Masteroppgave

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

Simen Nyutstumoen

Programmering og problemløsning i matematikk

En studie i hvordan programmering kan tilrettelegge for problemløsende aktivitet i matematikk

Masteroppgave i Matematikkdidaktikk 5.–10. trinn
Veileder: Tore Alexander Forbregd
Mai 2021

NTNU
Kunnskap for en bedre verden

Simen Nyutstumoen

Programmering og problemløsning i matematikk

En studie i hvordan programmering kan tilrettelegge for problemløsende aktivitet i matematikk

Masteroppgave i Matematikkdidaktikk 5.–10. trinn
Veileder: Tore Alexander Forbregd
Mai 2021

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

NTNU
Kunnskap for en bedre verden

Sammendrag

Denne masteroppgaven handler om hvordan programmering kan tilrettelegge for problemløsende aktivitet i matematikk. Høsten 2020 innførte Utdanningsdirektoratet en oppdatert lærerplan for matematikk, og med denne har programmering blitt en del av matematikkfaget. Når Utdanningsdirektoratet innfører programmering i matematikk, setter de fokus på dette på følgende måte: *«I læreplanen er algoritmisk tenkning synliggjort fordi dette er en viktig problemløsningsstrategi. Når elevene bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse.»* (Utdanningsdirektoratet, 2020a). For at skolen og lærere skal undervise og benytte programmeringsaktiviteter slik at elevene arbeider problemløsende, trenger vi forskning omkring hvordan en slik undervisning skal foregå.

I denne studien har jeg identifisert en pedagogisk modell, PRIMM, som jeg undersøker i et norsk klasserom med de rammefaktorer og forhold som finnes der. Problemstillingen min er: På hvilken måte kan programmering tilrettelegge for problemløsende aktivitet i matematikk? Dette undersøker jeg ved hjelp av forskningsspørsmålet mitt: Hvilke problemløsningsstrategier kommer til syne når elevene jobber etter PRIMM-modellen?

For å svare på problemstillingen har forskningsdesignet vært utformet som en case-studie, hvor jeg har benyttet meg av observasjon av undervisning for å samle inn data. Casen er fra en barneskole og informantene er elever på 6. trinn på denne skolen. Sosiokulturell læringsteori ligger som et bakteppe for å se hvordan elevene arbeider problemløsende i programmering. Oppgaven presenterer teori om problemløsning, *computational thinking* og PRIMM modellen. Analysene og drøftingene i etterkant av datainnsamlingen, peker på at elevene arbeider med mange problemløsningsstrategier når de programmerer i et undervisningsopplegg designet med PRIMM. Konklusjonene handler om behovet for et pedagogisk verktøy som kan heve programmeringsundervisningen i skolen. Konklusjonene peker også på at PRIMM kan tilrettelegge for problemløsende aktivitet hos elevene som programmerer.

Abstract

This master's thesis is about how programming can facilitate problem-solving activity in mathematics. In the autumn of 2020, the Directorate of Education introduced an updated curriculum for mathematics, and with this, programming has become part of the mathematics subject. When the Norwegian Directorate for Education and Training introduces programming in mathematics, they focus on this in the following way: «In the curriculum, computational thinking is made visible because this is an important problem-solving strategy. When students use programming to explore and solve problems, it can be a great tool for developing mathematical understanding.» (Utdanningsdirektoratet, 2020a). In order for the school and teachers to facilitate and teach programming activities, so that the students work with problem-solving activity, we need research on how such teaching should take place.

In this study, I have identified a pedagogical model, PRIMM, which I examine in a Norwegian classroom with the conditions that exist there. My main research question is: In what way can programming facilitate problem-solving activity in mathematics? I investigate this with the help of a second research question: What problem-solving strategies emerge when students work according to the PRIMM model?

