttrykk for min tolkning eller forståelse av innholdet i teksten (Grønmo, 2004, s. 247). I delanalyse 2 har jeg undersøkt blant annet hvilke programmeringsbegreper som finnes i løsningen av oppgavene, dermed har jeg brukt deskriptive koder. I delanalyse 3 som handler om koblingen mellom programmering og matematikk har jeg derimot undersøkt hvordan jeg synes programmering og matematikk blir koblet sammen. Her undersøker jeg blant annet om matematikk er kontekst for oppgaver eller om oppgaven lar elevene utforske nye matematiske ideer. Jeg har analysert på bakgrunn av mine tolkninger av oppgaven,

og derfor har jeg brukt fortolkende koder i delanalyse 3.

## 3.6 Reliabilitet

Reliabilitet refererer til datamaterialets pålitelighet. Reliabiliteten er høy dersom datainnsamlingen gir pålitelige data (Grønmo, 2004, s. 220). Altså, hvis man gjennomfører undersøkelsen på nytt med de samme kodene, så burde man få samme resultat. En positiv ting med innholdsanalyse, i tillegg til at det er ikke-påtrengende, er at innholdet er permanent og dermed er det mulig å analysere flere ganger for å sjekke reliabilitet. Spesielt etter at man har lagt til nye koder, er det viktig å sjekke om man må kategorisere det som allerede er kategorisert på nytt (Robson & McCartan, 2015, s. 357). For å sikre reliabilitet er det ifølge Tjora (2018, s. 237-238) viktig å redegjøre for fremgangsmåten i forskningsprosessen og beskrive hvordan forskeren kan ha påvirket resultatene. For å oppnå reliabilitet i denne studien har jeg beskrevet rammeverket jeg har tatt utgangspunkt i på en nøyaktig måte i teorien (2.9). I tillegg har jeg forklart hvordan analyseprosessen har foregått for hver av de 3 delanalysene (kapittel 4.1) og jeg har analysert dataene flere ganger når jeg har endret kodene underveis. Som Grønmo (2004, s. 220) hevder, vil det ikke alltid være mulig å få helt likt resultat dersom man gjennomfører undersøkelser på nytt, dette gjelder spesielt ved innsamling av kvalitative data. Som nevnt i forrige avsnitt har jeg brukt noen fortolkende koder i analysen. Det vil si at noen kan tolke programmeringsoppgavene annerledes enn hva jeg har gjort. Allikevel har jeg gitt en del eksempler på oppgaver som jeg har kodet og beskrevet hvorfor jeg har kategorisert de som jeg har gjort. For å sikre reliabilitet har jeg også diskutert med veileder der jeg var usikker på kodingen. Med bakgrunn i forklaringer, støttet av eksempler, tror jeg andre kan gjennomføre studien på nytt og få tilsvarende resultater som presentert i denne studien.

## 3.7 Validitet

Validitet handler om hvorvidt de svarene man finner i forskningen, faktisk er svar på de spørsmålene man ønsker å stille (Tjora, 2018, s. 232). Validiteten i en studie vil dermed være lav dersom fremgangsmåten man har valgt gjør at man ikke får svar på forskningsspørsmålene. Ifølge Grønmo (2004, s. 221) er validitet i første rekke avhengig av hvordan undersøkelsesopplegget er utformet. I tillegg er validiteten relatert til utvelgelsen av analyseenheter og valg av tema. Videre hevder Grønmo (2004, s. 239) at det er mulighet for å forbedre validiteten i kvalitative studier gjennom datainnsamlingen. Det er fordi opplegget er fleksibelt og kan tilpasses og endres etter hvert som undersøkelsen gjennomføres. I tillegg kan man styrke validiteten dersom man tydeliggjør hvordan man utfører forskningen ut fra spørsmålene man stiller. I denne undersøkelsen har jeg som nevnt tatt utgangspunkt i analyseverktøyet til Bråting og Kilhamn (2022). Deres forskningsspørsmål (se kapittel 2.9.3) er ganske like forskningsspørsmålene i denne studien, dermed vil denne studiens validitet være påvirket av deres studies validitet. Et problem med å bruke deres rammeverk har vært at de ser på lærebøker for barnetrinnet og jeg ser på videregående. For å styrke validiteten endret jeg blant annet kodene for matematiske begreper ettersom disse hørte mer til svenske læreplaner for barnetrinnet enn norske læreplaner for matematikk 1T. En mer utdypende begrunnelse for valget av rammeverket er gitt i kapittel 2.9.4. Jeg mener denne undersøkelsen har en god validitet ettersom delanalyse 1 og 2 sier noe om hva som karakteriserer

programmeringsoppgavene og delanalyse 2 og 3 sier noe om hvilket tema programmering knyttes til og hvordan programmering og matematikk knyttes sammen. Svaret på hvordan oppgavene kan knyttes til PRIMM og UMC blir presentert i diskusjonen, der jeg undersøker handlingene fra delanalyse 1 og ser på hvordan kombinasjoner av handlingene kan knyttes til PRIMM og UMC.

## 3.8 Etiske betraktninger

Ettersom studien baserer seg på å analysere lærebøker, blir ingen enkeltpersoner berørt. Allikevel er forfatterne av lærebøkene en tredjepart som er indirekte inkludert i forskningen.

> «Forskere skal ta hensyn til personer som direkte eller indirekte er berørt av forskningen, uten at de selv har gitt samtykke til å delta» (De nasjonale forskningsetiske komiteene, 2021).

Intensjonen med studien har ikke vært å vurdere kvaliteten av lærebøkene eller å ha en direkte sammenligning av lærebøkene. Derfor forsøker jeg å holde meg nøytral til bøkene og deres ulike tilnærminger. Det jeg vil undersøke er det som står beskrevet i forskningsspørsmålene. Før analysen begynte hadde jeg ikke en bok jeg satt høyere enn de andre, derfor fremstiller jeg funnene fra et nøytralt perspektiv. Grønmo (2004, s. 235) hevder at forskeren alltid vil ha et eller annet engasjement i temaet det forskes på og at fullstendig nøytralitet ikke kan eksistere. I diskusjonen vil jeg diskutere resultatene ut ifra teori og tidligere forskning og jeg kommer til å presisere dersom jeg fremmer noe som er min personlige mening.

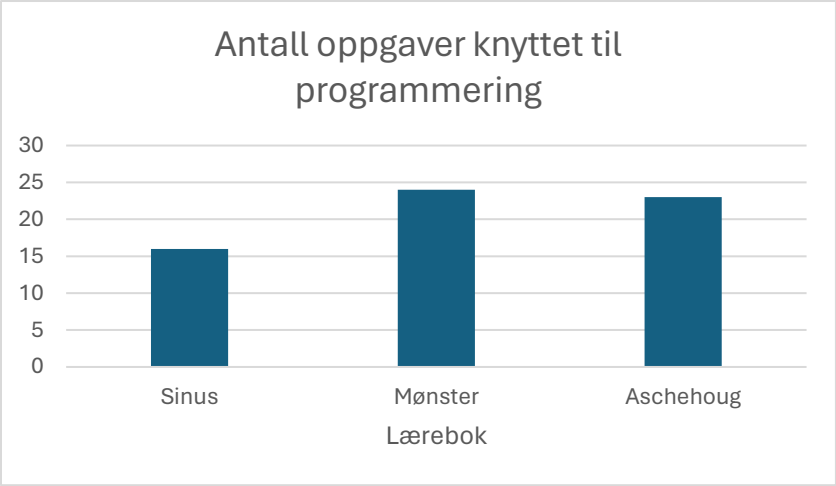
# 4. Analyse og resultater

Analysen består av tre delanalyser. Den første delanalysen omhandler hvilke handlinger som finnes i oppgavene. Den andre delanalysen omhandler hvilke programmeringsbegreper og matematiske begreper som finnes i oppgavene og den siste delanalysen omhandler hvordan programmeringsoppgavene knyttes til matematikk. I det kommende kapittelet skal jeg presentere analyseprosessen. Der skal jeg beskrive hvordan jeg gjennomførte de ulike delanalysene, og hvordan min analyse skiller seg fra Bråting og Kilhamn (2022) sin analyse. Etter analyseprosessen er presentert, presenteres resultatene for hver av de tre delanalysene.

## 4.1 Analyseprosessen

I analyseprosessen har jeg tatt utgangspunkt i analyseverktøyet fra Bråting og Kilhamn (2022) og videreutviklet det. I delanalyse 1 har jeg brukt Bråting og Kilhamn (2022) sine handlinger for å analysere datamaterialet og oppgavenes tilknytning til AT. I delanalyse 2 og 3 gjorde jeg noen endringer på Bråting og Kilhamn (2022) sitt rammeverk. Endringene vil bli diskutert når jeg presenterer analyseprosessen for hver av delanalysene. En årsak til at kodene til Bråting og Kilhamn (2022) måtte endres er at deres rammeverk ser på barneskolen, mens jeg ser på matematikk for VG1. I tillegg er rammeverket laget for svenske læreplaner og ikke norske. En utfordring med å måtte endre kodene var at jeg måtte gå gjennom datasettet på nytt hver gang jeg definerte en ny kode, for å dobbeltsjekke om jeg måtte endre hvordan jeg hadde kodet tidligere analyseenheter. Jeg gjennomførte analysen ved å ta for meg en delanalyse av gangen, der jeg så på en og en bok av gangen. Siden delanalyse 2 består av både programmeringsbegreper og matematiske begreper, så jeg først på programmeringsbegreper før jeg så på matematiske begreper. Til slutt gjennomførte jeg delanalyse 3 som handler om koblingen mellom programmering og matematikk.

Analysen er gjort på 63 oppgaver fordelt på tre lærebøker for matematikk 1T (figur 3). Jeg har analysert hver oppgave som en analyseenhet, uavhengig av antall deloppgaver. Noen av oppgavene har dermed blitt kategorisert innenfor flere handlinger eller begreper. Hvis en oppgave har flere deloppgaver som kan knyttes til samme handling eller begrep, teller jeg de ikke dobbelt. Det er fordi jeg er opptatt av hva som karakteriserer oppgavene som en helhet.




*Figur 3 Oversikt over antall programmeringsoppgaver i Sinus, Mønster og Aschehoug.*

Videre vil jeg presentere hvilke koder jeg brukte til hver av delanalysene og hvordan de eventuelt skiller seg fra kodene i analyseverktøyet til Bråting og Kilhamn (2022).

### 4.1.1 Delanalyse 1 - Handlinger

For å finne ut av hva som karakteriserer programmeringsoppgaver i matematikk 1T har jeg undersøkt hvilke handlinger oppgavene krever at elevene skal gjøre. Ved å undersøke hvilke handlinger som kreves av elevene får man også undersøkt hvilke AT-begreper som kreves av elevene (tabell 2). Jeg valgte å bruke handlingene som Bråting og Kilhamn (2022) utviklet i sitt rammeverk (2.9.3). Siden kategoriene var forhåndsbestemt, ble delanalyse 1 dermed en deduktiv analyse. Handlingene er som følger;

a) Følge en regel: Å følge steg-for-steg instruksjoner, repetere eller fortsette på et mønster.
b) Finne regel: Finne ut prosedyren, regelen eller mønsteret som genererer et utfall, for eksempel et tallmønster.
c) Feilsøke: Å feilsøke en kode.
d) Forme og skape: Gi instruksjoner, lage et mønster, skrive kode og å representere med symboler.
e) Forklare: Forklare med naturlig språk. Bruke ord for å beskrive en prosedyre, regel, et mønster eller et begrep.
f) Forestille seg: Forutse hva som vil skje. Reflektere over mulige utfall når verdier eller betingelser er endret.

Gjennom analyseprosessen kom jeg over to typer oppgaver der jeg var usikker på hvilken handling oppgavene ba om. Den første typen oppgaver var oppgaver der elevene ble bedt om å utvide et program som var gitt. Man kan argumentere for at denne typen oppgaver kunne blitt kategorisert som både «følge en regel» og «forme og skape» siden den vil kunne innebære å fortsette et mønster samtidig som elevene må skrive ny kode. Jeg har valgt å kategorisere alle oppgaver der elevene må utvide et program som «forme og skape» ettersom det vil kreve at elevene må produsere ny kode.

Den andre typen oppgaver var nesten helt like oppgaver gitt i eksempler. Når en løsning da ble gitt i eksempelet, ville oppgaven kunne bli løst på nesten samme måte som

eksempelet. Som oftest var det kun en kodelinje som måtte endres eller at man måtte endre input til programmet. Slike oppgaver er blitt kategorisert som en «følge en regel»-oppgave ettersom elevene i praksis kun må følge eksempelet «steg-for-steg».

### 4.1.2 Delanalyse 2 - Begreper

For å finne ut mer om hva som karakteriserer programmeringsoppgaver og for å finne ut hvilke temaer programmering knyttes til så jeg på hvilke programmeringsbegreper og matematiske begreper som var knyttet til oppgavene.

**Programmeringsbegreper**
For å analysere programmeringsbegreper tok jeg utgangspunkt i programmeringsbegrepene fra Bråting og Kilhamn (2022) sitt analyseverktøy. De ble dermed:

- stegvise instruksjoner
- algoritmer
- kode
- regel
- løkker
- vilkår
- feilsøking

I Bråting og Kilhamn (2022) sin undersøkelse brukte de en deskriptiv analyse der de så etter hvilke programmeringsbegrep som ble nevnt i oppgavene. I første gjennomgang av delanalyse 2 kategoriserte jeg hvilke programmeringsbegreper som står spesifikt i oppgavene. Da kom jeg frem til at mange av oppgavene innhold ordene «kode» og «program», men lite av de andre programmeringsbegrepene. En oversikt over antall programmeringsbegreper nevnt spesifikt i oppgavene finnes i figur 4.


*Figur 4 Programmeringsbegreper nevnt spesifikt i oppgaveteksten i datasettet*

I datasettet finnes mange oppgaver av typen «Lag et program […]» og som dermed ble kategorisert som «kode» og ingen andre programmeringsbegreper. Derfor var det ikke så interessant for å svare på forskningsspørsmålet «Hva karakteriserer programmeringsoppgavene i matematikk 1T […]», og se på resultatene fra den

deskriptive analysen. Jeg kom frem til at det ville være mer informativt å se etter hvilke programmeringsbegreper elevene må bruke eller forstå for å løse oppgavene. Det vil si hvilke programmeringskunnskaper oppgavene faktisk etterspør. Jeg så på løsningsforslagene til lærebøkene for å se hvordan forfatterne tenkte at oppgaven burde løses. I de tilfellene det ikke var et løsningsforslag tilgjengelig, lagde jeg løsningsforslag selv. «Feilsøking» er en kategori vi også finner som en av handlingene, men ettersom feilsøking både er en arbeidsmetode og et programmeringsbegrep, har jeg valgt å inkludere «feilsøking» til både delanalyse 1 og 2.

**Matematiske begreper**
Når jeg kategoriserte matematiske begreper, tok jeg først utgangspunkt i de matematiske begrepene fra Bråting og Kilhamn (2022) sitt rammeverk. Altså:

- mønster
- geometri
- aritmetikk
- rotasjon
- koordinatsystem

Før jeg begynte å analysere de matematiske begrepene skjønte jeg at disse begrepene ikke kom til å være dekkende for oppgaver på videregående nivå. I tillegg er programmering i matematikk i svenske læreplaner knyttet til algebragrenen, mens kompetansemålet om programmering i matematikk 1T ikke er knyttet til en spesifikk matematisk gren. Dermed valgte jeg å ta utgangspunkt i begrepene fra Bråting og Kilhamn (2022), men legge til nye kategorier for matematiske begreper etter hvert som jeg kom over oppgaver som ikke passet under de eksisterende kategoriene. Jeg gjennomførte dermed en blanding mellom deduktiv og induktiv analyse i denne delanalysen.

En oppgave ble kategorisert som et matematisk begrep dersom oppgaven, eller tilhørende eksempler og forklaringer, var knyttet til tradisjonell skolematematikk og hvis innholdet formidlet en viktig matematisk ide. Etter første gjennomgang av oppgavene hadde jeg lagt til «likning», «tilnærming», «funksjon», «vekstrate» og «tallteori» som nye kategorier. Det var også noen oppgaver uten matematisk innhold, dermed ble «uten matematisk innhold» også lagt til som en kategori. I tillegg fjernet jeg rotasjon og koordinatsystem, ettersom ingen oppgaver ble kategorisert under dem. Deretter tok jeg en ny gjennomgang av oppgavene med de nye kategoriene for å sikre at de ble plassert i riktig kategori. Etter andre gjennomgang så jeg at «aritmetikk»-kategorien og «tallteori»-kategorien delte mange oppgaver samtidig som at oppgavene opplevdes like. Ettersom mange ikke setter noe klart skille mellom aritmetikk og tallteori (Hofmann, 2007) valgte jeg å sette oppgavene fra «tallteori» inn i kategorien «aritmetikk», og fjerne «tallteori». Oppgavene i «aritmetikk»-kategorien ble dermed oppgaver som handlet om egenskaper til tall, der typiske oppgaver handlet om partall, oddetall, kvadratrot eller modulo. Noen kan nok argumentere for at «likning» og «mønster» burde samles i en kategori kalt «algebra». Ettersom det fantes oppgaver rettet spesifikt mot «likning» og «mønster», og jeg er ute etter å vise hvilke temaer programmering knyttes til, lot jeg de få sin egen kategori. Etter siste gjennomgang av oppgavene ble kategoriene for matematiske begreper:

- mønster
- geometri
- likning
- tilnærming
- funksjon
- vekstrate
- aritmetikk
- uten matematisk innhold

I tillegg til å analysere hvilke matematiske begreper oppgavene inneholder har jeg gjennomført en helhetlig analyse av oppgavene med matematisk innhold der jeg har sett om noen av oppgavene kan knyttes til sentrale ideer. Der tok jeg utgangspunkt i to oppgaver innenfor de matematiske begrepene «tilnærming» og «mønster». Jeg har også sett hvilke kapitler de ulike oppgavene hører til, og hvor mange programmeringsoppgaver som finnes i de ulike kapitlene. Det er for å kunne svare mer utdypende om hvilke temaer programmering knyttes til.

### 4.1.3 Delanalyse 3 - Kobling mellom programmering og matematikk

For å kunne svare på hvordan programmering og matematikk er knyttet sammen undersøkte Bråting og Kilhamn (2022, s. 600) kombinasjonen mellom handlinger og begreper. Spesielt så de etter om elevene ble bedt om å omformulere programmeringsideer ved å bruke matematisk notasjon eller om oppgavene gjorde at de kunne utforske ukjente matematiske ideer. I min analyse av kobling mellom programmering og matematikk har jeg også undersøkt om oppgavene krever at elevene omformulerer programmeringsideer ved å bruke matematisk notasjon, eller om oppgavene kan virke utforskende for elevene. Jeg tok også inspirasjon fra en norsk masteroppgave som fant at matematikk som skal være kjent for elevene ofte blir brukt for å konstruere en kontekst for programmering (Engen, 2022, s. 36). Dermed la jeg til «matematikk som kontekst» som en kategori. I tillegg fant jeg under delanalyse 2 at noen oppgaver er uten matematisk innhold. Derfor la jeg også det til som en kategori. Kategoriene ble laget før jeg gjennomførte delanalyse 3, så delanalysen ble deduktiv. Grunnen til at jeg ville ha noen spesifikke kategorier var for å lettere kunne sortere oppgavene. I diskusjonen av forskningsspørsmål 2 vil jeg ta for meg oppgavene innenfor de ulike kategoriene og presentere hvordan de knyttes til noen spesifikke handlinger og begreper. For å oppsummere er kategoriene i delanalyse 3:

- oppgaver uten matematisk innhold
- oppgaver med matematikk som kontekst for oppgaven
- oppgaver der elever må omformulere programmeringsideer til matematisk notasjon
- oppgaver der elevene kan utforske matematiske ideer som er nye for dem

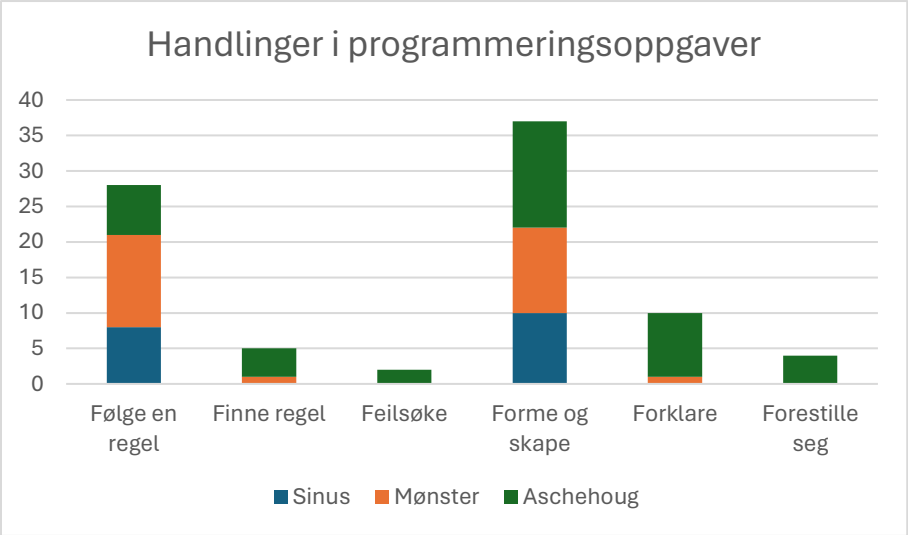
## 4.2 Resultater

Som nevnt i analyseprosessen (4.1) har analysen bestått av tre delanalyser, på totalt 63

oppgaver. Den første delanalysen omhandler hvilke handlinger som finnes i oppgavene. Den andre delanalysen omhandler hvilke programmeringsbegreper og matematiske begreper som finnes i oppgavene, og den siste delanalysen omhandler hvordan programmeringsoppgavene knyttes til matematikk. Jeg kommer til å presentere funnene og gi eksempler på oppgaver som hører til de ulike kategoriene i de neste delkapitlene.