To answer the main research question, the research design has been a case study, where I have used observation of teaching to collect data. The case is from a primary school and the informants are students age 10-11 in Norwegian 6th grade (k5) at this school. Sociocultural learning theory is a backdrop to see how students work problem-solving in programming. The thesis presents theory on problem solving, computational thinking and the PRIMM model. The analyzes and discussions after the data collection indicate that the students work with many problem-solving strategies when they program in a lesson designed with PRIMM. The conclusions are about the need for a pedagogical tool that can raise the level we teach programming in school. The conclusions also indicate that PRIMM can facilitate problem-solving activity in the students who program.

Forord

Å skrive forord til denne masteroppgaven føles rart. Ett et år med helger og kvelder fylt med hardt arbeid, lesing av forskningslitteratur og teori samt oppgaveskriving skal nå avsluttes. Å få muligheten til å fordype seg i et nytt og interessant fagområde har vært spennende, og en rot til mange faglige diskusjoner på jobb. En stor takk til min arbeidsgiver og mine kollegaer som støttet meg gjennom året, og gav meg muligheten til å jobbe med masteroppgaven når jeg trengte det. Alt jeg har lært gjennom dette året tar jeg med meg tilbake inn i skolen, og jeg gleder meg til at elevene mine skal få ny og bedre undervisning i programmering. Jeg gleder meg også til å fortsette givende faglige diskusjoner med kollegaer om hvordan vi skal løse programmeringsundervisningen på skolen i fremtiden. Vi er helt i starten på en ny bølge programmering i skolen, og jeg er glad for å ta del i denne spennende utviklingen. For å sitere Steve Jobs «Everybody in this country should learn to program a computer, because it teaches you how to think».

Å være student igjen har vært begivenhetsrikt og spesielt. Å komme tilbake inn i fagdiskursen er ikke bare lett, så en takk går til min veileder Tore Alexander Forbregd som har hjulpet meg gjennom masterstudien dette året. Jeg har stort sett vært på lesesalen på kveldene og i helgene, og støtten og samholdet blant studentene her har vært til stor hjelp. Spesielt har jeg kost meg når vi har spilt Ligretto og tenkt på noe helt annet enn masteroppgaven. Tusen takk til alle som har gjort lesesaltilværelsen til en positiv og bra opplevelse!

Så til min familie! Tusen takk for uvurderlig hjelp i innspurten av denne masteroppgaven! Til mor og far, uten dere hadde denne studien vært full av skrivefeil og rare setninger. Jeg setter utrolig stor pris på all den tiden dere la ned i lesing og retting. Dere har alltid støttet meg gjennom hele min utdanning, og hjulpet meg dit jeg er i dag, og for det er jeg evig takknemlig.

Sist, men ikke minst takk til min samboer og forlovede Julie! Uten deg hadde det ikke blitt noen masteroppgave. Takk for lange kvelder og netter, for råd og veiledning, for at du støttet og hadde troen på meg, og for at du pushet meg tilbake til studentlivet. Det har vært et langt og slitsomt år, og jeg gleder meg til å sette meg i sofaen og se på tv, gå på tur i helgene og slappe av sammen med deg igjen. Vi skal være stolte over at vi begge har skrevet en master i år!

Trondheim
25.mai.2021
Simen Nyutstumoen

Innhold

Sammendrag v
Abstract vii
Forord ix
Figurer xii
1 Innledning 13
1.1 Programmering i skolen 13
1.2 Problemstilling og forskningsspørsmål 15
1.3 Oppbygging av studien 16
2 Teori 17
2.1 Sosiokulturell læringsteori 17
2.2 Programmering 18
2.2.1 Et historisk perspektiv på programmering i skolen 18
2.2.2 LOGO – starten på programmering i skolen 19
2.2.3 Ludvigsenutvalget: Fremtidens skole 20
2.2.4 Programmering i skolen i dag 20
2.3 PRIMM modellen 22
2.3.1 Grunnlaget til PRIMM 22
2.3.2 PRIMM modellen. 23
2.3.3 En modell for å lære bort programmering. 25
2.3.4 Forskning på og utprøving av PRIMM 26
2.4 Problemløsning i matematikk 27
2.4.1 Problemløsningsprosessen 27
2.4.2 Problemløsningsstrategier 28
2.5 Algoritmisk tenking og Computational thinking 33
2.5.1 Computational thinking 33
2.5.2 Algoritmisk tenking 37
2.6 Scratch 38
3 Metode 41
3.1 Teoretisk perspektiv 41
3.2 Forskningsdesign 42
3.3 Oppgaven 43
3.4 Pilotundersøkelse 44
3.5 Utvalg av elever 44
3.6 Observasjon 45
3.7 Min rolle som lærer og observatør 46