### 4.2.1 Delanalyse 1 - Handlinger

I den første delanalysen har jeg undersøkt hvilke handlinger oppgavene ber elevene om å gjøre. Som nevnt i analyseprosessen (4.1.1) har jeg kategorisert oppgavene innenfor de seks handlingene (a) følge en regel, (b) finne regel, (c) feilsøke, (d) forme og skape, (e) forklare og (f) forestille seg. I figur 5 ser vi fordelingen av de seks handlingene i oppgavene. Det er verdt å merke seg at 46 av de 63 oppgavene har kun én handling knyttet til oppgaven. 17 oppgaver har to eller flere handlinger og dermed blir totalsummen av handlinger mer enn totalt antall oppgaver.




*Figur 5 Fordelingen av handlingene i programmeringsoppgaver. Noen oppgaver opptrer i flere kategorier.*

Fra figur 5 ser vi at den vanligste handlingen er «forme og skape» og den karakteriserer 37 av de 63 oppgavene. En oppgave er karakterisert som «forme og skape» hvis eleven blir bedt om å lage et program, endre et program eller utvide et program. Et eksempel hvor eleven blir bedt om å lage et program er gitt i figur 6.

**1.47**

Skriv et program som trekker et tilfeldig heltall mellom 1 og 100 og lar brukeren gjette hvilket tall maskinen har trukket ut. Så lenge brukeren gjetter feil, skal programmet oppgi om brukeren har gjettet for høyt eller for lavt.

Tips: Kommandoen `randint(0, 10)` fra biblioteket random trekker ut et tilfeldig tall mellom 0 og 10.

*Figur 6 Eksempel på «Forme og skape» i en oppgave fra Mønster (Kalvø et al., 2020, s. 44).*

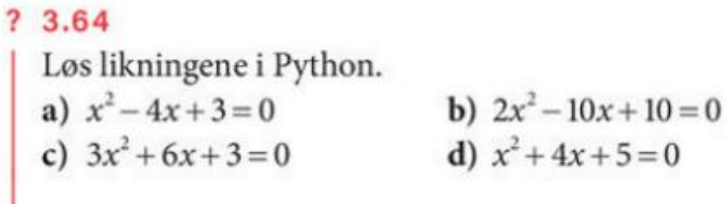
Hvis en «Lag et program […]» oppgave er gitt rett etter at boka forklarer hvordan det kan gjøres i form av et eksempel, så kategoriseres oppgaven som «følge en regel». Det vil være oppgaver som krever lite av elevene fordi elevene ofte bare må bytte ut en kodelinje eller gi en spesifikk input til programmet. I figur 7 ser vi et eksempel på hvordan man kan lage et program i Python som løser andregradslikninger ved hjelp av andregradsformelen. På samme side kommer en oppgave der elevene blir bedt om å løse ulike likninger i Python. Figur 8 viser denne oppgaven. Det som kreves av elevene for å løse oppgaven er å skrive av programmet i eksempelet, kjøre programmet og gi programmet riktig input for a, b, og c. Dermed er oppgaven i figur 8 karakterisert som «følge en regel». «Følge en regel» er den nest mest vanlige handlingen sammenlagt og karakteriserer 28 av de 63 oppgavene.

Ved hjelp av andregradsformelen kan vi lage et program i Python som løser alle andregradslikninger. Her er et slikt program:

<> Kodeeditor

```
from math import sqrt
print("Vi skal løse andregradslikningen ax^2+bx+c=0")
svar = input ("a =")
a = float(svar)
if a == 0:
  print("Med a = 0 blir det ikke en andregradslikning.")
else:
  svar = input("b =")
  b = float(svar)
  svar = input("c =")
  c = float(svar)
  d = b**2-4*a*c
  if d < 0:
    print("Ingen løsning")
  elif d == 0:
    x1 = -b/(2*a)
    print("En løsning x =", round(x1,2))
  else:
    x1 = (-b+sqrt(d))/(2*a)
    x2 = (-b-sqrt(d))/(2*a)
    print("To løsninger x =", round(x1,2), "og x =", round(x2,2))
```

*Figur 7 Et program for å løse andregradslikninger. Fra Sinus (Oldervoll et al., 2020, s. 115).*



**? 3.64**

Løs likningene i Python.

**a)** $x^2 - 4x + 3 = 0$ **b)** $2x^2 - 10x + 10 = 0$

**c)** $3x^2 + 6x + 3 = 0$ **d)** $x^2 + 4x + 5 = 0$

*Figur 8 Oppgave sterkt knyttet til et eksempel og dermed en «følge en regel»-oppgave. Fra Sinus (Oldervoll et al., 2020, s. 115).*

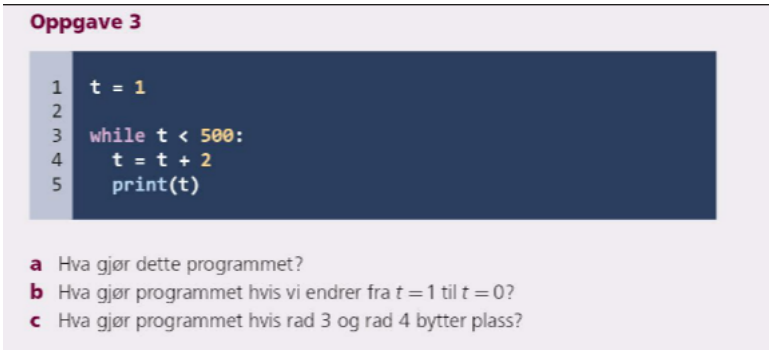
Handlingen «forklare» finnes i 10 oppgaver. Typisk kommer handlingen som en deloppgave der elevene skal forklare hvordan et program, eller deler av programmet, fungerer. De tre siste handlingene; «finne regel», «forestille seg» og «feilsøke» er lite representert i lærebøkene. Disse handlingene er til stede i henholdsvis 5, 4 og 2 av oppgavene med programmeringsinnhold.

Oppgaver som er karakterisert som «finne regel» er oppgaver der elevene blir bedt om å

finne ut regelen eller mønsteret som genererer et utfall. I de fleste oppgavene under denne kategorien blir elevene bedt om å finne ut av regelen før de blir bedt om å lage et program eller forklare et program som bruker regelen. Tre av de fem oppgavene som har handlingen «finne regel» er definert som utforsk-oppgaver, og finnes i Aschehoug. I figur 9 finner vi et eksempel på en oppgave som er karakterisert som «finne regel». I oppgave a) må eleven finne ut mønsteret i programmet for å finne ut hvilken output programmet har. I dette tilfellet at programmet printer ut alle oddetall fra og med 3 til og med 501.

Oppgaven i figur 9 er også et eksempel på en oppgave som er karakterisert som «forestille seg». I oppgave b og c må eleven se for seg hva som skjer når visse betingelser endrer seg.

Oppgaver er karakterisert som «feilsøke» dersom det er presentert en kode som inneholder feil som elevene må finne. Oppgavene er slik at elevene blir bedt om å skrive inn en input som gjør at programmet ikke kjører. Deretter må elevene finne ut av hva som er feil. Et eksempel ser vi i oppgave b) i figur 10. Oppgaven ber elevene om å skrive inn et likningssystem som fører til at programmet gir en feilmelding. Ved å lese feilmeldingene kan eleven forstå at betingelsen som ikke er oppfylt er at nevneren er null. Det betyr at likningssystemet enten ikke har en løsning eller at det har uendelig mange løsninger.

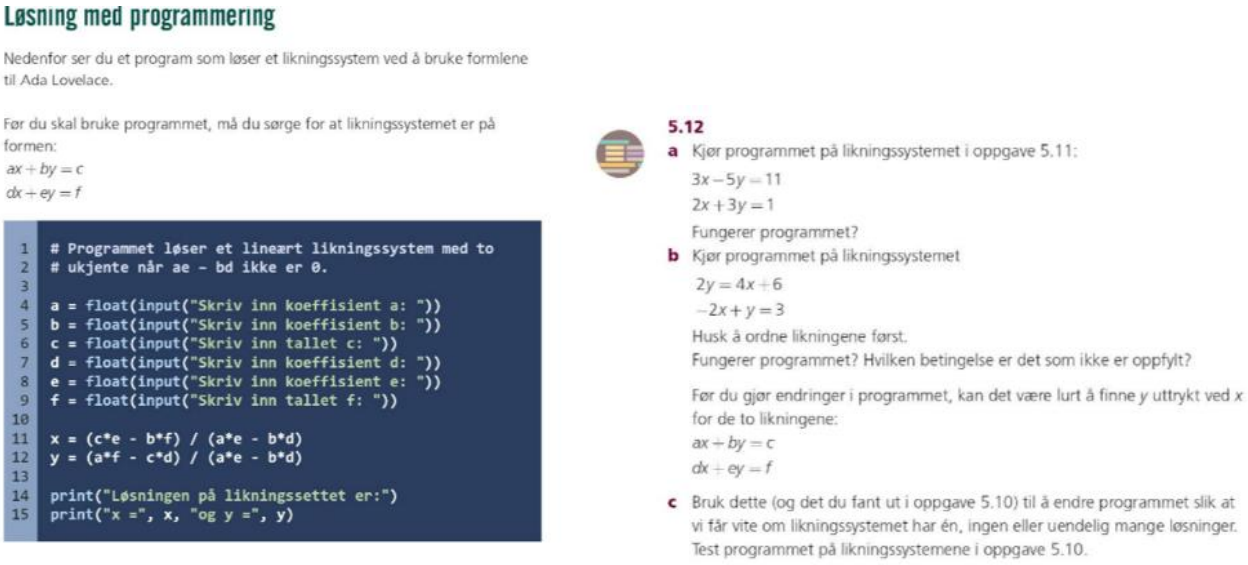


**Oppgave 3**

```
1  t = 1
2
3  while t < 500:
4    t = t + 2
5    print(t)
```

**a** Hva gjør dette programmet?
**b** Hva gjør programmet hvis vi endrer fra $t = 1$ til $t = 0$?
**c** Hva gjør programmet hvis rad 3 og rad 4 bytter plass?

*Figur 9 Eksempel på en oppgave karakterisert som «finne regel» og «forestille seg». Fra Aschehoug (Borge et al., 2020, s. 51).*



**Løsning med programmering**

Nedenfor ser du et program som løser et likningssystem ved å bruke formlene til Ada Lovelace.

Før du skal bruke programmet, må du sørge for at likningssystemet er på formen:
$ax + by = c$
$dx + ey = f$

```
# Programmet løser et lineært likningssystem med to
# ukjente når ae - bd ikke er 0.

a = float(input("Skriv inn koeffisient a: "))
b = float(input("Skriv inn koeffisient b: "))
c = float(input("Skriv inn tallet c: "))
d = float(input("Skriv inn koeffisient d: "))
e = float(input("Skriv inn koeffisient e: "))
f = float(input("Skriv inn tallet f: "))

x = (c*e - b*f) / (a*e - b*d)
y = (a*f - c*d) / (a*e - b*d)

print("Løsningen på likningssettet er:")
print("x =", x, "og y =", y)
```

**5.12**
**a** Kjør programmet på likningssystemet i oppgave 5.11:
$3x - 5y = 11$
$2x + 3y = 1$
Fungerer programmet?
**b** Kjør programmet på likningssystemet
$2y = 4x + 6$
$-2x + y = 3$
Husk å ordne likningene først.
Fungerer programmet? Hvilken betingelse er det som ikke er oppfylt?

Før du gjør endringer i programmet, kan det være lurt å finne $y$ uttrykt ved $x$ for de to likningene:
$ax + by = c$
$dx + ey = f$

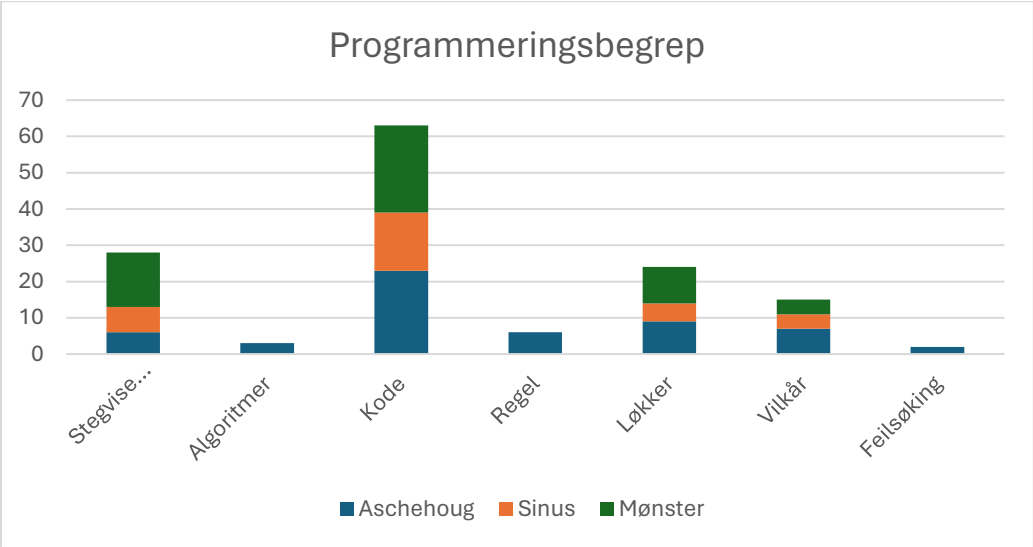
**c** Bruk dette (og det du fant ut i oppgave 5.10) til å endre programmet slik at vi får vite om likningssystemet har én, ingen eller uendelig mange løsninger. Test programmet på likningssystemene i oppgave 5.10.

*Figur 10 Eksempel på en oppgave karakterisert som «feilsøke». Fra Aschehoug (Borge et al., 2020, s. 238).*

### 4.2.2 Delanalyse 2 - Begreper

I delanalyse 2 har jeg undersøkt hvilke programmeringsbegreper elevene må kunne for å løse oppgavene og hvilke matematiske begreper oppgavene kan knyttes til. Jeg vil først presentere programmeringsbegrepene, før jeg presenterer de matematiske begrepene. Presentasjonen av de matematiske begrepene består av tre deler. Først en analyse av hvilke matematiske begreper som finnes i oppgavene. Til slutt vil jeg presentere hvordan to matematiske begreper kan knyttes til sentrale ideer og gi eksempler på to oppgaver som er knyttet til sentrale ideer. Deretter vil jeg presentere en oversikt over hvilke kapitler programmeringsoppgavene er knyttet til (og ikke knyttet til) i de tre lærebøkene.

**Programmeringsbegreper**




*Figur 11 Antall oppgaver der ulike programmeringsbegreper er identifisert.*

I figur 11 ser vi at alle de 63 programmeringsoppgavene inneholdt programmeringsbegrepet «kode». Dette innebærer oppgaver der eleven enten må skrive kode eller forstå kode. At alle oppgavene inneholder «kode» kan ha noe med hvilke oppgaver jeg har valgt å analysere. Utvalget av oppgaver består av de oppgavene som inneholder ordet «programmering», er tydelig knyttet til programmering eller har et programmeringssymbol ved siden av oppgaven. Når jeg gikk igjennom oppgavene fant jeg ingen oppgaver som var tydelig knyttet til programmering, men som ikke inneholdt ordet «programmering». Altså var det ingen oppgaver med analog programmering eller oppgaver der elevene kunne reflektere over hvilke digitale hjelpemidler de kunne ha brukt for å løse oppgaven. At alle oppgavene inneholdt kode, kan ha noe med at dette er matematikk på et videregående nivå. Programmeringsbegreper skal hovedsakelig være innlært på grunnskolen, som det går frem av kompetansemålene for matematikk 1.-10. trinn (Kunnskapsdepartementet, 2019). I tillegg blir løkker og vilkår godt forklart sammen med eksempler, sånn at elevene ved hjelp av forklaringen skal kunne klare å bruke begrepene riktig i oppgaver.

«Stegvise instruksjoner» er det nest vanligste programmeringsbegrepet, og finnes i 28 oppgaver. En oppgave er kategorisert som «stegvise instruksjoner» dersom oppgaven innebærer at eleven bare trenger å følge en «rett frem» forklaring eller bruke tidligere eksempler som mal for å løse oppgaven. En oppgave med «stegvise instruksjoner»

krever dermed lite av elevene. De oppgavene som har blitt plassert under kategorien «stegvise instruksjoner» er de oppgavene der jeg oppfatter at elevene ikke trenger å forstå programmeringsbegreper for å løse oppgaven, men bare forstå sammenhengen mellom et eksempel og oppgaven. Denne kategoriseringen medfører at «stegvise instruksjoner» blir mye av det samme som handlingen «følge en regel». I noen av oppgavene med «stegvise instruksjoner» krever løsningen også bruk av andre programmeringsbegreper, for eksempel «vilkår». Ettersom elevene bare trenger å skrive av eksempelet og endre noen tall, trenger de egentlig ikke å forstå «vilkår». I disse tilfellene har ikke oppgaven blitt kategorisert som «vilkår», bare «stegvise instruksjoner». Et eksempel på dette ser vi i figur 7 og 8 der oppgavene fra figur 8 kan løses ved å skrive av eksempelet i figur 7 og skrive inn riktig tall i «input». Noen oppgaver er definert som «stegvise instruksjoner» og andre programmeringsbegrep. Det er fordi noen oppgaver har en deloppgave som er å følge eksempelet og en annen deloppgave som for eksempel inkluderer å utvide programmet.

Programmeringsbegrepet «løkker» finnes i 24 oppgaver. Oppgavene er kategorisert her dersom elevene er nødt til å forstå løkker, enten ved å forklare et program med løkker eller skrive et program der løkker burde brukes for å løse oppgaven. På samme måte som beskrevet ovenfor vil ikke oppgaver bli kategorisert som «løkker» dersom det er «stegvise instruksjoner».

«Vilkår» finnes i 15 oppgaver. Oppgaver er kategorisert her dersom oppgaven krever at elevene må forklare vilkår, forstå vilkår for å kunne forstå et program eller bruke vilkår for å skrive et program.

Alle begrepene hittil («kode», «stegvise instruksjoner», «løkker» og «vilkår») har blitt funnet i alle lærebøkene. De tre siste begrepene, «regel», «algoritmer» og «feilsøking» finnes kun i læreboken fra Aschehoug. 6 oppgaver har blitt kategorisert som å inneholde «regel». Det er oppgaver der elevene er nødt til å finne ut hvorfor et program fungerer, eller hva programmet kommer til å gjøre. I tillegg er oppgaver der elevene må undersøke et mønster for å så skulle lage et program som fortsetter med mønsteret, kategorisert som «regel».

3 oppgaver er karakterisert som «algoritmer». En oppgave er kategorisert under kategorien «algoritmer» dersom oppgaven spesifikt ber elevene om å lage en algoritme eller skrive et program som løser en gitt algoritme. Statped (2021) definerer en algoritme som følgende:

> En algoritme er helt enkelt forklart, en oppskrift. Det er en steg-for-steg-plan for å få noe gjort. Hvis algoritmen skal utføres av en datamaskin vil den at du skal skrive stegene med kommandoer som datamaskinen forstår. Dette kaller vi et programmeringsspråk. Når du er i utlandet og snakker engelsk kan det hende at du ikke er helt nøyaktig med grammatikk eller ordvalg, men likevel forstår andre hva du mener. Slik er det ikke for en datamaskin. Den må få beskrevet alle stegene i riktig rekkefølge. Oppskriften, eller algoritmen, må være helt nøyaktig.

Dersom man skulle fulgt denne definisjonen ville man kunne argumentert for at all kode er en algoritme. Når man skriver programkode, må man skrive algoritmer som datamaskinen forstår. For at denne kategorien skal kunne bidra til å svare på det første

forskningsspørsmålet, satt jeg mer spesifikke krav for å kunne kategorisere en oppgave under begrepet «algoritme». Kravet for å kategorisere en oppgave under «algoritme» ble dermed at «algoritme» spesifikt måtte bli nevnt i oppgaven, eller at det var en helt spesifikk oppskrift elevene skulle følge eller en spesifikk oppskrift de skulle skrive ned. Det måtte altså være en detaljert oppskrift der rekkefølgen sto i fokus. Et eksempel på en oppgave kategorisert under programmeringsbegrepet «algoritme» finner vi i figur 12.

**1.42**

Ta for deg denne algoritmen:

*Tenk på et tall. Trekk så fra tre multiplisert med tallet som er én mindre enn tallet du tenkte på. Legg til det dobbelte av tallet du tenkte på. Trekk til slutt fra kvadratroten av ni.*

**a** Utfør algoritmen for noen tilfeldige tall. Hva ender du opp med?

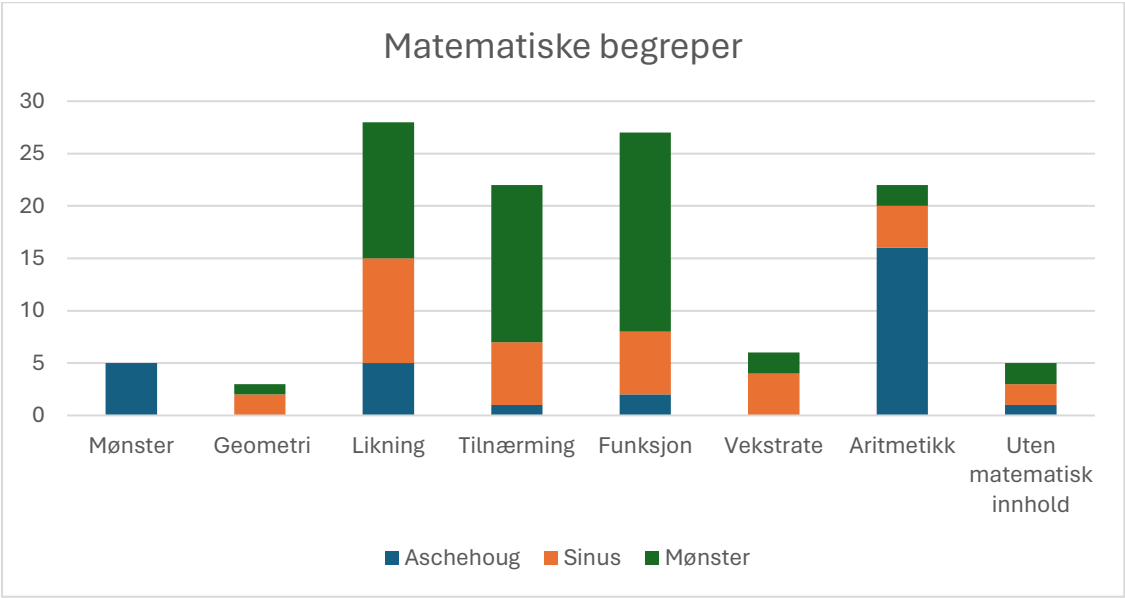
**b** Frank tenker på tallet −5.
Sett opp algoritmen for Franks tall i ett regnestykke, som du så regner ut.

**c** Lag et program som utfører algoritmen.

*Figur 12 Oppgave med programmeringsbegrepet «algoritmer». Fra Aschehoug (Borge et al., 2020, s. 27).*

Programmeringsbegrepet «feilsøking» er begrepet som forekommer minst i oppgavene. Oppgaver ble kategorisert her dersom oppgaven ba elevene spesifikt om å se etter feil eller dersom oppgaven er lagt opp slik at elevene skal få opp en feilmelding. Man kan argumentere for at feilsøking alltid kan skje når man skriver kode, men det vil ikke være hensiktsmessig å kategorisere alle oppgaver der elevene må skrive kode under denne kategorien. Det som gir informasjon med tanke på forskningsspørsmålene, er om oppgavene legger opp til at elevene skal bruke feilsøking som et verktøy. Med denne karakteriseringen fant jeg 2 oppgaver som krever «feilsøking».

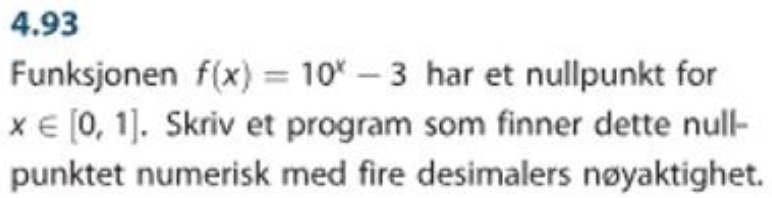
**Matematiske begreper**




*Figur 13 Antall oppgaver der ulike matematiske begreper er identifisert. Totalt 63 oppgaver, der noen oppgaver hører til flere kategorier.*

Figur 13 viser hvor mange oppgaver som omhandler ulike matematiske begreper. Fra figuren ser vi at de dominerende begrepene er «likning», «funksjon», «aritmetikk» og

«tilnærming» med henholdsvis 28, 27, 22 og 22 oppgaver. Fra analysen er det nyttig å merke seg at oppgaver kategorisert som «tilnærming» er i alle tilfeller, unntatt ett, også kategorisert som enten «likning» eller «funksjon». I tillegg er det mange oppgaver der alle de tre begrepene; «funksjon», «likning» og «tilnærming» finner sted. Oppgaver med funksjoner handler typisk om å finne nullpunkter eller den deriverte til en funksjon, ved hjelp av en numerisk metode. Et eksempel på en oppgave karakterisert som både «funksjon», «likning» og «tilnærming» er gitt i figur 14.

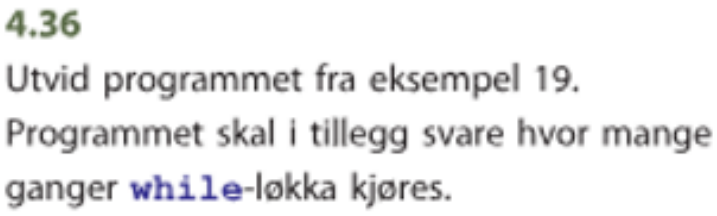

**4.93**

Funksjonen $f(x) = 10^x - 3$ har et nullpunkt for $x \in [0, 1]$. Skriv et program som finner dette nullpunktet numerisk med fire desimalers nøyaktighet.

*Figur 14 Oppgave som inneholder de matematiske begrepene «funksjon», «likning» og «tilnærming». Fra Mønster (Kalvø et al., 2020, s. 273).*

«Aritmetikk»-kategorien inneholder som nevnt (i analyseprosessen, 4.1.2) oppgaver knyttet til aritmetikk og tallteori. Oppgavene i denne kategorien kommer fra kapitler kalt «algebra», «tallforståelse» og «tall og regning». En typisk oppgave kategorisert som «aritmetikk» er: «Skriv et program som skriver ut alle oddetall mellom 1 og 100». Et annet eksempel kan ses i figur 9 (kapittel 4.2.1).

De fire siste begrepene er «mønster», «vekstrate», «geometri» og «uten matematisk innhold». Disse begrepene ser vi lite av i oppgavene. Her skiller oppgavene fra Aschehoug seg noe ut. Aschehoug er den eneste boka som har oppgaver knyttet til «mønster», og har ingen oppgaver knyttet til «vekstrate» eller «geometri». Matematikk 1T har ingen kompetansemål knyttet direkte til geometri, men har flere kompetansemål knyttet til trigonometri, som er en undergren av geometri (Kunnskapsdepartementet, 2020). Dermed er oppgaver som kan knyttes til trigonometri plassert under kategorien «geometri». Oppgavene kategorisert som «vekstrate» er typisk knyttet til personlig økonomi i form av sparing og lån. I tillegg knyttes oppgavene ofte til den deriverte ved at elevene for eksempel skal undersøke årlig avkastning på en sparekonto. Det fantes kun 5 oppgaver uten matematisk innhold og figur 15 viser et eksempel på en slik oppgave.


**4.36**

Utvid programmet fra eksempel 19. Programmet skal i tillegg svare hvor mange ganger `while`-løkka kjøres.

*Figur 15 Oppgave uten matematisk innhold. Fra Mønster (Kalvø et al., 2020, s. 264).*

**Matematiske begreper som kan knyttes til sentrale ideer - to eksempler**

Fra figur 13 ser vi at mange oppgaver kan knyttes til det matematiske begrepet «tilnærming». Som nevnt i teorien (kapittel 2.8) kan oppgaver om tilnærming knyttes til en sentral ide om numerisk tilnærming:

> «Numeriske beregninger kan tilnærmes ved å erstatte tall med andre tall som er nærme i verdi og enkle å regne med mentalt. Målinger kan tilnærmes ved å bruke kjente referenter som enheten i måleprosessen» (Charles & Carmel, 2005, s. 17).

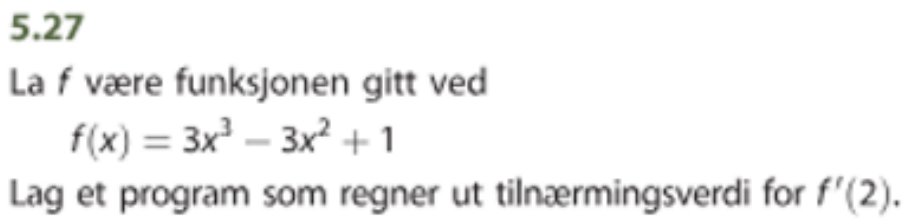
De sentrale ideene fra Charles og Carmel (2005) forstås i kontekst av matematikk på bare- og ungdomsskolen og før programmering ble integrert i matematikkfaget. Det medfører at man må gjøre en liten endring i de sentrale ideene for at de skal være passende for et videregående nivå. I lys av kompetansemål på videregående nivå ser jeg på numerisk tilnærming som et mer avansert konsept enn beskrevet i den sentrale ideen til Charles og Carmel (2005, s. 17). Det gjelder spesielt at jeg ønsker at elevene skal kunne analysere hvordan man lager gode tilnærminger og finne ut hva som kan være en tilstrekkelig tilnærming til den korrekte løsningen.

Hvis elevene har forstått den sentrale ideen om numerisk tilnærming, vil de kunne se på nyttigheten av programmering som et verktøy for å tilnærme ulike beregninger. Man trenger ikke lenger å velge tall som er enkle å regne med mentalt ettersom datamaskinen kan regne vanskelige uttrykk for oss. Allikevel må man fortsatt velge verdier som er nærme i verdi og som er løsbare når man gjør en tilnærming.

Et eksempel på en oppgave knyttet til den sentrale ideen er gitt i figur 16. Her må elevene lage et program der de må bruke det de kan om tilnærming av den deriverte i et punkt. Nettopp at:

$$f'(a) \approx \frac{f(a+\Delta x)-f(a)}{\Delta x}$$

I formelen brukes gjennomsnittlig vekstfart i et lite område fra $a$ og dermed er $\Delta x$ er et lite tall, for eksempel 0,01 (Kalvø et al., 2020). Formelen kan brukes for å finne en tilnærmet verdi for den deriverte i punktet $a$. Denne formelen kan være nyttig ettersom det finnes uttrykk som er vanskelige å løse analytisk. Akkurat oppgaven i figur 16 vil være enklere å løse analytisk enn ved en tilnærming. Den deriverte i punktet $x = 2$ kan finnes ved å bruke potensregelen og deretter bytte ut $x$ med verdien 2. Allikevel kommer oppgaven før elevene får presentert regneregler for den deriverte og oppgaven kan dermed knyttes til introduksjonen av derivasjon. Det som kan være positivt med å bruke programmering i dette tilfellet er at elevene kan se hva som skjer med den deriverte når $\Delta x$ endrer verdi.



**5.27**

La $f$ være funksjonen gitt ved

$$f(x) = 3x^3 - 3x^2 + 1$$

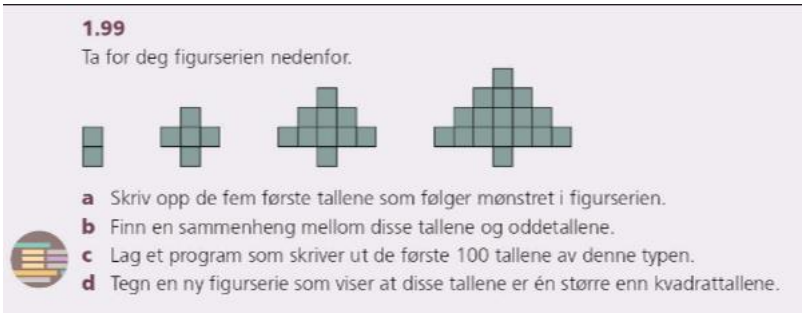
Lag et program som regner ut tilnærmingsverdi for $f'(2)$.

*Figur 16 Oppgave knyttet til den sentrale ideen om numerisk tilnærming. Fra Mønster (Kalvø et al., 2020, s. 305).*

Et annet matematisk begrep som kan knyttes til en sentral ide er «mønster». I følge Charles og Carmel (2005) er en sentral ide:

> «Sammenhenger kan beskrives og generaliseringer kan gjøres for matematiske situasjoner som inneholder tall eller objekter som gjentar seg på forutsigbare måter» (Charles & Carmel, 2005, s. 17).

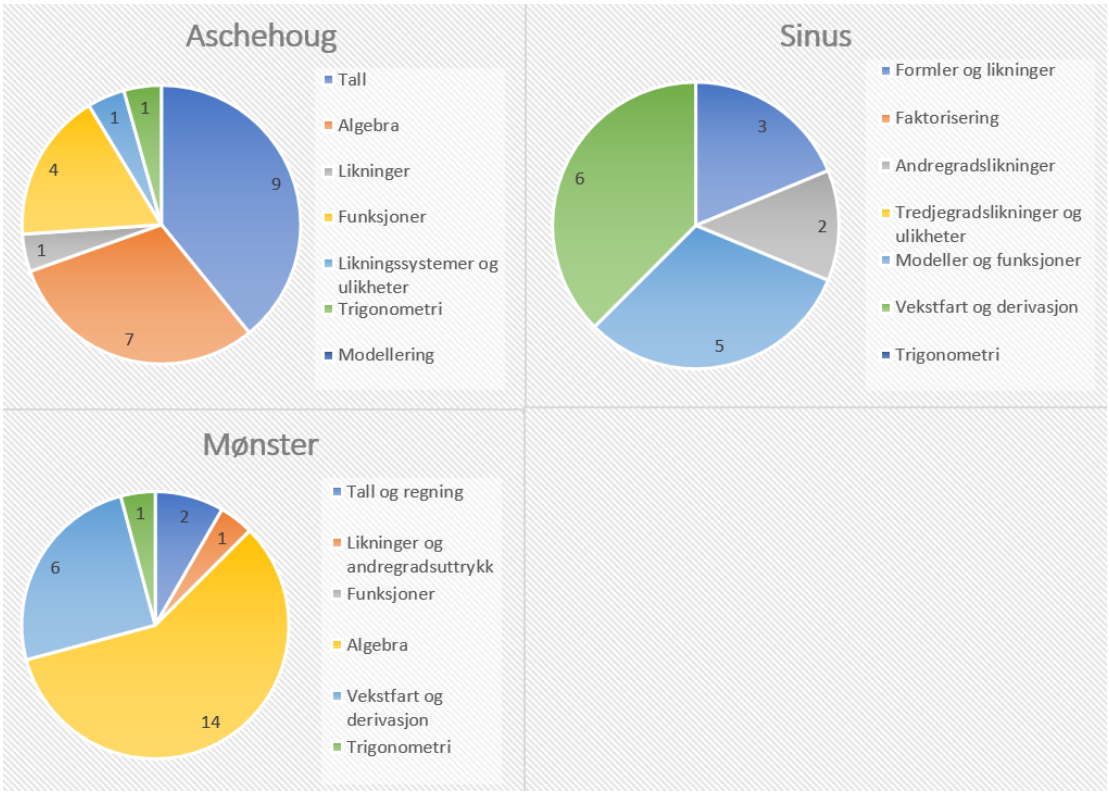
I Aschehoug finnes flere oppgaver som handler om figurtall. Disse oppgavene hører til det matematiske begrepet «mønster». Et eksempel på en av oppgavene knyttet til den sentrale ideen om mønster kan observeres i eksempel 17.




*Figur 17 Oppgave knyttet til den sentrale ideen om mønster. Fra Aschehoug (Borge et al., 2020, s. 48).*

I oppgave c i figur 17 bes elevene om å lage et program som skriver ut de 100 første tallene i figurserien. Sammenhengen mellom de 100 tallene kan generaliseres på en rekursiv måte. Det vil si at verdien av den avhengige variabelen uttrykkes basert på at man kjenner til verdiene før (Solem et al., 2017, s. 351). Videre hevder Solem et al. (2017, s. 351) at å kunne generalisere rekursivt har stor verdi for elevenes algebraiske tenking og er selve grunnlaget for å resonnere induktivt i matematikken. En annen måte å generalisere på er ved å bruke en eksplisitt generalisering. Da finner man verdien av den avhengige variabelen direkte ut fra den uavhengige variabelen. Altså at man kan lage en funksjon som sier hvor mange kvadrater det er i figur nummer n. Det vil være vanskelig å finne antall kvadrater i en figur med et høyt figurnummer ved en rekursiv generalisering dersom man ikke har hjelpemidler. Hvis man kan bruke programmering som et hjelpemiddel, kan man derimot finne antall kvadrater i et høyt figurnummer ved hjelp av en rekursiv metode. Dette gjøres ved å utnytte programmeringsbegrepet løkker. Oppgave c i figur 17 kan dermed løses ved at elevene klarer å generalisere sammenhengen mellom figurtallene. Det som kan være negativt ved å bruke en rekursiv metode er at man mister sammenhengen mellom mønster og funksjonssammenhenger. Det er nettopp korrespondansen mellom den uavhengige og avhengige variabelen som er funksjonstenking og den eksplisitte formelen som gir selve funksjonsuttrykket (Solem et al., 2017, s. 351). Funksjonsuttrykket for figurserien i figur 17 er: $f(n) = n^2 + 1$, gitt at den første figuren i oppgaven er figur nummer 1. Selv om både rekursive og eksplisitte generaliseringer har sine fordeler, vil programmering være nyttig i de tilfellene et mønster er vanskelig å finne eksplisitt. Et eksempel på et mønster som er vanskelig å løse eksplisitt (og som faktisk er uløst) er Collatz sekvens (Garner, 1981). Her vil programmering fungere som en nyttig metode for å finne sekvensen.