3.8 Video og lyd opptak 47
3.9 Å se etter tegn til problemløsning 47
3.10 Relabiltet og validitet i oppgaven 48
4 Analyse 49
4.1 Predict og run 50
4.2 Investigate 55
4.3 Modify 59
4.4 Make 66
5 Drøfting 71
5.1 Problemløsningsstrategier hos elevene 71
5.1.1 Predict, run, investigate 71
5.1.2 Modify og make 72
5.2 Programmering og matematikk 77
5.3 Studiens begrensninger 77
5.4 Tidliligere forskning på PRIMM 78
5.5 PRIMM i skolen 79
5.6 Metaperspektiv 79
6 Konklusjon 81
6.1 Videre forskning på problemløsing 82
Referanser 83
Vedlegg 86
Vedlegg 1: Undervisningsplan 87
Vedlegg 2: PowerPoint for undervisnings økt 88
Vedlegg 3: Aktivitetsark A 95
Vedlegg 4: Aktivitetsark B – Løkker. 97
Vedlegg 5: Samtykke skjema 98

Figurer

Figur 1: Den proksimale utviklingssonen Imsen (2005, s.259) 17
Figur 2: Programmering= Tenking + kommunikasjon (Utdanningsdirektoratet, 2020b) .18
Figur 3: Use- Modeify- Make. Lee et al., (2011).....23
Figur 4: PRIMM modellen. Scentance et al (2019).....24
Figur 5: Modell for design av en programmeringsøkt (Sentance et al., 2019b) 26
Figur 6:Computational thinkers, et rammeverk for undervisning i *Computational thinking* (Csizmadia et al., 2015, s. 8).....34
Figur 7: Den algoritmiske tenkeren slik Utdanningsdirektoratet bruker den i sine kompetansepakker og hjemmesider.....38
Figur 8: Fargekoding i Scratch (Wikipedia contributors, 2121).....39
Figur 9: Oversatt kode 43
Figur 10: Orginal kode 43
Figur 11: Predict del 1.....51
Figur 12: Kvadrat tegnet fra 1. kode Elevgruppe A 52
Figur 13: Predict del 2.....53
Figur 14:Kvadrat fra 2. Kode, Elevgruppe A 53
Figur 15: Kvadrat fra 1. Kode, før og etter *Run*-steget Elevgruppe A 55
Figur 16: Oppgave: undersøk løkken 58
Figur 17: Halv åttekant 60
Figur 18: Kode for halv åttekant.....60
Figur 19: 3/8 åttekant.....60
Figur 20: 160° 60
Figur 21: Trekant 61
Figur 22: Trekant 2 med kode.....63
Figur 23: femkant, for stor vinkel 64
Figur 24: Fem kant for liten vinkel 64
Figur 25: Femkant og kode 64
Figur 26: Sirkel med kode 65
Figur 27: Firkant og trekant.....66
Figur 28: Firkant og trekant.....67
Figur 29: Sekskant med kode 67
Figur 30: Stor seks kant med kode 68
Figur 31: Hus laget med en kodesekvens 69

1 Innledning

Samfunnet er i stadig endring, og det er viktig for lærere og elever å holde tritt med utviklingen. Hverdagen er stadig mer preget av teknologi, og som en konsekvens har Utdanningsdirektoratet innført programmering som en del av læreplanen i grunnskolen i fagfornyelsen 2020 (NOU 2014:7; Sanne et al., 2016).