**Fordeling av programmeringsoppgaver i kapitlene til lærebøkene**

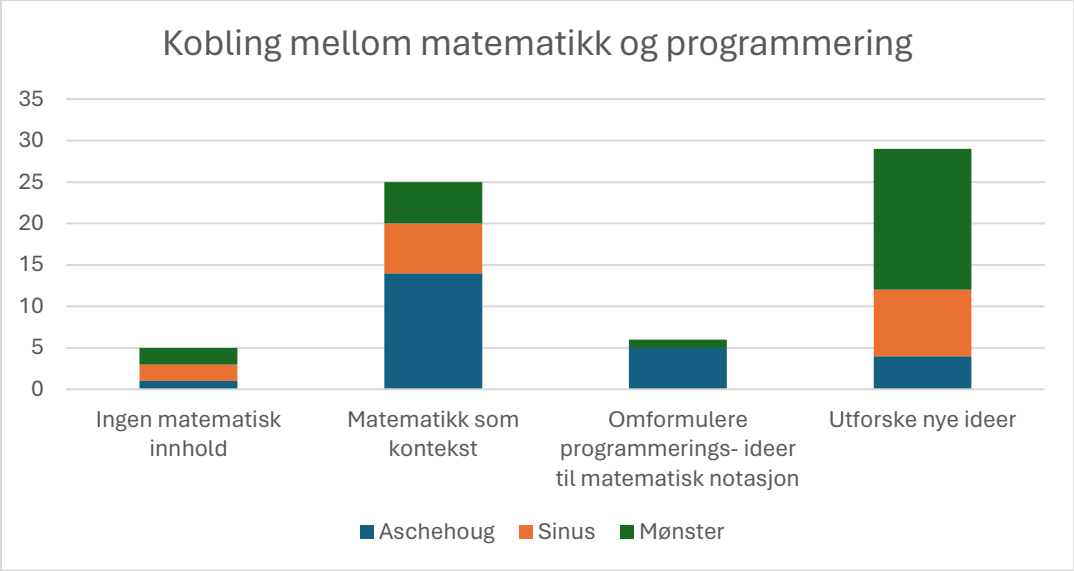



*Figur 18 Fordeling av programmeringsoppgaver i de ulike kapitlene i Aschehoug, Sinus og Mønster*

I figur 18 ser vi en oversikt over hvilke kapitler som inneholder programmeringsoppgaver og hvor mange oppgaver som finnes i hvert kapittel. Alle kapittelnavnene er tatt med og gitt en farge, selv om det ikke finnes programmeringsoppgaver i kapittelet. Det er for å illustrere også hvilke kapitler som ikke inneholder programmeringsoppgaver. Hvis kapittelet ikke har en programmeringsoppgave, vises ikke fargen i sektordiagrammet. I Aschehoug finnes programmeringsoppgaver i seks av sju kapitler. I sinus finnes programmeringsoppgaver i fire av syv kapitler og i mønster finnes programmeringsoppgaver i fem av seks kapitler. I Aschehoug er de fleste oppgavene knyttet til de to første kapitlene; «tall» og «algebra». I de to kapitlene handler mesteparten av oppgavene om å forstå eller lage et program der man regner med oddetall, partall, trekanttall og lignende. Som nevnt tidligere i delkapittelet, er slike oppgaver knyttet til det matematiske begrepet «aritmetikk». Programmeringsoppgavene i Sinus er hovedsakelig lagt til modeller og funksjoner og vekstfart og derivasjon, og handler om henholdsvis prosentregning og numerisk likningsløsning. I Mønster kommer halvparten av programmeringsoppgavene fra algebrakapittelet. Alle disse oppgavene kommer kun fra delkapittelet: numerisk løsning av likninger. En del oppgaver ligger også i kapittelet: vekstfart og likninger. I dette kapittelet kommer alle oppgavene fra delkapittelet: numerisk derivasjon. Det er verdt å merke seg at numerisk løsning av likningssystemer hører til vekstfart og derivasjonskapittelet i Sinus, men i algebrakapittelet i Aschehoug. Det er verdt å merke seg at mange oppgaver med programmering i Sinus og Mønster er knyttet til numeriske metoder, mens numeriske metoder ikke er nevnt i Aschehoug. I gjengjeld har Aschehoug satt søkelys på programmeringsoppgaver knyttet til «mønster» og «aritmetikk».

### 4.2.3 Delanalyse 3 - Kobling mellom programmering og matematikk

I delanalyse 3 har jeg undersøkt hvordan programmeringsoppgavene knyttes til matematikk. Som nevnt i analyseprosessen (4.1.3) har jeg kategorisert oppgavene under de fire kodene: «ingen matematisk innhold», «matematikk som kontekst», «omformulere programmeringsideer til matematisk notasjon» og «utforske nye ideer». I figur 19 ser vi fordelingen av kategoriene i de tre lærebøkene. Jeg vil videre presentere hver kategori for seg og gi eksempler på oppgaver som hører til hver kategori.



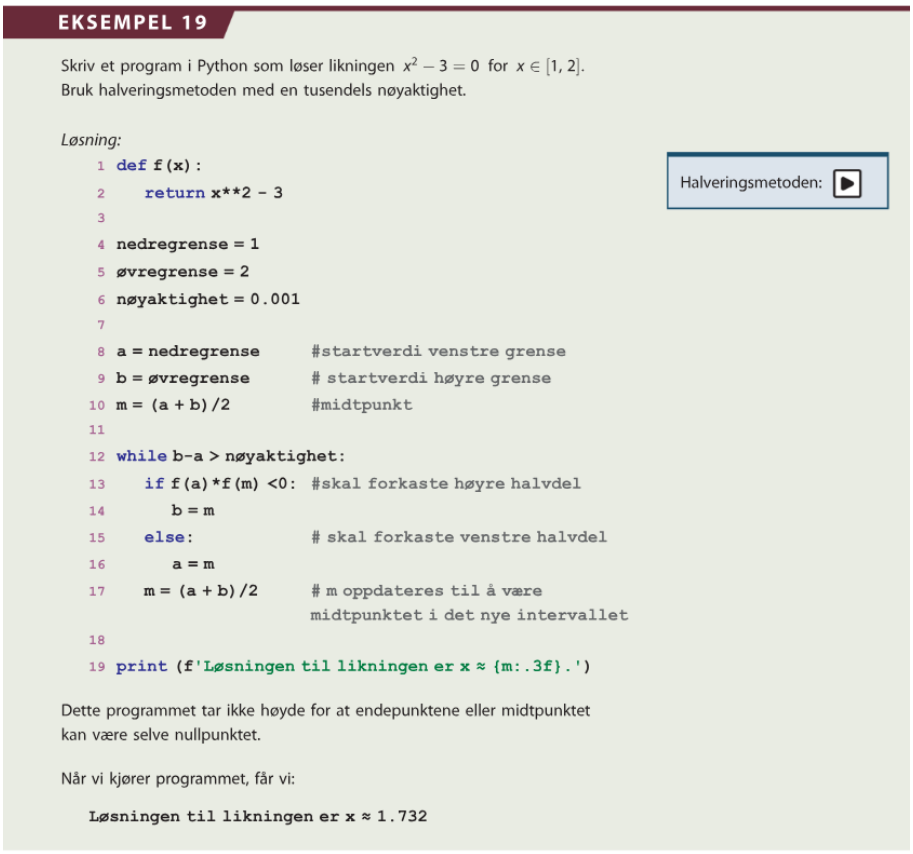

*Figur 19 Antall oppgaver under fire kategorier som sier noe om koblingen mellom matematikk og programmering. Noen oppgaver er kategorisert under flere kategorier.*

**Oppgaver uten matematisk innhold**

Som det som frem av delanalyse 2, og figur 19, er det 5 oppgaver uten matematisk innhold. En av oppgavene uten matematisk innhold er vist i figur 15 (i kapittel 4.2.2). Her blir elevene bedt om å utvide et program fra et eksempel som er gitt i læreboka (figur 20). Eksempelet i figur 20 demonstrerer hvordan man kan skrive et Python-program som finner nullpunktene til en funksjon ved hjelp av halveringsmetoden. Oppgaven elevene får er å utvide programmet gitt i figur 20 slik at programmet printer hvor mange ganger while-løkka kjøres. Til tross for at oppgaven er knyttet til et eksempel med en god og fornuftig kobling til matematikk, mangler selve oppgaven matematisk innhold. Det virker som om oppgaven er inkludert kun som en ren programmeringsøvelse, der elevenes forståelse av løkker og evne til å forstå og skrive kode blir testet.

Med noen få justeringer kunne oppgaven gitt elevene muligheten til å utforske halveringsmetoden. Antallet gjennomkjøringer av løkken avhenger av valget av øvre og nedre grense samt posisjonen til nullpunktet. For eksempel, hvis nullpunktet er 0,5 og øvre og nedre grense er henholdsvis 0 og 2, og **hvis** programmet hadde sjekket om midtpunktet er nullpunktet, ville to halveringer vært nødvendig før nullpunktet blir funnet. Imidlertid sjekker ikke programmet i figur 20 om midtpunktet er nullpunktet, noe som fører til at while-løkken kjører mer enn to ganger. Dette aspektet kunne elevene ha diskutert, noe som ville gitt en bedre forståelse av halveringsmetoden og hvorfor de må

endre koden i neste oppgave (der bes elevene om å legge til kode som sjekker om nullpunktet er endepunktene eller midtpunktet). Allikevel, slik oppgaven står nå, er den en ren programmeringsoppgave der det kun kreves at elevene definerer en variabel som blir oppdatert inne i løkken, etterfulgt av en utskrift av variabelens verdi.



**EKSEMPEL 19**

Skriv et program i Python som løser likningen $x^2 - 3 = 0$ for $x \in [1, 2]$.
Bruk halveringsmetoden med en tusendels nøyaktighet.

*Løsning:*

Halveringsmetoden:

```
1  def f(x):
2      return x**2 - 3
3
4  nedregrense = 1
5  øvregrense = 2
6  nøyaktighet = 0.001
7
8  a = nedregrense       #startverdi venstre grense
9  b = øvregrense        # startverdi høyre grense
10 m = (a + b)/2         #midtpunkt
11
12 while b-a > nøyaktighet:
13     if f(a)*f(m) <0:  #skal forkaste høyre halvdel
14         b = m
15     else:             # skal forkaste venstre halvdel
16         a = m
17     m = (a + b)/2     # m oppdateres til å være
                         midtpunktet i det nye intervallet
18
19 print (f'Løsningen til likningen er x ≈ {m:.3f}.')
```

Dette programmet tar ikke høyde for at endepunktene eller midtpunktet kan være selve nullpunktet.

Når vi kjører programmet, får vi:

```
Løsningen til likningen er x ≈ 1.732
```

*Figur 20 Et program som bruker halveringsmetoden for å finne nullpunktet til en funksjon. I Mønster (Kalvø et al., 2020, s. 263).*

Et annet eksempel på en oppgave uten matematisk innhold er vist i figur 6 (i kapittel 4.2.1). Oppgaven har noe matematikk i seg i form av heltall og sammenligning av heltallene. Ettersom begrepene er såpass enkle for en typisk elev som tar matematikk 1T, har jeg valgt å kategorisere oppgaven uten matematisk innhold.

**Oppgaver med matematikk kun som kontekst for oppgaven**

I figur 19 ser vi at det er 25 oppgaver der matematikk kun er kontekst for oppgaven. Det er oppgaver som trekker på kompetansemål fra tidligere trinn og ser ut til å være inkludert hovedsakelig for å lære elevene programmering. Et eksempel på en slik oppgave, der matematikk kun er kontekst for oppgaven, finnes i figur 21. For elever på videregående nivå kan oppgaven i figur 21 brukes for å lære programmering, men den gir ikke mulighet for utforsking av nye matematiske begreper. Oppgaven handler om hvordan programmet fungerer og legger vekt på programmeringsbegreper. For å løse programmet bør elevene forstå vilkår og variabler, som skal være kjent fra 4 trinn (Kunnskapsdepartementet, 2019).

Oppgaven bruker modulo-operatoren for å sjekke om et heltall delt med 3 har en rest. Modulo er sterkt knyttet til divisjon og spesielt heltallsdivisjon, som elevene er kjent med

fra barnetrinnet (Kunnskapsdepartementet, 2019). Allikevel er elevene mest sannsynlig ikke kjent med selve modulo-operatoren fra før, uansett er det sannsynlig at de kan forstå operatoren ut ifra konteksten. Modulo-operatoren brukes hovedsakelig i programmering, og er ikke mye brukt i matematikken på grunnskolen og på videregående. Derfor vil oppgaven være mer knyttet til opplæring av programmering enn opplæring av matematikk. I oppgave b), i figur 21, kan man observere at spørsmålet kun handler om forståelse av programmering og har ingen direkte tilknytning til matematikk.

En annen observasjon er hvor lite hensiktsmessig dette programmet er. Det finnes mange enklere måter å finne ut om et tall er delelig med tre. Hvis elevene har tilgang til hjelpemidler, kan de bruke en kalkulator til å dele på 3 og se om svaret inneholder desimaltall eller ikke. Hvis de må løse programmet uten hjelpemidler, kan de bruke ulike divisjonsstrategier som de skulle ha lært i 4. trinn (Kunnskapsdepartementet, 2019). Det er nettopp det at elevene skal være kjent med divisjonsstrategier så tidlig som gjør at denne oppgaven kategoriseres som en oppgave der matematikk kun er kontekst for oppgaven. Det er dermed tydelig at denne oppgaven er utformet for å lære elevene programmering og ikke matematikk. Oppgaven kunne derimot vært noe nyttig dersom elevene er nødt til å bruke modulo for å løse oppgaver senere, men en forklaring av begrepet hadde kanskje vært nok for de fleste elevene.


2.139

Programmet nedenfor undersøker om et vilkårlig tall er delelig med 3:

```
1  a = int(input("Skriv inn et heltall: "))
2
3  if a % 3 != 0:
4    print(a, "er ikke delelig med 3.")
5  else:
6    d = a/3
7    print(a, "er delelig med 3, og", a, "/3 blir", d)
```

**a** Forklar hvordan programmet finner ut om a er delelig med 3.
**b** Hva skjer hvis den nederste linja står uten innrykk?

*Figur 21 Oppgave der matematikk kun er en kontekst for oppgaven. Fra Aschehoug (Borge et al., 2020, s. 97).*

Andre typer oppgaver der matematikk kun fungerer som kontekst for oppgaven inkluderer oppgaver av typen «Tenk på et tall». Aschehoug presenterer tre slike oppgaver, hvorav én er illustrert i figur 12 (i kapittel 4.2.2). Disse oppgavene innebærer enkel aritmetikk og hovedsakelig de fire grunnleggende regneartene. Siden elevene bruker matematikk som allerede skal være kjent, er koblingen mellom programmering og matematikk svak. Programmering er heller ikke en hensiktsmessig måte å løse disse problemene på, da algoritmene kan løses med enkle regnestykker. Det kan tenkes at oppgavene er tatt med som en slags innledning av algoritmer, da oppgavene finnes i første kapittel av boka.

Det er interessant å merke seg at Aschehoug er den eneste læreboken som inkluderer «Tenk på et tall»-oppgaver. Dette kan antyde at de andre lærebokforfatterne ikke anser slike oppgaver som relevante for å oppnå kompetansemålene i 1T, verken kompetansemålet med programmering eller de med ren matematikk. Hvis elevene skulle bevise at algoritmen alltid gir det samme svaret, så kunne oppgaven ha vært mer knyttet til kompetansemålene. Det er fordi et kompetansemål er at «Elevene skal utvikle bevis i relevante matematiske emner» (Kunnskapsdepartementet, 2020). Dette kunne vært oppnådd ved å tilordne det vilkårlige tallet som en variabel, for eksempel $x$, og deretter løse uttrykket for $x$. Likevel ville ikke programmering ha bidratt til å gi en hensiktsmessig

kobling mellom programmering og matematikk, da det er like enkelt, om ikke enklere, å løse uttrykk for hånd eller ved hjelp av CAS.

**Omformulere programmeringsideer til matematisk notasjon**

Som vi ser i figur 19 finnes det 6 oppgaver i bøkene der elevene blir bedt om å omformulere programmeringsideer til matematisk notasjon. De fleste oppgavene er av typen «Lag et program […]» og disse er knyttet til handlingene «forme og skape» eller «følge en regel». Oppgaver der elevene får omformulere programmeringsideer til matematisk notasjon er sterkt knyttet til handlingen «forklare». Ettersom det kun er 10 oppgaver av handlingen «forklare», kan det være en forklaring på at det også er få oppgaver der elevene må omformulere programmeringsideer. Et eksempel på en oppgave der elevene må omformulere programmeringsideer er vist i figur 22. Her må elevene forklare hva programmet gjør. For å kunne forklare programmet, bør de forstå løkker og hvordan løkker knyttes til figurmønster. Oppgaven ber ikke eksplisitt om at elevene må bruke matematisk språk, men et naturlig svar vil inneholde det faktum at programmet skriver ut noen av trekanttallene. Et mer detaljert svar vil kunne inkludere hvordan programmet legger til et større tall for hver runde i løkka, noe som indikerer at elevene må forstå hvordan trekanttallene utvikler seg. For å knytte matematikk og programmering tettere sammen kunne oppgaven spurt om hvordan programmet fungerer, ikke bare hva det gjør. På den måten ville oppgaven vært mer utfordrende og lagt bedre til rette for at elevene kunne forklart mer. De fleste oppgavene jeg har karakterisert som å omformulere programmeringsideer er ikke tydelig knyttet til kategorien. Det er ingen av oppgavene som legger opp til en utdypende forklaring av programmeringsbegreper og matematikken involvert i oppgaven. Dermed har jeg tatt med oppgaver der jeg mener det er potensiale for at elever kan omformulere programmeringsideer til matematisk notasjon.


UTFORSK

Mengden {1, 3, 6, 10, ...} inneholder *trekanttallene*.

**a** Lag en figur som illustrerer de fire første trekanttallene og som samtidig viser hvorfor de har fått dette navnet.

**b** Hva er det femte og det sjette trekanttallet?

**c** Hvilken sammenheng er det mellom trekanttall og rektangeltall?

**d** Kan du finne en sammenheng mellom trekanttallene og de naturlige tallene?

**e** Hva gjør dette programmet?

```
1  tall = 0
2
3  for n in range(1, 11):
4    tall = tall + n
5    print(tall)
```

*Figur 22 Oppgave der elevene må omformulere programmeringsideer til matematisk notasjon. Fra Aschehoug (Borge et al., 2020, s. 17).*

**Utforske nye matematiske ideer**

Av de 63 oppgavene som ble gjennomgått, gir 29 oppgaver elevene mulighet til å utforske nye matematiske ideer (figur 19). Hovedvekten av disse oppgavene er knyttet til det matematiske begrepet «tilnærming», spesielt gjennom oppgaver knyttet til halveringsmetoden eller Newton-Raphson-metoden. Disse metodene er designet for å finne nullpunkter til en funksjon numerisk. Et eksempel på en slik oppgave ble vist i figur

14. Å finne nullpunkter til en funksjon numerisk er mye enklere å finne på verktøy som GeoGebra, hvor man bare trenger å skrive inn likningen og trykke på «numerisk løsning»-knappen. Det som er negativt med å bruke «numeriske løsning»-knappen er at man ikke får se noe av utregningene som skjer. Her har programmering en nyttig rolle, da man kan se fremgangsmåten for numeriske beregninger. Ved å lese programmer med numerisk tilnærming eller skrive dem selv, kan elevene forstå hvordan man kommer frem til det estimerte tallet.

Det er mange oppgaver som blir knyttet til numerisk løsning av likninger, spesielt i Sinus og Mønster, hvor det er flere tilsvarende oppgaver etter hverandre. Altså oppgaver som alle krever numerisk løsning av likninger, der kun likningen i oppgaven endrer seg. Etter å ha gjort to oppgaver av samme type, vil nok ikke den tredje oppgaven føre til at elevene utforsker noe nytt. Jeg valgte å karakterisere oppgavene individuelt, selv om mange oppgaver var nesten like. Det var for å få en helhetlig oversikt over oppgavene og for å kunne telle oppgavene for en kvantitativ vurdering. I tillegg ville en analyse som ser på alle oppgavene i helhet krevd mye tid, og er derfor utenfor masteroppgavens omfang.

Et annet tema der elevene får mulighet til å utforske matematiske ideer handler om sparing og lån med innskudd. Det involverer oppgaver der tallene for hvert år avhenger av resultatene fra den forrige utregningen. Tidligere har denne typen oppgaver blitt løst ved hjelp av regneark, der man kan få en oversikt over periode for periode. Her er programmering et nyttig verktøy som kan forenkle utregningen, og som gir et alternativ til regneark. Ved å bruke løkker kan man tenke på hver runde i løkken som en spareperiode. Deretter kan man legge til penger og kjøre løkken på nytt. Det vil altså være ideen av en rekursjon som tas i bruk. Slike oppgaver vil også kunne knyttes til den sentrale ideen om mønster, ettersom hver spareperiode gjentar seg på en forutsigbar måte, og dermed er et mønster. Programmering vil være en nyttig metode for å løse oppgaver om sparing og lån med innskudd, ettersom den rekursive regelen er enklere å finne enn den eksplisitte regelen.

# 5. Diskusjon

Jeg vil i dette kapittelet diskutere funnene fra analysen og drøfte de opp mot teori og egne refleksjoner. Jeg kommer til å diskutere de to forskningsspørsmålene hver for seg og deretter diskutere rammeverket og studiens begrensninger.

## 5.1 Hva karakteriserer programmeringsoppgaver i lærebøker i matematikk 1T og hvordan knyttes oppgavene til PRIMM og UMC?

I delanalyse ble det identifisert hvilke handlinger oppgavene krever av elevene, mens delanalyse 2 avdekket hvilke matematiske begreper og programmeringsbegreper som finnes i oppgavene. Gjennom analysen dukket det opp to resultater som jeg vil diskutere videre. Det første er at det var et ensidig fokus av handlinger i oppgavene. Dette funnet kan knyttes opp mot arbeidsmetodene PRIMM og UMC som jeg også vil bruke for å foreslå hva slags kombinasjoner av handlinger som kan gi mulige forbedringer av oppgavene. Det andre resultatet jeg vil diskutere er mangel på feilsøking i oppgavene.

### 5.1.1 Ensidig fokus av handlinger

Figur 5 gir en oversikt over hvilke handlinger som finnes i de tre lærebøkene. Fra figuren kan det observeres hvilke lærebøker som inneholder oppgaver knyttet til de ulike handlingene. Aschehoug har oppgaver som dekker alle handlingene, Mønster har ingen oppgaver med handlingene «feilsøke» eller «forestille seg», og Sinus mangler handlingene: «følge en regel», «feilsøke», «forklare» eller «forestille seg». Når jeg analyserte handlingene i bøkene oppdaget jeg at det var et ensidig fokus med tanke på handlinger. 46 av de 63 oppgavene har kun én handling, som videre gir at kun 17 av de 63 oppgavene har to eller flere handlinger. Aschehoug var den boka som hadde flest handlinger i oppgavene. Boka har 12 av de 17 oppgavene med flere handlinger. Det er kanskje ikke så overraskende ettersom det var den boka som hadde oppgaver innenfor alle handlingene. Aschehoug hadde også flest deloppgaver per oppgave og det var vanlig at det ble bedt om ulike handlinger i de forskjellige deloppgavene.

Fra figur 5 ser vi også at det er flest oppgaver med handlingen «forme og skape», etterfulgt av «følge en regel». Som nevnt har oppgavene ofte kun en handling, så disse to handlingene står som oftest alene. Oppgavene med «følge en regel» er oppgaver der elevene ikke nødvendigvis trenger å forstå hvordan de kan løse oppgaven med programmering, bare hvordan de knyttes til eksempelet. Det var litt variasjon i hvor mye av eksempelet som måtte endres, men et kjennetegn på alle oppgaver med «følge en regel» var at elevene ikke måtte ha en helhetlig forståelse av programmet i eksempelet. Dette er oppgaver som krever lite av elevene dersom de klarer å se hvilke tall de må bytte ut eller hva slags input de skal gi programmet. Selv om oppgavene krever lite av elevene, kan slike oppgaver være nyttige for elever på et faglig lavere nivå. Å skrive kode fra bunnen av kan være krevende, og for utforskende oppgaver kan gjøre elevene frakoblet (Benton et al., 2017, s. 122). Derfor må man finne en balanse mellom struktur og individuell utforsking. En ting som kunne forbedret oppgaver med «følge en regel» er om man kunne hatt deloppgaver med «forklare» eller «forutse» (Bråting & Kilhamn,

2022, s. 604). Da kunne man «tvunget» elevene til å sette seg inn i koden som var gitt i eksempelet og man ville fått en bedre kobling mellom programmering og matematikk. Likevel forekommer det sjelden oppgaver som inneholder handlingene «forklare» eller «forestille seg» etter handlingen «følge en regel» i datasettet. Den sammensetningen finnes kun i 3 oppgaver. Ifølge Kunnskapsdepartementet (2020) krever kompetansemålet at elevene skal kunne løse problemer ved hjelp av programmering. Det kan forklare den høye andelen oppgaver som kun har handlingen «forme og skape». Uansett lærer ikke elever programmering automatisk ved å lage egne programmer (Flø, 2021, s. 5). Det vil også være lettere for en elev som ikke kan så mye programmering å skrive av et program, prøve å forstå det og gjøre mindre justeringer, enn at eleven skal skrive et program fra bunn av (Flø, 2021, s. 6). En slik arbeidsmetode kan knyttes til rammeverket UMC, som jeg vil diskutere mer i neste delkapittel.

### 5.1.2 Kobling av oppgavene mot PRIMM og UMC

Som nevnt er oppgavene i datasettet ensidig med tanke på handlinger, noe jeg anser som negativt. I forrige avsnitt kom det frem at oppgaver med handlingen «følge en regel» kunne blitt forbedret ved å legge til «forklare» eller «forestille seg», men jeg vil også påstå at de fleste oppgavene kan bli forbedret ved å legge til flere handlinger. Argumentasjonen for påstanden kommer fra at handlingene fra Bråting og Kilhamn (2022) kan knyttes til handlingene i arbeidsmetoden kalt PRIMM. I tabell 4 har jeg presentert alle stegene i PRIMM og hvilke handlinger de kan kobles opp mot. Det er ikke nødvendigvis klart hvordan «feilsøke» og «forklare» kan knyttes til PRIMM, men begge handlingen finnes i det tredje steget, «investigate» (Sentance et al., 2019, s. 11).

| Steg i PRIMM: | Handlinger: |
|---|---|
| P: Forutse et program | Forestille seg |
| R: Kjøre et program | Følge en regel |
| I: Undersøke et program | Feilsøke, forklare, finne regel |
| M: Modifisere et program | Forme og skape |
| M: Lage et program | Forme og skape |

*Tabell 4 Kobling mellom steg i PRIMM (Sentance et al., 2019, s. 10) og handlingene fra Bråting og Kilhamn (2022, s.599).*

Som nevnt i teorien (2.10.2) er PRIMM en metode for å lære programmering som tilbyr stillasbygging rundt elevenes forståelse. Hvis oppgaver følger PRIMM-metodikken så kan det føre til at elevers opplevde vanskelighetsgrad av oppgaver reduseres (Flø, 2021, s. 5-6). PRIMM-metoden består av de fem delene som er presentert i tabell 4. Aktivitetene i PRIMM metodikken utfordrer elevene på resonnering og problemløsing og vil dermed være godt knyttet til AT (Matematikksenteret, u.å.-a), på samme måte som handlingene også er godt knyttet til AT (tabell 2). Dermed ville elevene kunne fått større utbytte av programmeringsoppgaver dersom oppgavene hadde inneholdt flere handlinger enn de gjør i lærebøkene jeg har undersøkt. Det finnes allikevel noen oppgaver som inkluderer flere handlinger, blant annet oppgaven i figur 10, hvor et likningssystem skal løses med formlene til Ada Lovelace. Oppgaven inneholder handlingene «følge», «feilsøke», «forme og skape» og «forklare». Hvis vi ser på tabell 4, ser vi at handlingene dermed kan knyttes til: «run», «investigate» og «modify» i PRIMM.

Som nevnt i kapittel 2.7.2 er UMC en undervisningsmetode i programmering der elevene først bruker ferdige kodesnutter som de kjører, og prøver å forstå. Når elevene forstår hvordan koden fungerer kan de endre programmet. Til slutt kan elevene lage sin egen kode. Denne didaktiske undervisningsmetoden kan bedre elevenes AT (Sentance et al., 2018, s. 115). Til tross for denne metodens positive innvirkning, er det få oppgaver som følger denne metoden. «Use-modify» finnes i oppgaver hvor elevene først må «følge en regel» for deretter å «forme og skape». Denne rekkefølgen finnes i 7 av de 63 oppgavene, der 3 er fra Aschehoug, 2 fra Sinus og 1 fra Mønster. Ifølge Maugesten et al. (2021, s. 7) kan «use» og «modify» være tilstrekkelig i starten når man lærer seg programmering. Det er fordi målet med programmering i skolen er at elevene skal bli demokratiske borgere som skal få utvikle forståelse for hvordan den digitale verden er oppbygd, ikke bli programmerere. «Create» vil også være den mest utfordrende fasen i arbeidsmetoden. Fra resultatene mine virker det som at handlingen «forme og skape», som vil være lik «create», er noe lærebokforfattere setter større pris på. 37 av de 63 programmeringsoppgavene inneholder handlingen «forme og skape». Ettersom så få oppgaver inneholder et sett med handlinger som kan knyttes til PRIMM og UMC, så tyder det på at programmeringsoppgavene i lærebøkene antageligvis ikke har et overordnet didaktisk konsept de er strukturert etter.

Oppgaver med flere handlinger kan passe bedre inn i didaktiske paradigmer som PRIMM og UMC, som blir sett på som nyttige for opplæring i programmering. Likevel betyr ikke det at oppgaver med flere handlinger alltid er bedre. En positiv ting med oppgaver som ikke har flere handlinger, for eksempel en «forme og skape»-oppgave, er at elevene får øvelse i å dekomponere oppgaver selv. Å dekomponere er en av begrepene i AT og handler om å kunne bryte opp et problem i mindre bestanddeler og håndtere hovedproblemet i mindre biter (Gjøvik & Torkildsen, 2019, s. 33). Hvis en oppgave har mange deloppgaver, så kan det være at oppgaven allerede er dekomponert for elevene. En annen ting å ta til betraktning er at PRIMM og UMC hovedsakelig er utviklet for å gjøre det lettere å lære programmering. Hvis man ser på kompetansemålet som inkluderer programmering for matematikk 1T (Kunnskapsdepartementet, 2020), og tidligere kompetansemål (tabell 1), kan det tyde på at opplæringen av programmeringsbegreper er ferdig når elevene begynner på matematikk 1T. Da er målet å bruke programmering på en fornuftig måte. Det kan fortsatt være nyttig å bygge opp noen oppgaver etter trinnene i PRIMM og UMC. Det vil alltid være forskjell i elevers faglige nivå og hvis man har en god variasjon av oppgaver, så vil man kunne treffe flere elevers nivå. Samtidig er programmering en ferdighet som krever praksis og øvelse. Oppgaver med «forme og skape» vil kunne passe for elever som trenger mer utfordring (Maugesten et al., 2021, s. 7), mens oppgaver med flere deloppgaver og ulike handlinger kan passe for elever som ikke er helt trygge på programmering enda.

### 5.1.3 Mangel på feilsøking i oppgaver

Noe av det mest oppsiktsvekkende i diagrammet i figur 5, er hvor få oppgaver som har handlingen «feilsøke». I tillegg finnes handlingen kun i Aschehoug. Der er det to oppgaver med «feilsøke», og en av de er vist i figur 10. Feilsøking anses som svært nyttig i programmering og for utvikling av AT, og er noe som gjenspeiles flere steder i litteraturen. Som vist i figur 1 er feilsøking en av arbeidsmetodene til den algoritmiske tenkeren (Utdanningsdirektoratet, 2019a). I følge Sentance et al. (2018, s. 29) er testing

og feilsøking essensielt i enhver form for problemløsning. Når man lærer å programmere, er en viktig kompetanse å kunne beherske systematisk testing av programmet for både gyldige og ugyldige inndata. Helt spesifikt utvikles ferdigheter i abstraksjon og dekomponering når man feilsøker (tabell 2). I tillegg fremmer Bråting og Kilhamn (2022) feilsøking som fundamentalt for å lære programmering, og Brennan og Resnick (2012, s. 7) hevder at ting sjeldent fungerer akkurat slik man forestiller seg, og at det dermed er kritisk for programmerere å utvikle strategier både for å håndtere problemer og for å forutse problemer.

Man kan argumentere for at elevene må feilsøke uavhengig om oppgaven legger opp til det eller ikke, spesielt i videregående skole. I matematikk 1T er det mest vanlig å bruke TBP og det medfører et større behov for å huske syntaks enn BBP som ofte blir brukt i lavere trinn (Moors et al., 2018, s. 58). Å forstå feilmeldinger er også veldig nyttig, og er en ferdighet man må lære. I oppgaven i figur 10, som handlet om løsning av likningssystemer ved hjelp av formlene til Ada Lovelace, vil elevene kunne bruke feilmeldingen til å se at det ikke går an å dele på null. Det gir elevene en pekepinn på hvorfor programmet ikke fungerer. I neste deloppgave får de muligheten til å legge inn kode som forhindrer at programmet prøver å dele på null. Når nevneren i likningen er null, har likningssettet enten ingen eller uendelig mange løsninger. For å løse deloppgaven må elevene vise at de forstår «vilkår» ved å sette inn if-setninger eller if-elif-else setninger. Hvis man sammenligner med resultatene fra Bråting og Kilhamn (2022) sin analyse ser man at også de fant lite av handlingen «feilsøke». I min analyse var feilsøking med i 3,17% av oppgavene og i Bråting og Kilhamn (2022) sin undersøkelse fant de feilsøking i 4,87% av oppgavene. Det er snakk om et lite antall i begges undersøkelser, men hvis man tar man hensyn til at det er lettere å få feilmeldinger i TBP enn i BBP, på grunn av syntaks, så burde det kanskje vært en større andel oppgaver med handlingen «feilsøke» i oppgaver med TBP.

En mulig årsak til at det er lite feilsøking i oppgavene kan være at lærebokforfattere ser på feilsøking som for rettet mot programmering og at det dermed ikke passer godt nok i matematikkoppgaver. Samtidig vil dette være litt rart da alle tre forlagene har oppgaver som er tydelig rettet mot at elevene skal lære programmering, og ikke matematikk (blant annet oppgaven i figur 15). Forfatterne kan tenke at feilsøking er en arbeidsmetode som bør læres bort av læreren slik at elevene kan bruke det i alle typer oppgaver. Allikevel er opplæringa av programmering satt til matematikkfaget (Utdanningsdirektoratet, 2019b) og da burde feilsøking fått en større plass i matematikkutdanningen.

Ifølge Sevik (2016, s. 25) var det en risiko for at programmering måtte vike til side i matematikken på grunn av plassmangel. Lærebokforfatterne har trolig også kjent på denne plassmangelen. Programmering tar mye av tiden i matematikkundervisning og det er også mange andre temaer som tar tid (Maugesten et al., 2021, s. 5). Dermed er lærebokforfatterne nødt til å vurdere hvor mange oppgaver de vil inkludere for hvert tema. De kan ha tenkt at de allerede hadde tilstrekkelig med programmeringsoppgaver og at de derfor måtte prioritere andre kompetansemål, noe som har ført til mindre vekt på feilsøking. Denne nedprioritering av programmering i matematikkfaget er en av årsakene til at «Teknologi for alle»-rapporten og Senter for IKT anbefalte at programmering burde innføres som et eget obligatorisk fag i grunnskolen.

Et kompetansemål for 8. trinn er at elevene skal «utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering» (Kunnskapsdepartementet,

2019). Læreplanen spesifiserer ikke om dette skal gjøres ved hjelp av BBP eller TBP, men alle kompetansemålene på ungdomsskolen kan nås ved hjelp av blokkbasert programmering (Flø, 2021, s. 5). Det kan dermed hende at noen kun lærer BBP, og ikke TBP, i 8. trinn. De elevene som hovedsakelig lærer BBP i 8. trinn lærer også å feilsøke BBP, og ikke TBP. Ettersom TBP krever et høyere nivå for å holde styr på kontekst og har et mer avansert språk (Moors et al., 2018, s. 58) så krever det mer og annerledes bruk av feilsøking enn ved bruk av BBP. Manglende fokus på feilsøking på høyere trinn er derfor uheldig, da dette er en viktig ferdighet som kan hjelpe elevene å forstå og rette feil i programmeringskode. Lærebokforfatterne kan ha tenkt at elevene har lært feilsøking for TBP i 8.trinn og at feilsøking derfor ikke trengte å være fokus i oppgavene for matematikk 1T. For videre forskning kunne det vært interessant å se hvor mange oppgaver som har «feilsøke» i lærebøkene for 8. trinn, og om det kobles til BBP eller TBP.

Selv om handlingen «feilsøke» ikke ble funnet så mye i oppgavene, kan visse sammensetninger av andre handlinger føre til at elevene feilsøker. Blant annet i oppgaver som følger UMC, så må elevene prøve og feile, stille seg spørsmål og undre seg (Maugesten et al., 2021, s. 6). Videre hevder Maugesten et al. (2021, s. 6) at å prøve og feile blir en naturlig del av arbeidet i modifyfasen. I tillegg kan feilsøking komme naturlig inn i undersøkelsestrinnet av PRIMM, da elevene har god nytte av å feilsøke eller teste koden for ulike input når de skal forstå hva koden gjør (Sentance et al., 2019, s. 11).

## 5.2 Hvilke temaer knyttes programmeringsoppgavene i matematikk 1T til og på hvilke måter knyttes programmering og matematikk sammen?

I delanalyse 2 kom jeg frem til at noen matematiske temaer har flere programmeringsoppgaver enn andre, og at det er forskjell i hvilke matematiske begreper de ulike bøkene fokuserer på. Det tyder på at lærebokforfatterne knytter programmering også til ulike sentrale ideer i matematikken. I første del av diskusjonen skal jeg diskutere hvilke temaer programmeringsoppgaver knyttes til før jeg diskuterer lærebøkenes ulike tilnærminger til sentrale ideer. Deretter vil jeg diskutere resultatene fra delanalyse 3 og ta for meg oppgaver uten matematisk innhold, oppgaver der matematikk er kontekst for oppgaven, oppgaver der elevene må omformulere programmeringsbegreper til matematisk notasjon og til slutt oppgaver der elevene får utforske nye matematiske ideer.

### 5.2.1 Hvilke temaer knyttes programmeringsoppgavene til?

Ifølge Munthe (2023, s. 9) er en utfordring med programmering i matematikkfaget at programmering blir tvunget inn i oppgaver som bedre kan løses ved hjelp av Excel, Geogebra, kalkulator eller penn og papir. Da forsvinner både gleden og nytten av å bruke verktøyet. Som vist i figur 13 har flere oppgaver blitt knyttet til det matematiske begrepet «tilnærming». Programmeringsoppgaver med tilnærming vil være nyttig fordi elevene nå kan se metoden som ligger bak, slik at de unngår en «black box». En «black box» i dette tilfellet er et system som produserer et resultat der brukeren ikke har mulighet til å se hvordan systemet fungerer ("Black box," u.å.). Altså et program som tar inn en «input» og sender ut en «output». Brukeren trenger ikke å forstå programmet, og

får heller ikke mulighet til å forstå hvordan programmet fungerer. Det eneste brukeren trenger å forstå er hvordan man gir programmet en «input». Tidligere har elevene brukt Geogebra til å finne numeriske løsninger, der elevene bare har behøvd å trykke på knappen: «numerisk løsning». Geogebra gir ingen informasjon om hva som skjer når elevene trykker på «numerisk løsning»-knappen. Programmet tar kun inn en input og gir ut et svar basert på inputen. Programmering kan være med på å vise metoden som ligger bak disse programmene (Munthe, 2023, s. 9). En utdypende diskusjon av oppgaver med «tilnærming» som utforskende oppgaver kommer i kapittel 5.2.6.

Det finnes derimot flere programmeringsoppgaver i lærebøkene som bedre kan løses for hånd, eller ved hjelp av Geogebra, og som ikke motvirker en såkalt «black box». Blant annet «tenk på et tall»-oppgaven i figur 12, som er kategorisert som en «matematikk som kontekst»-oppgave. Her bidrar ikke programmering til at elevene får utforske, diskutere eller forstå matematikk. En mer utdypende begrunnelse for påstanden kommer i delkapittel 5.2.4, der jeg diskuterer oppgaver med matematikk som kontekst. Oppgave 12 er også kategorisert under de matematiske begrepene «likning», «mønster» og «aritmetikk» og hører til et kapittel kalt «algebra».