Innføringen av programmering førte til en debatt rundt implementeringen i skolen. Den ene siden ledet blant annet av Sanne-utvalget, leverte en rapport i 2016 som konkluderte med at det burde opprettes et nytt teknologifag som skulle inneholde programmering som en betydelig komponent (Sanne et al., 2016). På den andre siden av debatten var regjeringens digitaliseringsstrategi og innstillingen fra regjeringen om at gjeldende fag- og timefordeling skulle ligge til grunn for fagfornyelsen. De mente at programmeringen skulle underlegges allerede eksisterende fag i skolen (Kirke- utdannings- og forskningskomiteen, 2016; Kunnskapsdepartementet, 2017). Dette resulterte i at fra høsten 2020 ble programmering en del av grunnskoleopplæringen i matematikk, naturfag, musikk og kunst og håndverk. Mens anvendelsen av programmering er lagt til alle fire fag, er ansvaret for opplæringen i programmering lagt til matematikkfaget (Utdanningsdirektoratet, 2020a).

Dette resulterte i at det ble innført kompetansemål i matematikk, på alle trinn i grunnskolen, som handler om å lære seg å programmere. Et eksempel er dette kompetansemålet etter 4. trinn: «Lage algoritmer og uttrykke dem ved bruk av variabler, vilkår og løkker» (Utdanningsdirektoratet, 2020c, s. 8). Elevene skal altså lære seg flere ulike programmeringsferdigheter allerede tidlig i barneskolen. I en skolehverdag med allerede stor stofftrengsel kan en stille spørsmål ved hvordan elevene skal få tid til å lære dette i tillegg, og om det finnes et godt pedagogisk verktøy for planlegging og gjennomføring av undervisning i henhold til de retningslinjene Utdanningsdirektoratet har lagt?

1.1 Programmering i skolen

Sentance et al. (2019b) sier at den tradisjonelle programmeringsundervisning er preget av kognitiv læringsteori, altså at læring foregår som mentale prosesser. Forelesninger og undervisning følger et strukturert læringsopplegg, og er ofte basert på at elevene kopierer koden som læreren presenterer mens programmeringskonsepter blir forklart (Sentance et al., 2019b). En motsats til dette er konstruktivistene, som mener mer ustrukturerte og utforskende læringsaktiviteter kan være en god måte å undervise programmering på. Forkjempere for denne retningen mener tilnærmingen kan bidra til å utvikle andre ikke-fagspesifikke ferdigheter som kreativitet og problemløsning. De mener altså at programmering ikke bare er et mål i seg selv, men en vei til å nå andre mål (Dolonen et al., 2019).

Utdanningsdirektoratet har skrevet et notat til læreplanen i matematikk kalt «Hva er nytt i matematikk?». Her beskriver de noen endringer i matematikkfaget, og innføringen av

programmering som en del av matematikkfaget. Her står det at algoritmisk tankegang ble vektlagt og synliggjort i læreplanen i fagfornyelsen 2020. Utdanningsdirektoratet argumenterer med at dette er fordi det er en viktig problemløsningsstrategi. Intensjonen med innføringen av programmering i matematikkfaget kommer også frem under dette punktet. De skriver følgende *«I læreplanen er algoritmisk tenkning synliggjort fordi dette er en viktig problemløsningsstrategi. Når elevene bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse»* (Utdanningsdirektoratet, 2020a). Elevene skal altså lære seg programmering for å kunne bruke dette som et verktøy til å utforske og løse problemer, samtidig som de utvikler matematisk forståelse.