Fra figur 18 ser vi at mange andre programmeringsoppgaver også er knyttet til kapitler kalt algebra. Det gjelder oppgavene fra Aschehoug og Mønster. Som nevnt tidligere er oppgavene i algebrakapittelet i Mønster kategorisert som «tilnærming», fordi alle programmeringsoppgavene i algebrakapittelet er knyttet til numerisk løsning av likninger. I Aschehoug finnes oppgaver kategorisert som «aritmetikk» i algebrakapittelet. I tillegg kan oppgaver med mønster og likninger knyttes til algebra (selv om jeg har valgt å ha «mønster» og «likning» som egne kategorier). I svenske læreplaner for matematikk er programmering inkludert som et emne i algebra (Bråting & Kilhamn, 2022, s. 605), så det kan tyde på at de ser på programmering tydelig knyttet til algebra. Hvis man ser på likninger og mønster som en del av algebra, ser vi fra figur 13 at alle bøkene har en del oppgaver knyttet til algebra. Ofte blir programmering knyttet til algebra på grunn av at både programmering og algebra inneholder variabler (Bråting & Kilhamn, 2022, s. 607). Ifølge Sande (u.å, s. 4) fungerer variabler på samme måte når man programmerer som når de brukes i matematikk. Det mener jeg er en feilaktig påstand. Det er likheter mellom variabler i matematikk og programmering, men de oppfører seg ikke likt. En av de vanskeligste programmeringsbegrepene for en nybegynner vil faktisk være variabeltilordning og evaluering (Kohn, 2017, s. 345).

> Studenter virker å tro at maskinen er i stand til algebraisk manipulasjon av uttrykkene som brukes i programmet: under evaluering ville en variabel bli erstattet av sitt definerte uttrykk i stedet for en tidligere bestemt numerisk verdi. Med andre ord: studenter ser ut til å forvente at datamaskinen utfører beregninger på samme måte som de har lært i matematikk. (Kohn, 2017, s. 349, egen oversettelse).

I dette sitatet er en «maskin» definert som et abstrakt konsept som henviser til hvordan et program er utført (Kohn, 2017, s. 349). Altså kan man tenke på «maskinen» som programmet Python. En typisk misforståelse kan dermed være uttrykket x=x+1. Mange elever vil slite med dette uttrykket (Kohn, 2017, s. 349). Fra et matematisk syn vil denne likningen være uløselig, men i et programmeringsspråk betyr uttrykket at man skal evaluere høyre siden med x sin tidligere definerte verdi, og lagre resultatet som en verdi av variabelen x. Ifølge Kohn (2017, s. 349) vil ikke elever se at maskinen/Python er en

ren numerisk maskin uten kunnskap om algebraiske forhold. Dermed vil koblingen til algebra kunne by på misforståelser hos elevene. På bakgrunn av disse misforståelsene burde lærebøkene og lærere tydelig skille mellom variabler i matematikken og variabler i et programmeringsspråk. Det er verdt å merke seg at lærebøkene i min analyse ikke fremmer denne forskjellen. Blant annet etter at elevene får en oppgave om å løse likningssystemer med Ada Lovelace sine formler (figur 10, i 4.2.1), kunne det blitt presisert hvorfor denne metoden blir brukt. Mange elever er nok kjent andre metoder, sånn som innsettingsmetoden. Med CAS kan elever løse likningssystemer ved bruk av innsettingsmetoden. Det gjøres ved å skrive inn hver likning, markere alle likningene og trykke på knappen; $x =$. Ettersom programmeringsspråk ikke kan holde på ukjente verdier (uten at eksterne biblioteker er importert), kan ikke elevene bruke innsettingsmetoden på samme måte som de gjør for hånd eller ved hjelp av CAS. Disse aspektene burde lærebøker eller lærere presisere når programmering blir brukt knyttet til det matematiske temaet «likninger», eller når programmering knyttes til algebragrenen generelt.

### 5.2.2 Ulike tilnærminger til sentrale ideer

Fra delanalyse 2 kom det frem at det er store forskjeller i hvilke matematiske begreper lærebøkene knytter programmering til. Dermed har lærebøkene også ulike tilnærminger til sentrale ideer. På grunn av begrensninger av masteroppgaven har jeg satt søkelys mot to utvalgte sentrale ideer; numerisk tilnærming og mønster. Det har store konsekvenser at det er store forskjeller i hvilke tema programmering blir knyttet til. Når for eksempel Aschehoug ikke har noen oppgaver om tilnærming av nullpunkter så vil dette temaet mest sannsynlig bli utelatt i undervisningen for de elevene som bruker den som lærebok (Lepik et al., 2015, s. 132). Aschehoug bruker derimot andre metoder for å løse likninger, blant annet viser de et program som bruker ABC-formelen for å løse andregradslikninger.

En ting som er praktisk med halveringsmetoden og Newton-Raphson-metoden, som det finnes oppgaver om i henholdsvis Mønster og Sinus, er at det ikke er noe krav om at det må være en andregradsfunksjon, sånn som ved ABC-metoden. I tillegg kan man finne tilnærminger til nullpunkter som er vanskelig å finne analytisk. ABC-formelen kan brukes for å finne nullpunktene til en andregradsfunksjon, men formelen vil ikke kunne knyttes til den sentrale ideen om numerisk tilnærming, sånn som halveringsmetoden og Newton-Raphson-metoden vil. At elevene skal lære numeriske metoder for å løse likninger kan knyttes til et kompetansemål for matematikk 1T: «Utforske strategier for å løse likninger, likningssystemer og ulikheter og argumentere for tenkemåtene sine» (Kunnskapsdepartementet, 2020). I tillegg kan numeriske metoder knyttes til kompetansemålet om programmering, når programmering blir brukt for å løse oppgavene. Ifølge Sanne et al. (2016, s. 61) er numeriske metoder en ny gren av matematikk i skolen som ikke fikk innpass i læreplanene før LK20, men som nå ved hjelp av programmering har blitt mye mer aktuell. Forfatterne av Aschehoug har derimot ikke sett på numeriske metoder som nødvendig for at elevene skal kunne nå disse to kompetansemålene. De har derimot tatt med Newton-Raphson-metoden i sin R1 bok (Borgan et al., 2021). Det virker dermed som at de setter søkelys mot den sentrale ideen om numerisk tilnærming da. Jeg vil anta at de da knytter numerisk løsning av likninger til et kompetansemål for R1 som er:

1. «Forstå begrepene vekstfart, grenseverdi, derivasjon og kontinuitet, og bruke

disse for å løse praktiske problemer» (Utdanningsdirektoratet, 2020)

Numerisk løsning av likninger kan knyttes til kompetansemålet ettersom Newton-Raphson-metoden krever at man finner den deriverte verdien i flere punkter. Da brukes den deriverte for å løse praktiske problemer. Dermed ser alle lærebokforfatterne på programmering som nyttig for numerisk løsning av likninger og programmering kan dermed knyttes godt opp til den sentrale ideen om numerisk tilnærming. Forskjellen ligger i når de mener elevene burde lære det. Det er ikke tydelig for meg hvilket kompetansemål som numerisk løsning av likninger hører mest til. At det ikke er tydelig kan ha sammenheng med at LK20 har lagt mer vekt på dybdelæring og at det dermed ikke er like tydelige læreplanmål som før (Flø, 2021, s. 3). Det er derfor tydelig at numerisk løsning av likninger er et tema Mønster og Sinus har gått i dybden på (i matematikk 1T). Samtidig har Aschehoug hatt fokus på andre sentrale ideer enn numerisk tilnærming, og andre områder der programmering kan brukes som hjelpemiddel (i matematikk 1T). Som nevnt i kapittel 4.2.2 har Aschehoug fokus på blant annet en sentral ide knyttet til det matematiske begrepet «mønster».

I de neste avsnittene vil jeg gå over til diskusjonen om hvordan matematikk og programmering knyttes sammen. Først vil jeg ta for meg oppgaver uten matematisk innhold.

### 5.2.3 Oppgaver uten matematisk innhold

Som vist i delanalyse 2 (4.2.2) og 3 (4.2.3) finnes det 5 oppgaver i datamaterialet som er uten matematisk innhold. Dette er en stor forskjell fra Bråting og Kilhamn (2022, s. 603) sin undersøkelse der de kom frem til at 116 av 327 oppgaver var uten matematisk innhold. En mulig grunn til den store forskjellen kan være knyttet til forskjellen i de trinnene som er undersøkt. Bråting og Kilhamn (2022) så på 1.- 6. trinn og jeg har sett på 11. trinn. Hvis man undersøker kompetansemålene fra grunnskolen, ser man at elevene skal ha lært de grunnleggende programmeringsbegrepene før de begynner på videregående. Etter 6.trinn skal de blant annet ha lært å bruke variabler, løkker, vilkår og funksjoner i programmering til å utforske geometriske figurer og mønster (Kunnskapsdepartementet, 2019). Det betyr at elevene må forstå disse programmeringsbegrepene før 6. trinn er ferdig. I 11. trinn kan man anta at de grunnleggende programmeringsbegrepene er kjent. Det ser man av kompetansemålet der det står at elevene skal bruke programmering for å formulere og løse problemer (Kunnskapsdepartementet, 2020). Her er det ingen spesifikk kompetanse i programmering som står i fokus. Ettersom elevene skal lære flere programmeringsbegreper på barneskolen, kan man tenke seg at noe plass i lærebøkene på barnetrinnet går til ren opplæring av programmeringsbegreper og derfor er uten matematisk innhold. Både undersøkelsen fra Bråting og Kilhamn (2022) og to masteroppgaver fra Norge er med på å underbygge denne teorien (Stokkenes, 2022; Engen, 2022).

På videregående burde elevene være kjent med programmeringsbegreper og fokuset i oppgavene (sånn som i kompetansemålet) burde være å bruke programmering som en problemløsningsmetode. Allikevel må man ta hensyn til at programmering kom som et nytt tema i LK20 (Flø, 2021). Den gamle læreplanen i matematikk fellesfag (1-11. trinn) hadde gyldighet til Juli 2021 (Kunnskapsdepartementet, 2006). Det medfører at noen

elever begynte på matematikk 1T i august 2021, og skulle oppnå det nye kompetansemålet i programmering uten å ha vært igjennom programmering på grunnskolen. Dette kan ha gjort at lærebokforfatterne har valgt «enklere» programmeringsoppgaver i denne utgaven av boka, nettopp for å hjelpe de som ikke har lært noe om programmering fra grunnskolen. I Aschehoug står det: «Programmeringsoppgaver er tatt med der det er hensiktsmessig. Kommandoene som blir brukt følges opp av grundige forklaringer slik at forkunnskaper ikke er nødvendig» (Borge et al., 2020, s. 3). I tillegg har Mønster en opplæringsdel om programmering bakerst i boken og Aschehoug har oversikt over viktige Python-kommandoer bakerst i boka. At programmering i skolen er såpass nytt vil nok føre til at lærere må tenke over hvilke oppgaver som er relevante med tanke på ulike elevkull som har hatt ulik mengde programmeringsoppgaver gjennom studieløpet. Hvis jeg hadde analysert oppgavene i opplæringsdelen til Mønster ville det nok vært mange flere oppgaver uten matematisk innhold eller med matematikk som kontekst. Jeg tok ikke med opplæringsdelen i analysedelen ettersom det ikke hører til noen spesifikke temaer eller kapitler i boken.

Som nevnt i delanalyse 3 kan oppgaven i figur 15, som ber elevene om å finne ut hvor mange ganger en while-løkke kjøres i et program om halveringsmetoden, bli knyttet til matematikk med få justeringer. Oppgaven kan knyttes til matematikk dersom oppgaven legger opp til at elevene må forstå hva antall gjennomkjøringer av løkka har å si for hvor fort man finner nullpunktet til funksjonen. At oppgaver uten matematisk innhold kan knyttes til matematikk ved hjelp av noen justeringer gjelder også flere av de andre oppgavene. I figur 6 (4.2.1) ble det presentert en oppgave der elevene skal lage et program der det velges et tilfeldig heltall mellom 1 og 100 som skal tippes av brukeren. Ifølge Munthe (2023, s. 12) virker programmet i figur 6 i utgangspunktet ikke spesielt relevant for matematikk, men har en god del matematikk som elevene kan diskutere. Programmet kan fungere som en inngang til å forstå halveringsmetoden ettersom programmet bruker noen av det samme prinsippene. Allikevel hevder Munthe (2023, s. 12) at diskusjon, utforsking og forståelse er sentralt for å forstå sammenhengen. Ettersom oppgaven i figur 6 står i et annet kapittel enn halveringsmetoden, og den eneste handlingen er «forme og skape», vil nok ikke de fleste elevene klare å se sammenhengen til halveringsmetoden. Dermed vil de heller ikke klare å knytte programmeringen til matematikk. Oppgaven kan være hensiktsmessig dersom elevene får mulighet til å diskutere programmet. Noen spørsmål elevene kan diskutere, der elevene får bruk for AT, er:

1. Hva er den beste taktikken for gjetting?
2. Hvor mange gjetninger trenger man maksimalt hvis man bruker en best mulig taktikk?
3. Hvor mange gjetninger trenger man for et program som ber spilleren om å gjette på et tall mellom 1 og n (der n er et valgfritt heltall)?
   (Munthe, 2023, s. 12)

I tillegg burde oppgaven komme som en introduksjon til halveringsmetoden slik at elevene forstår sammenhengen mellom aspektene de diskuterer og fremgangsmåten til halveringsmetoden. Allikevel vil jeg ikke påstå at 5 oppgaver uten matematisk innhold er en så stor andel, i hvert fall ikke hvis man sammenligner med Bråting og Kilhamn (2022) sine resultater. I tillegg så kan vi håpe at lærere legger merke til slike oppgaver, som har potensial for å kunne knyttes til matematikk, og legger til deloppgaver der elevene må diskutere oppgaven eller resultatet. Da kan mange oppgaver faktisk få en kobling til

matematikk. I neste avsnitt vil jeg diskutere oppgaver der matematikk er kontekst for oppgaven.

### 5.2.4 Matematikk som kontekst

I delanalyse 3 kom det frem at mange av programmeringsoppgavene inneholder matematikk som burde være kjent for elevene. Det innebærer at oppgavene ikke legger opp til at elevene skal få utforsket nye temaer, men heller at de bruker matematikk som kontekst for å lære programmering. Som vi ser i figur 19 kommer de fleste programmeringsoppgavene, der matematikk kun er kontekst for oppgaven, fra Aschehoug. Flesteparten av oppgavene, der matematikk er kontekst for oppgaven finnes i oppgaver med det matematiske begrepet «aritmetikk». I denne kategorien var det mange oppgaver knyttet til enkel aritmetikk, blant annet de fire regneartene, eller «enkle» figurtall-oppgaver, som burde være kjent for elevene. En annen observasjon som ble gjort var at mange av oppgavene med matematikk som kontekst for oppgaven brukte programmering uhensiktsmessig. Det vil si at oppgaven handlet om matematikk, men at oppgaven kunne blitt bedre løst ved hjelp av Excel, Geogebra eller penn og papir. Ifølge Munthe (2023, s. 9) mister man dermed både gleden og nytten av å bruke verktøyet. Blant annet «Tenk på et tall»-oppgaven i figur 12 kunne like så godt blitt løst i Geogebra eller for hånd. Gjennom analysen observerte jeg at de oppgavene jeg fant som uhensiktsmessige og der matematikk var kontekst for oppgaven, manglet programmeringsbegrepene «løkker» og «vilkår». Koden som ble skrevet ville dermed, av maskinen, bli lest linje for linje nedover og man fikk ikke utnyttet styrken i de to programmeringsbegrepene. Styrken til løkker er at de kan gjøre at deler av koden kjører flere ganger. Styrken til vilkår er at de har evnen til å ta valg basert på visse betingelser samtidig som de støtter at programmet kan ha flere utfall (Benton et al., 2016, s. 4-5).

Selv om disse oppgavene er vurdert av meg til å ha en svak kobling til matematikk, så kan de være nyttige for noen elever. Matematikken er jo ikke fraværende, den skulle bare vært kjent. Elever har ulik matematisk kompetanse og det vil være elever som ikke har forstått matematiske begreper fra tidligere år, samtidig som noen sannsynligvis har glemt noen temaer. Dermed kan nok noen av de oppgavene der matematikk kun er kontekst for oppgaven ha virkning som utforskende oppgaver for visse elever. Samtidig vil programmering kunne gi elevene en annen type innfallsvinkel som kan gjøre at elevene forstår noen matematiske begreper de kanskje ikke ville gjort uten å se en løsning ved hjelp av programmering. Spesielt oppgaver som tidligere har inneholdt en «black box». Oppgaver med matematikk som kontekst kan også være nyttige i en opplæringsfase av programmering. Da kan elevene prøve å forstå hvordan et program fungerer ved å se på den matematiske konteksten. Allikevel vil jeg fremheve at det er bedre å ha rike oppgaver enn noen oppgaver der matematikk er kontekst for oppgaven, ettersom rike oppgaver vil treffe både elever som har lav og høy kompetanse i temaet (Wæge & Nosrati, 2018, s. 83). Rike oppgaver innebærer oppgaver som har lav inngangsterskel og stor takhøyde. Det vil si at oppgavene skal gi alle elevene mulighet til å begynne å arbeide, samtidig som at oppgavene skal gi elevene mulighet til å jobbe etter sitt eget nivå. Rike oppgaver gir også elevene muligheter for å jobbe med utfordrende matematikk, der man kan bruke ulike løsningsmetoder. Et eksempel på oppgaver som kan fungere som rike oppgaver er oppgaver som følger UMC-rammeverket (Maugesten et al., 2021, s. 6). I neste delkapittel blir det diskutert en måte programmering og matematikk kan knyttes godt sammen, nemlig hvis elevene får mulighet til å

omformulere programmeringsbegreper til matematisk notasjon.

### 5.2.5 Omformulere programmeringsideer til matematisk notasjon

I delanalyse 3 kom det frem at 6 oppgaver ga elevene mulighet til å kunne omformulere programmeringsideer til matematisk notasjon. Ofte er disse oppgavene knyttet til handlingen «forklare». Ettersom oppgavene er knyttet til handlingen «forklare», er det naturlig at flesteparten av oppgavene kommer fra Aschehoug (figur 19). Det kan tenkes at det er få oppgaver der elevene kan omformulere programmeringsideer til matematikk notasjon fordi slike oppgaver ofte er fasilitert av læreren. I tillegg blir det gjerne gjort muntlig i grupper med eksempler eller oppgaver funnet i bøkene eller andre steder. Det kan tyde på at Oldervoll et al. (2020) har tenkt på dette. Sinus har ingen «forklare» oppgaver, men de har diskusjonsspørsmål i form av snakkebobler. Diskusjonsspørsmålene hører ikke til en oppgavedel i boka, men står sammen med forklaringer og eksempler, og er dermed ikke tatt med som analyseenheter. Sinus har dermed lagt opp til at elevene kan omformulere programmeringsideer til matematisk notasjon som en aktivitet som skal gjøres i grupper, og det kan antas at det er noe læreren må fasilitere ettersom diskusjonsspørsmålene ikke er lagt til som spesifikke oppgaver.

Oppgaver der elevene må omformulere programmeringsideer til matematisk notasjon kan bedre elevenes problemløsningsferdigheter (Sentance et al., 2018, s. 123). Samtidig kan erfaring i å omformulere programmeringsideer til matematisk notasjon også hjelpe elever med å knytte sammen sentrale ideer (Toh & Yeo, 2019, s. 8). Elevene lærer av å koble ny informasjon med deres tidligere kunnskaper. Hvis elevene for eksempel klarer å koble løkker i en figurmønster-oppgave til den rekursive formelen for figurtallet, så vil de forhåpentligvis klare å se den sterke koblingen mellom matematikk og programmering. Har de sett denne sammenhengen så har de antakelig forstått en sentral ide i matematikken, nemlig den knyttet til mønster (Charles & Carmel, 2005, s. 17). Sentrale ideer underbygger matematikkundervisning i skolen og de fleste sentrale ideer er knyttet sammen med mange andre sentrale ideer (Charles & Carmel, 2005, s. 10). Dermed kan forståelsen av en sentral ide hjelpe elever med å utvikle en dyp forståelse av matematikk. Til tross for de positive aspektene ved å omformulere programmeringsideer til matematisk notasjon, observeres det sjelden oppgaver som legger opp til dette. I det neste delkapittelet skal jeg ta for meg den andre måten programmering og matematikk kan knyttes godt sammen på, nemlig hvis elevene får bruke programmering til å utforske nye matematiske ideer.

### 5.2.6 Utforske nye matematiske ideer

Som nevnt i delanalyse 3, finnes det en god del oppgaver som lar elevene utforske nye matematiske ideer. Dette er positivt både fordi det knytter programmering og matematikk sammen på en hensiktsmessig måte (Bråting & Kilhamn, 2022), men også fordi det passer godt med satsingsområdet i LK20. Ifølge Flø (2021, s. 3) står det beskrevet i både overordnet del, i kjerneelementene og i mange kompetansemål at elevene skal utforske i matematikken.

I dette delkapittelet vil jeg presentere ulike tilnærminger til utforskende oppgaver.

Spesielt nyttig er de oppgavene som handler om «tilnærming». Det som kan være negativt er at en stor andel av disse oppgavene er knyttet til handlingen «følge en regel» og programmeringsbegrepet «stegvise instruksjoner». I Sinus er alle oppgavene om tilnærming knyttet til «følge en regel» og «stegvise instruksjoner». Grunnen til at jeg har karakterisert oppgavene under den handlingen og det programmeringsbegrepet er fordi boka har gitt gode eksempler til hvordan man kan finne nullpunktene til en likning, eller tilnærmingsverdien til kvadratroten av et positivt tall, ved hjelp av programmering. Dette kan føre til at elevene ikke får utforske tilnærmingsmetoder, og i dette tilfellet Newton-Raphson-metoden, selv om oppgaven er karakterisert som en utforskende oppgave. I en av de seks oppgavene om «tilnærming» har oppgaven handlingen «forme og skape» i tillegg til «følge en regel». Altså må elevene også skrive kode, som de ikke finner i et eksempel. Det betyr at elevene i fem av de seks tilnærmingsoppgavene kun trenger å vise at de mestrer programmeringsbegrepene «stegvise instruksjoner» og «kode». Allikevel vil oppgavene med handlingen «følge en regel» kunne fungere som «forme og skape». Hvis elevene ikke studerer eksemplene som er gitt får de muligheten til å skrive programmet selv. Dersom de bruker eksempelet kun til veiledning, men ikke skriver direkte av, vil oppgaven også kunne fungere som en utforskende oppgave.

For å sikre at elevene faktisk forstår Newton-Raphson-metoden og hvordan programmet i eksemplene fungerer, kunne det vært satt inn deloppgaver av handlingene «forklare» eller «forutse», slik at elevene må jobbe seg gjennom eksempelet. Bråting og Kilhamn (2022, s. 607) hevder også at å inkludere mer av «forklare», «forutse» og «feilsøke» kan føre til å berike programmeringsinnholdet og styrke koblingen mellom programmering og matematikk. I Mønster er også oppgavene knyttet til «tilnærming», kategorisert som «følge en regel» eller «forme og skape», men de har totalt sett flere oppgaver enn Sinus som krever numerisk løsning. De har 14 oppgaver som er knyttet til numerisk løsning av likninger og har i tillegg 6 oppgaver som handler om numerisk derivasjon. For å kunne presentere to ulike tilnærminger til numerisk løsning tar jeg utgangspunkt i oppgavene med numerisk løsning av likninger for resten av diskusjonen om utforskende oppgaver. Av de 14 oppgavene om numerisk løsning av likninger, inneholder 8 oppgaver kun handlingen «følge en regel». De 14 oppgavene i Mønster bærer også mye preg av spørsmål av typen «Lag et program […]», der de kan løse oppgavene ved å skrive av eksempelet og gjøre små endringer. Mønster har derimot flere oppgaver som handler om å utvide programmet som er gitt i et eksempel. Oppgavene legger opp til at elevene skal bruke programmering for å få mer informasjon om programmet eller likningene, for eksempel hvor mange halveringer programmet har gjort. Her kunne også koblingen mellom programmering og matematikk blitt styrket dersom elevene hadde blitt spurt om å forklare mer av programmet og hvorfor det er nyttig å legge inn de kodelinjene de blir bedt om å skrive. I Aschehoug finnes ingen programmeringsoppgaver som handler om tilnærming av nullpunkter. Det representerer hvor store forskjeller det er mellom lærebøkene. Som nevnt i 5.2.2 vil dette mest sannsynlig føre til at elevene som bruker Aschehoug ikke lærer dette temaet (i matematikk 1T).

Som nevnt i kapittel 4.2.3, var det flere oppgaver om «tilnærming» som var veldig like. For eksempel har Mønster 14 oppgaver knyttet til numerisk løsning av likninger. Flesteparten av de 14 oppgavene har jeg kategorisert som utforskende oppgaver, men mange av oppgavene er nesten helt like. Man kan da tenke seg at ikke alle oppgavene faktisk vil kunne fungere som utforskende oppgaver, nettopp fordi elevene allerede har gjort en lignende oppgave. Selv om en bok har mange oppgaver på et tema, betyr ikke det nødvendigvis at boka blir bedre. Som vist i figur 19 har Mønster en del oppgaver som

er utforskende, men det kan diskuteres om oppgavene faktisk gir mulighet for utforsking når oppgavene kan løses på tilsvarende måte hver gang. I de to siste delkapitlene vil jeg gjøre en vurdering av rammeverket til Bråting og Kilhamn (2022) og presentere studiens begrensninger.

## 5.3 Vurdering av rammeverket

Som nevnt i kapittel 4.1 har jeg videreutviklet rammeverket til Bråting og Kilhamn (2022). I utgangspunktet er det en del forskjeller på Bråting og Kilhamn (2022) sine analyseenheter og analyseenhetene for denne studien. Bråting og Kilhamn (2022) har sett på barneskolen der det er andre kompetansemål enn på videregående. Deres oppgaver baserer seg på BBP imens mine analyseenheter bruker TBP. I tillegg er programmering i svenske læreplaner i matematikk knyttet til algebra (Bråting & Kilhamn, 2022, s. 595), mens programmering i norske læreplaner ikke er knyttet til et spesifikt tema i matematikken. Til tross for forskjellene har rammeverket fungert til å svare på deler av forskningsspørsmålene, som forventet. Jeg har ikke endret rammeverket når det kom til handlinger, da ulike typer oppgaveoppbygging synes å være lik uavhengig av klassetrinn. Handlingene er ganske generelle, og er dermed relevante på et høyere nivå. Det handler nok om at handlingene kan knyttes opp mot AT (tabell 2) og at AT er en ferdighet som ikke er knyttet til et spesifikt klassetrinn (Brennan & Resnick, 2012, s. 1). Det som skiller mitt rammeverk fra Bråting og Kilhamn (2022) sitt, med tanke på handlinger, var at «følge en regel» fikk en ensidig betydning. I oppgavene Bråting og Kilhamn (2022, s. 601) analyserte fant de flere oppgave der elevene måtte følge trinnvise instrukser. Disse oppgavene ble karakterisert som «følge en regel», men ingen slike oppgaver fantes i mitt datasett. De oppgavene jeg karakteriserte som «følge en regel» var de oppgavene der elevene kunne bruke eksempler som en instruks for å løse oppgaven.

Det som er interessant er at det ikke var store forskjeller mellom resultatene av handlinger i mine resultater og Bråting og Kilhamn (2022) sine resultater. Bråting og Kilhamn (2022, s. 601-602) fant at de to vanligste handlingene var «følge en regel» og «forme og skape». De fant betydelig færre oppgaver med andre handlinger og helt spesifikt fant de minst av oppgaver (totalt for 1.-6. trinn) der elevene måtte «forestille seg». I tillegg er det flere trekk som er fundamentale for å lære programmering som de fant lite av i oppgavene. En av dem var feilsøking. Som vist i resultater (kapittel 4) fant jeg også flest av handlingene «forme og skape» og «følge en regel» og mindre av de andre handlingene. Spesielt kan man legge merke til at det er lite feilsøking i begge sine resultater, til tross for viktigheten av feilsøking når man lærer programmering (Sentance et al., 2018, s. 29).

En forskjell som kommer til syne når man sammenligner programmeringsbegreper, er at mitt datasett ikke inneholder noen oppgaver med analog programmering. Sammenligner man med resultatene til Bråting og Kilhamn (2022), finner man at de har mange oppgaver med analog programmering. Et eksempel er en oppgaver der de skal flytte en robot i et rutenett ved å følge trinnvise instrukser. At Bråting og Kilhamn (2022) finner mange oppgaver med analog programmering er nok knyttet til at de ser på barneskolen. For oppgaver på et videregående nivå kunne det vært interessant å analysere flere programmeringsbegreper som skal være kjent ifølge læreplanen. To programmeringsbegreper som finnes i læreplanen for matematikk 1.-10., men ikke i analyseverktøyet til Bråting og Kilhamn (2022), er «variabler» og «funksjoner»

(Kunnskapsdepartementet, 2019). Ifølge Sande (u.å, s. 3) er «variabler» og «funksjoner» sentrale for programmering. Grunnen til at jeg ikke la til begrepene som koder var for å holde meg så tett opp til rammeverket som mulig.

En utfordring med å bruke rammeverket var at oppgavene i matematikk 1T ofte ikke ba om så spesifikke ting som oppgavene Bråting og Kilhamn (2022) undersøkte. Spesielt med tanke på programmeringsbegreper. I Bråting og Kilhamn (2022) sin undersøkelse handlet oppgavene ofte om kun en av begrepene, med oppgaver som «Knytt armbåndet med rett algoritme» og «Følg instruksene». I oppgavene jeg analyserte var det mange oppgaver av typen «Lag et program som […]». Som nevnt i kapittel 4.1.2 krevde det at jeg måtte gå mer i dybden i hver oppgave for å se hva oppgaven faktisk handlet om. Det førte også til at det ble unyttig å se etter programmeringsbegreper eksplisitt nevnt i oppgavene. At oppgavene ofte handlet om kun et begrep kan nok knyttes til at programmeringsbegreper blir spesifikt nevnt på lavere trinn. Det gjelder både for Sverige (Bråting & Kilhamn, 2022, s. 606) og for Norge (Kunnskapsdepartementet, 2019). Kompetansemålet i matematikk 1T handler om å kunne anvende programmering for å løse problemer, og da antas det at elevene har kontroll på programmeringsbegrepene fra tidligere kompetansemål (Kunnskapsdepartementet, 2020).

For å oppsummere synes jeg rammeverket fungerte som et godt utgangspunkt. Spesielt handlingene fungerer godt som koder også på videregående ettersom AT ikke er knyttet til et spesifikt klassetrinn (Brennan & Resnick, 2012, s. 1). Det fører dermed til at resultatene gir informasjon som bidrar til å svare på forskningsspørsmålene mine. Allikevel anbefaler jeg å endre programmeringsbegreper og matematiske begreper slik at de passer til kompetansemålene til det trinnet man undersøker.

## 5.4 Studiens begrensninger

I denne studien har jeg undersøkt de to forskningsspørsmålene; (1) Hva karakteriserer programmeringsoppgaver i lærebøker i matematikk 1T og hvordan knyttes oppgavene til PRIMM og UMC? (2) Hvilke temaer knyttes programmeringsoppgavene i matematikk 1T til og på hvilke måter knyttes programmering og matematikk sammen? Undersøkelsen har blitt gjort ved å analysere oppgavene i de tre lærebøkene; Aschehoug, Sinus og Mønster. Min intensjon har hele tiden vært å undersøke forskningsspørsmålene, ikke sammenligne eller bedømme kvaliteten på læreverkene. Likevel har jeg vurdert oppgavene som lærebokforfatterne har valgt å bruke. På denne måten kan man si at kvaliteten av lærebøkene er vurdert indirekte, da jeg har påpekt både hvilke handlinger og sammensetninger av handlinger som kan være nyttige, om temaene programmering er knyttet til er hensiktsmessige og hvor hensiktsmessig kobling det er mellom programmering og matematikk i oppgavene. Å ikke indirekte vurdere kvaliteten vil være vanskelig i en analyse av lærebøker. I tillegg har jeg presentert handlinger, begreper og hvordan programmering og matematikk knyttes sammen i hver lærebok for seg. Det er som nevnt ikke for å sammenligne bøkene, men for å kunne se ulike tilnærminger til hvordan programmering har blitt, og kan bli, inkludert i matematikk 1T.

Analyseverktøyet har bidratt til kodingen av oppgavene, men jeg har gjort justeringer på kategoriene for at analyseverktøyet skal passe til matematikk 1T. En svakhet ved innholdsanalyser er at andre som gjennomfører innholdsanalyser med samme

analyseverktøy, vil kunne tolke oppgavene annerledes (Grønmo, 2004, s. 220). Likevel har jeg forsøkt å gjøre det jeg kan for å gi et objektivt inntrykk av prosessen og gitt forklaringer på hvordan jeg har analysert. I tillegg vil man kunne kode oppgavene ulikt, hvis man gjennomfører analysen på nytt. For å sikre oppgavens reliabilitet har jeg forsøkt å ta med tydelige forklaringer i analysen på hvordan jeg karakteriserer ulike oppgaver i tillegg til å kode oppgavene flere ganger. Selv om jeg analyserte oppgavene flere ganger kan det være en svakhet at det kun var en person som analyserte oppgavene, da man mister muligheten til å få flere ulike synsvinkler og diskutere oppgaver. Samtidig diskuterte jeg flere oppgaver der jeg var usikker på kodingen med både veileder og medstudenter.

En annen begrensning i studien er at jeg kun har sett på tre lærebøkers fremstillinger. Det finnes flere lærebøker i matematikk 1T som kan ha inkludert programmering på andre måter enn de tre bøkene jeg har sett på. Det finnes også flere nettressurser til lærebøkene jeg har analysert, som kan ha et annet fokus på programmering enn de fysiske bøkene. I tillegg har jeg ikke tatt hensyn til eksempler (med mindre de var tydelig knyttet til oppgavene), forklaringer eller læringsaktiviteter knyttet til temaet. Det er også verdt å merke seg at studien tar for seg lærebøkenes fremstilling og hvordan denne fremstillingen kan påvirke elevene. Lærerveiledninger kan gi instrukser og variasjoner for oppgavene samtidig som at lærere også kan endre oppgavene.

# 6. Konklusjon

I denne masteroppgaven har målet vært å svare på forskningsspørsmålene:

(1) Hva karakteriserer programmeringsoppgaver i lærebøker i matematikk 1T og hvordan knyttes oppgavene til PRIMM og UMC?

(2) Hvilke temaer knyttes programmeringsoppgavene i matematikk 1T til og på hvilke måter knyttes programmering og matematikk sammen?

For å svare på forskningsspørsmålene har jeg gjennomført en innholdsanalyse av programmeringsoppgavene i tre lærebøker for matematikk 1T. Jeg har tatt utgangspunkt i analyseverktøyet fra Bråting og Kilhamn (2022) og gjort noen endringer der jeg fant det hensiktsmessig for det jeg ville undersøke. Jeg har blant annet brukt en annen modell for AT som finnes i Gjøvik og Torkildsen (2019). I tillegg har jeg brukt sentrale ideer for å analysere hvilke tema matematikk knyttes til. Gjennom analysen har jeg undersøkt hvilke handlinger elevene blir bedt om å gjøre i oppgavene, hvilke programmeringsbegreper og matematiske begreper oppgavene kan knyttes til, hvilke kapitler og tema oppgavene hører til og hvordan matematikk og programmering knyttes sammen. Videre har jeg diskutert resultatene og sammenhengen mellom handlingene og metodene PRIMM og UMC. Det er viktig å nevne at jeg har undersøkt oppgavene i boka og ikke noe om hva lærere gjør i timen eller hva som står i eventuelle lærerveiledninger. Dermed kan jeg ikke si noe om blant annet samarbeid eller helklassediskusjoner som har en naturlig plass i mange undervisningssituasjoner.

## 6.1 Forskningsspørsmål 1

Funn fra diskusjonen (5.1) viser at de mest typiske programmeringsoppgavene i matematikk 1T handler om at elevene skal skrive eller utvide kode (forme og skape), etterfulgt av å endre kode gitt i eksempler (følge en regel). Det observeres sjeldent oppgaver som innebærer at elevene må feilsøke, forestille seg, finne regler eller forklare kode. «Feilsøke» er den handlingen og det programmeringsbegrepet som forekommer minst i oppgavene. Kun to oppgaver la opp til at elevene kom til å få en feilmelding og at de deretter måtte endre programmet. Dette til tross for at feilsøking anses som svært nyttig i programmering og for utvikling av AT (Utdanningsdirektoratet, 2019a). Ettersom syntaks ofte blir sett på som det vanskeligste når man skal lære TBP (Moors et al., 2018, s. 58) kan det være negativt at bøkene ikke inneholder flere oppgaver der elevene kan lære å lese og forstå feilmeldinger. Jeg fant også ut at det var et ensidig fokus av handlinger. Som oftest ber oppgavene kun om en av de seks handlingene. Under analysen av programmeringsbegreper kom det fram at alle oppgavene er knyttet til programkode. Altså er det ingen av lærebokforfatterne som anser analog programmering som hensiktsmessige oppgaver i matematikk 1T. I tillegg ber oppgavene sjeldent om bruk av spesifikke programmeringsbegreper. Mange oppgaver var av typen «Lag et program […]» og la dermed opp til at elevene selv måtte finne ut hvilke programmeringsbegreper de måtte bruke for å løse oppgaven.

Som nevnt inneholder oppgavene typisk kun én handling. Det finnes dermed få oppgaver som følger arbeidsmetodene PRIMM og UMC, som er arbeidsmetoder der elevene bruker en og en handling for å gradvis gjøre et program sitt eget. I følge Sentance et al. (2018,

s. 114-115) vil arbeidsmetodene PRIMM og UMC kunne føre til at elevene utvikler sin AT. Dermed vil det være nyttig at lærere er bevisst på ulike handlinger og kombinasjoner av handlinger, slik at de eventuelt kan justere oppgavene i en undervisningssammenheng. Samtidig er PRIMM og UMC metoder laget hovedsakelig for å hjelpe elever med å lære programmering. Ettersom kompetansemålet innebærer å «Formulere og løse problemer ved hjelp av […] programmering» (Kunnskapsdepartementet, 2020), tyder det på at elevene skal lære å lage program fra bunn av. Det kan være en grunn til at det finnes mange oppgaver som er av typen «Lag et program […]» som er kategorisert som «forme og skape». Uansett er programmering såpass nytt i skolen, så mange elever i matematikk 1T har hatt lite om programmering tidligere, og mange har nok dermed behov for stillasbygging. Da vil oppgaver med flere handlinger være nyttige. Lærere må altså ta hensyn til elever som er på ulikt faglig nivå og der kan nok oppgaver med flere handlinger være nyttig. I fremtiden vil nok flere elever klare oppgaver der de må skrive kode fra bunn av, men det vil nok uansett være noen som sliter med programmering (sånn som alle andre temaer i matematikken). For lærere med elever som sliter med programmering, vil jeg dermed anbefale å endre oppgavene fra lærebøkene slik at oppgavene følger PRIMM eller UMC metodikken.

Vi kan også se forskjell mellom lærebøkene når det gjelder hva som karakteriserer programmeringsoppgavene. Sinus har mange «forme og skape» oppgaver som vil kunne være fine for elever som trenger mer utfordring (Maugesten et al., 2021, s. 7). De gjenværende programmeringsoppgavene er «følge en regel»-oppgaver. Disse oppgavene kan fungere som «forme og skape»-oppgaver, og være utfordrende, så lenge elevene ikke gransker eksempelet gitt tidligere. Aschehoug har oppgaver med flere deloppgaver som følger PRIMM-metodikken og har oppgaver innenfor alle handlingene. Det vil passe godt for elever som ikke er så trygge på programmering ennå. Mønster har også en del oppgaver med «forme og skape» og «følge en regel» som kan være nyttig for elever som trenger mer utfordring.

## 6.2 Forskningsspørsmål 2

Funn fra delanalyse 2 og diskusjonen (5.2) viser at programmering typisk knyttes til de matematiske begrepene; «likning», «tilnærming», «funksjon» og «aritmetikk». Oppgaver om likninger, tilnærminger og funksjoner kan som oftest knyttes til numeriske metoder, inkludert numerisk løsning av likninger eller numerisk derivasjon. Ifølge Charles og Carmel (2005, s. 17) kan numeriske metoder knyttes til en sentral ide i matematikken. Den sentrale ideen er:

> «Numeriske beregninger kan tilnærmes ved å erstatte tall med andre tall som er nærme i verdi og enkle å regne med mentalt. Målinger kan tilnærmes ved å bruke kjente referenter som enheten i måleprosessen» (Charles & Carmel, 2005, s. 17).

Å forstå hvordan numeriske metoder kan brukes til å estimere tall kan bidra til at elevene får en sterkere forståelse for matematikken og spesielt hvordan programmering kan brukes på en fornuftig måte. Fra resultatene kom det frem at flere oppgaver knyttes til kapitler kalt algebra. Hvis programmering skal knyttes til algebra er det viktig at lærere er bevisste på at programmering som verktøy ikke kan løse algebraiske uttrykk på samme måte som vi gjør i matematikken (Kohn, 2017, s. 349). Python kan blant annet ikke holde ukjente verdier i minnet, sånn som for eksempel CAS kan. Programmering

knyttes også til mønster og figurtall (som kan knyttes til algebra), der programmering vil være en nyttig metode for å regne ut et høyt figurnummer rekursivt dersom det er vanskelig å finne den eksplisitte formelen.

Funn fra delanalyse 3 tyder på at programmeringsoppgavene ofte legger opp til at elevene får utforske nye ideer. Mange av oppgavene der elevene får utforske nye ideer er knyttet til oppgaver med numeriske metoder. Numeriske metoder er en ny gren av matematikk i skolen som ikke fikk innpass i læreplanene før LK20, men som nå ved hjelp av programmering har blitt mye mer aktuell (Sanne et al., 2016, s. 61). Mange av programmeringsoppgavene bærer også preg av at matematikk kun er kontekst for oppgaven. Det kan tyde på at målet i flere oppgaver er at elevene skal lære programmering, og ikke ny matematikk, ved å bruke matematikk som allerede burde være kjent for elevene. Det er få oppgaver der elevene får mulighet til å omformulere programmeringsideer til matematisk notasjon eller oppgaver som er uten matematisk innhold. Ifølge Bråting og Kilhamn (2022, s. 604) kan oppgaver som har liten kobling mellom programmering og matematikk få en større kobling dersom man legger til deloppgaver med handlingene «forklare» eller «forutse». Det medfører at mine funn kan brukes til å anbefale et arbeidsoppdrag for lærere. For at elevene skal få et godt utbytte av programmering i matematikk burde lærere legge til flere handlinger i oppgavene. Det kan føre til at den opplevde vanskelighetsgraden av opplæring av programmering blir mindre. At mine funn anbefaler et arbeidsoppdrag for lærere medfører at denne masteroppgaven får en god profesjonsrelevans. Jeg vil diskutere mer om profesjonsrelevansen av masteroppgaven i neste avsnitt.

## 6.3 Profesjonsrelevans

Masteroppgaven har god profesjonsrelevans på flere måter. For meg selv og for andre som skal undervise i matematikk 1T vil det være nyttig å vite mer om programmeringsoppgavene i lærebøkene, og hvordan matematikk og programmering i oppgaven knyttes sammen. Spesielt ettersom lærebøkene blir betydelig mer brukt i matematikk, enn i andre skolefag (Lepik et al., 2015, s. 130). Uavhengig om man kommer til å bruke lærebøkene eller ikke har jeg fremmet hvilke handlinger, og kombinasjoner av handlinger, som kan være nyttige i opplæringen av programmering og for utvikling av AT. Handlingene er heller ikke spesifikke for matematikk 1T, men kan brukes i oppgaver på både barneskolen, ungdomsskolen og videregående skole. Fra eksempler og forklaringer gitt i oppgaven kan forhåpentligvis leseren forstå hvordan man kan analysere programmeringsoppgaver ved hjelp av mitt analyseverktøy. Jeg tror også oppgaven kan bidra til refleksjoner rundt nyttigheten av programmering i matematikk, og refleksjoner rundt hvilke temaer programmering burde knyttes til for at det skal bli en hensiktsmessig kobling mellom programmering og matematikk.

## 6.4 Kommentarer til veien videre

Som en forlengelse av denne studien, kunne det vært interessant å undersøke hvordan lærere bruker programmeringsoppgavene i lærebøkene i sin undervisning. Programmering er tatt med som kompetansemål fra 2.trinn, og læreplanen (LK20) trådte i kraft i 2020. Dermed vil det fortsatt ta noen år før elevene som begynte med programmering i 2.trinn skal begynne på matematikk 1T. Det kan medføre at elevene har

et dårligere grunnlag for å oppnå kompetansemål i høyere trinn. Derfor kunne det vært spennende å se om lærere gjør noen forandringer på oppgavene i lærebøkene og hva slags endringer de gjør for å treffe elevenes faglige nivå.

# 7. Litteraturliste


Benton, L., Hoyles, C., Kalas, I. & Noss, R. (2016). *Building mathematical knowledge with programming: insights from the ScratchMaths project*. Constructionism 2016, Bangkok, Thailand.

Benton, L., Hoyles, C., Kalas, I. & Noss, R. (2017). Bridging Primary Programming and Mathematics: Some Findings of Design Research in England. *Digital Experiences in Mathematics Education*, *3*(2), 115-138. https://doi.org/10.1007/s40751-017-0028-x

Black box. (u.å.). I *Cambridge dictionary*. Hentet 8 mai 2024 fra https://dictionary.cambridge.org/dictionary/english/black-box

Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., Earp, J., Horvath, M., Jasutė, E. & Malagoli, C. (2022). *Reviewing computational thinking in compulsory education: State of play and practices from computing education* (JRC128347). Publications Office of the European Union. https://publications.jrc.ec.europa.eu/repository/handle/JRC128347

Borgan, Ø., Borge, I. C., Engseth, J., Heir, O., Moe, H., Norderhaug, T. T. & Vie, S. M. (2021). *Matematikk R1* (4. utg.). Aschehoug.

Borge, I. C., Engeseth, J., Haug, H., Heir, O., Moe, H., Norderhaug, T. T. & Vie, S. M. (2020). *Matematikk 1T* (4. utg.). Aschehoug.

Brennan, K. & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. American Educational Research Association meeting, Vancouver, canada. https://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Bråting, K. & Kilhamn, C. (2022). The Integration of Programming in Swedish School Mathematics: Investigating Elementary Mathematics Textbooks. *Scandinavian Journal of Educational Research*, *66*(4), 594-609. https://doi.org/10.1080/00313831.2021.1897879

Büyükkarci, A. & Taşlidere, E. (2023). The effect of 5e learning model enriched with coding on primary school mathematics lesson. *Education and Information Technologies*, *29*, 7969-7995. https://doi.org/10.1007/s10639-023-12129-1

Charles, R. I. & Carmel, C. (2005). Big ideas and understandings as the foundation for elementary and middle school mathematics. *Journal of Mathematics Education Leadership 7*(3), 9-24. https://jaymctighe.com/wp-content/uploads/2011/04/MATH-Big-Ideas_NCSM_Spr05v73p9-24.pdf

Csunplugged. (u.å.). *Computer Science without a computer*. Hentet 29 mai fra https://www.csunplugged.org/en/

De nasjonale forskningsetiske komiteene. (2021). *Forskningsetiske retningslinjer for samfunnsvitenskap og humaniora*. Hentet 19 mai fra https://forskningsetikk.no/retningslinjer/hum-sam/forskningsetiske-retningslinjer-for-samfunnsvitenskap-og-humaniora/

Dolonen, J. A., Kluge, A., Litherland, K. & Mørch, A. I. (2019). *Litteraturgjennomgang av programmering i skolen*. Universitetet i Oslo. http://urn.nb.no/URN:NBN:no-79405

Engen, H. (2022). *Programmeringsinnhold i lærebøker* [Masteroppgave, Norges teknisk-naturvitenskapelige universitet]. NTNU Open. https://hdl.handle.net/11250/3041706

Fløⱪ, E. E. (2021). Programmering i LK20. *Tangenten–tidsskrift for matematikkundervisning*, *32*(1), 3-9.

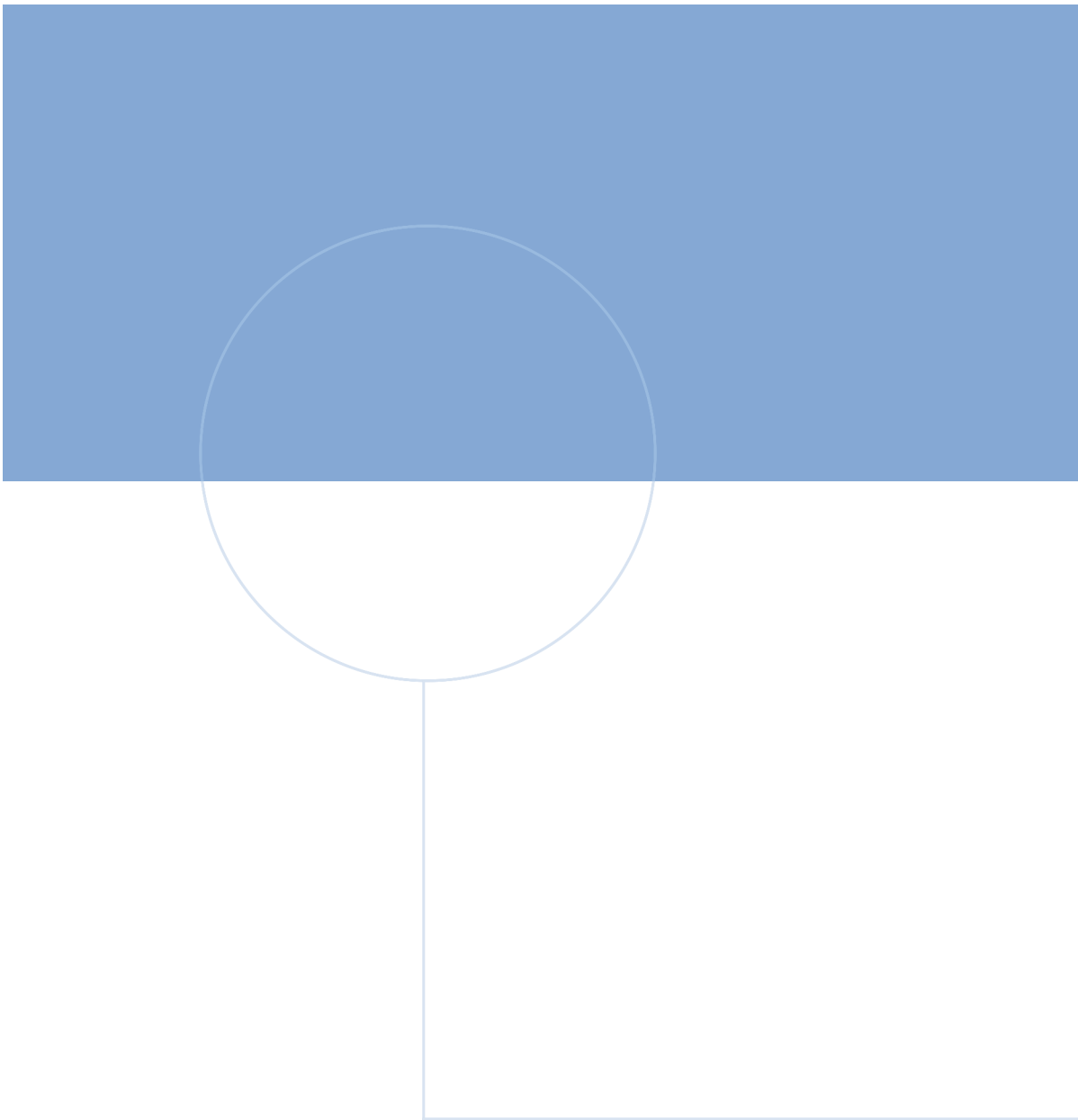
FN. (u.å.). *God utdanning*. Hentet 16 mai fra https://fn.no/om-fn/fns-baerekraftsmaal/god-utdanning#Hvagj%C3%B8rNorge?-1

Garner, L. E. (1981). On the Collatz 3n+1 algorithm. *Proceedings of the American Mathematical Society*, *82*(1), 19-22. https://doi.org/https://doi.org/10.2307/2044308

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

Grønmo, S. (2004). *Samfunnsvitenskapelige metoder*. Fagbokforlaget.
Hofmann, A. (2007). aritmetikk. I *Store norske leksikon*. Hentet 29 april 2024 fra https://snl.no/aritmetikk
Kalvø, T., Opdahl, J. C. L., Skrindo, K. & Weider, Ø. J. (2020). *Mønster 1T*. Gyldendal Norsk forlag AS.
Kirke- og undervisningsdepartementet. (1991). *Mønsterplan for grunnskolen M87.* (3. utg.). Aschehoug & Co.
Kohn, T. (2017). *Variable Evaluation: an Exploration of Novice Programmers' Understanding and Common Misconceptions*. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, Seattle, Washington, USA. https://doi.org/10.1145/3017680.3017724
Kongelf, T. R. (2015). Introduksjon av algebra i matematikkbøker for ungdomstrinnet i Norge. *Nordic Studies in Mathematics Education*, *20*(3-4), 83–109.
KUF. (1996). *Læreplanverket for den 10-årige grunnskolen (L97)*. Det kongelige kirke-, utdannings- og forskningsdepartement (KUF).
Kunnskapsdepartementet. (2006). *Læreplan i matematikk (MAT1-01)*. Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2006. https://www.udir.no/kl06/MAT1-04
Kunnskapsdepartementet. (2019). *Læreplan i matematikk 1–10 (MAT01-05)*. Fastsatt som forskrift ved kongelig resolusjon. Læreplanverket for Kunnskapsløftet 2020. . https://www.udir.no/lk20/mat01-05
Kunnskapsdepartementet. (2020). *Læreplan i matematikk T(MAT09-01)*. Fastsatt som forskrift ved kongelig resolusjon. Læreplanverket for Kunnskapsløftet 2020. . https://www.udir.no/lk20/mat09-01/kompetansemaal-og-vurdering/kv42?lang=nob
Lepik, M., Grevholm, B. & Viholainen, A. (2015). Using textbooks in the mathematics classroom–the teachers' view. *Nordic Studies in Mathematics Education*, *20*(3-4), 129-156. https://www.researchgate.net/publication/286232223_Using_textbooks_in_the_mathematics_classroom_-_the_teachers'_view
Lindsø, J. F. (2020, 27. februar). *Hva er programmering.* NDLA. https://ndla.no/article/22522
Matematikksenteret. (u.å.-a). *Didaktisk grunnlag*. Hentet 4. mai 2024 fra https://www.matematikksenteret.no/l%C3%A6ringsressurser-og-undervisningsopplegg/programmering/anbefalt-arbeidsmetodikk
Matematikksenteret. (u.å.-b). *Programmering i LK20*. Hentet 29. februar 2024 fra https://www.matematikksenteret.no/l%C3%A6ringsressurser-og-undervisningsopplegg/programmering/programmering-i-lk20
Maugesten, M., Stigberg, H. & Stigberg, S. K. (2021). Programmering på ungdomstrinnet. *Tangenten – tidsskrift for matematikkundervisning*, *32*(3), 2-7.
Moors, L., Luxton-Reilly, A. & Denny, P. (2018). Transitioning from block-based to text-based programming languages. 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE), Auckland, New Zealand. https://doi.org/https://doi.org/10.1109/LaTICE.2018.000-5
Munthe, M. (2023). Programmering og diskusjon. *Tangenten – tidsskrift for matematikkundervisning*, *33*(4), 9-12.
Neraal, A. (2024). Norges 50 største forlag. *bok365*. https://bok365.no/artikkel/norges-50-storste-forlag/
Oldervoll, T., Svorstøl, O., Vestergaard, B., Gustafsson, E., Osnes, E. R., Jacobsen, R. B. & Pedersen, T. A. (2020). *Sinus 1T* (4. utg.). Cappelen Damm.
Papert, S. (1980). *Mindstorms-Children, Computers, and Powerful Ideas*. Basic Books, Inc., Publishers.
Pepin, B., Gueudet, G. & Trouche, L. (2013). Investigating textbooks as crucial interfaces between culture, policy and teacher curricular practice: Two contrasted case studies in France and Norway. *ZDM Mathematics Education*, *45*, 685-698. https://doi.org/https://doi.org/10.1007/s11858-013-0526-2
Robson, C. & McCartan, K. (2015). *Real World Research* (4. utg.). John Wiley & Sons Inc.

Sande, H. (u.å). Programmering - Noen sentrale begrep. Hentet 9. mai 2024 fra https://www.matematikksenteret.no/publikasjoner/programmering-noen-sentrale-begrep
Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., Mørken, K. M., Svorkmo, A.-G. & Voll, L. O. (2016). *Teknologi og programmering for alle*. Utdanningsdirektoratet. https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/teknologi-og-programmering-for-alle/
Sentance, S., Barendsen, E. & Schulte, C. (2018). *Computer Science Education: Perspectives on Teaching and Learning in School*. Bloomsbury Publishing.
Sentance, S., Waite, J. & Kallia, M. (2019). Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education*, *29*(2-3), 136-176. https://doi.org/https://doi.org/10.1080/08993408.2019.1608781
Sevik, K. (2016). *Programmering i skolen* (Notat nr.2). Senter for IKT i utdanningen. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf
Solem, I., Alseth, B., Eriksen, E. & Smestad, B. (2017). *Tall og tanke 2: matematikkundervisning på 5. til 7. trinn*. Gyldendal.
Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., Papert, A. & Silverman, B. (2020). History of logo. *Proceedings of the ACM on Programming Languages*, *4*(79), 1-66. https://doi.org/https://doi.org/10.1145/3386329
Statped. (2021, 1 mars). *Programmering*. Hentet 6. Mars 2024 fra https://www.statped.no/laringsressurser/teknologitema/programmering-for-barn-med-saerskilte-behov/
Statped. (2023, 22. juni). *Programmering for elever med nedsatt syn*. Statped. Hentet 29. mai 2024 fra https://www.statped.no/laringsressurser/syn/temaside-programmering-for-elever-med-nedsatt-syn-temaside/programmering-for-elever-med-nedsatt-syn/blokkprogrammering-og-tekstprogrammering/
Stokkenes, J. H. (2022). *Programmering i norske lærebøker i matematikk* [Masteroppgave, OsloMet-Storbyuniversitetet, OsloMet-Storbyuniversitetet]. ODA. https://hdl.handle.net/11250/3031623
Stoltenberg, C. (2022). Deskriptiv. I *Store norske leksikon*. Hentet 22. februar 2024 fra https://snl.no/deskriptiv
Sun, D., Looi, C.-K., Li, Y., Zhu, C., Zhu, C. & Cheng, M. (2024). Block-based versus text-based programming: a comparison of learners' programming behaviors, computational thinking skills and attitudes toward programming. *Educational technology research and development*, *72*, 1067-1089. https://doi.org/https://doi.org/10.1007/s11423-023-10328-8
Svingen, O. L. & Gilje, Ø. (2018). *Kunnskapsgrunnlag for kvalitetskriterium for læremiddel i matematikk*. Utdanningsdirektoratet. https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/kunnskapsgrunnlag-for-kvalitetskriterium-for-laremiddel-i-matematikk/
Tjora, A. (2018). *Kvalitative forskningsmetoder i praksis* (3. utg.). Gyldendal akademisk.
Toh, T. L. & Yeo, J. B. W. (2019). *Big ideas in mathematics*. World Scientific Publishing Co.
Utdanningsdirektoratet. (2011). R94 (UTGÅTT). https://www.udir.no/laring-og-trivsel/lareplanverket/utgatt/utgatt-lareplanverk-for-vgo-R94/
Utdanningsdirektoratet. (2019a). *Algoritmisk tenking*. Hentet 27. mars 2024 fra https://www.udir.no/kvalitet-og-kompetanse/digitalisering/algoritmisk-tenkning/
Utdanningsdirektoratet. (2019b, 7 November). *Hva er nytt i matematikk?* [Video]. Vimeo. https://vimeo.com/371743660
Utdanningsdirektoratet. (2020). *Læreplan i Matematikk R (MAT03-02)*. Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2020. . https://www.udir.no/lk20/mat03-02
Wing, J. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215
Wæge, K. & Nosrati, M. (2018). *Motivasjon i matematikk*. Universitetsforlaget.



NTNU

Kunnskap for en bedre verden