Utdanningsdirektoratet har lagt seg tett opptil det konstruktivistiske synet på programmeringsundervisning i skolen. Utfordringen med konstruktivismen, er at det inntil nylig har eksistert lite konkret forskning på at denne tilnærmingen fungerer, og forskningen som finnes er motstridende. Studier fra Papert (1980), Benton et al. (2018) og Mørch et al. (2019), konkluderer ulikt når de diskuterer om programmering bidrar til å utvikle matematisk forståelse. Ideen med konstruktivisme stammer fra Papert (1980), han var en sterk forkjemper for at programmering bidro til at elevene lærte ikke-fagspesifikke ferdigheter som problemløsning og kreativitet som hadde overføringsverdi til andre fag. I nyere tid har vi hatt enkelte studier som finner en sammenheng mellom programmering og læring i fag som matematikk. Et eksempel er studien av Benton et al. (2018) som peker på at elever kan oppnå et godt læringsutbytte i matematikk gjennom å lære programmering i tilpassede settinger. En studie av Mørch et al. (2019) konkluderer derimot med at elevene ikke lærte noe i andre fag gjennom programmeringen, men at elevene kunne se sammenhengene mellom det de programmerte og fagene. I en litteraturgjennomgang fra 2018 forteller Waite (2018) at lærere i skolen blir presentert for et stort antall læringsressurser for å lære bort programmering, men det er lite empirisk forskning på den underliggende pedagogikken og hva som er effekten av ulike tilnærminger i undervisningen. Han konkluderer derfor med at det trengs mer forskning på området.

På tross av manglende forskning på emnet og tvetydige svar på om programmering kan hjelpe elevene i fagene slik Utdanningsdirektoratet påstår, innføres nå programmering i skolen under allerede eksisterende fag. Dolonen et al. (2019) gjorde i 2019 en litteraturgjennomgang av programmering i skolen, og konkluderte med: *«Programmering innføres relativt raskt i skolen, og gir betydelige institusjonelle utfordringer med hensyn til lærernes kompetanse. Få lærere har kompetanse i å undervise i programmering, noe som gjør at det er et behov innen feltet for å finne frem til gode læringsressurser og undervisningsopplegg»*. Lærerne trenger altså et godt pedagogisk verktøy som kan hjelpe dem med å designe og produsere egne undervisningsopplegg i programmering. Disse verktøyene kan gjøre det mulig å utforme og gjennomføre undervisning som oppfyller Utdanningsdirektoratets ønske, altså at elevene kan bruke programmering som et verktøy for å lære problemløsning og matematikk (Utdanningsdirektoratet, 2020a).

Noen av de som har forsøkt å løse dette problemet er Sentance et al. Gjennom flere artikler presenterer de en modell for pedagogisk opplæring i programmering PRIMM. I denne

studien benyttes pedagogisk opplæring og kunnskap om de teknikkene som er hensiktsmessige ved overføring av definerte læringsmål til en bestemt målgruppe. PRIMM står for *Predict, Run, Investegate, Modify og Make*. Modellen bygger i hovedsak på fire tidligere forskningsprosjekt «Tracing and read-before-you-write», «Use-Modify-Create», «Levels of Abstraction» og «Block Model» (Sentance et al., 2019a, s. 3). Jeg kommer til å gå nærmere inn på PRIMM modellen og de ulike delene den består av i kapittel 2.3. Gjennom å forene og sette sammen disse områdene forsøker de å finne en middelvei mellom klassisk opplæring i programmering og utforskende problemløsning. PRIMM modellen er testet ut på et utvalg ungdomsskoler i England. Resultatet fra denne utprøvingen har vært en signifikant forbedring av elevenes resultater i en posttest sammenliknet med kontrollgruppen som fikk ordinær programmeringsundervisning. Tilbakemeldingene fra lærere som deltok i prosjektet, var at de syntes modellen var nyttig og de mente at den støttet elevene til å få en bedre forståelse for konseptene i programmering (Sentance et al., 2019a, 2019b).

1.2 Problemstilling og forskningsspørsmål

I læreplanen er det lagt vekt på at elevene skal bli gode problemløsere. Utdanningsdirektoratet (2020a) skriver at når elevene bruker programmering til å utforske og løse problemer, kan programmering være et godt verktøy for å utvikle matematisk forståelse. I denne masteroppgaven vil jeg derfor se på hvordan programmering kan tilrettelegge for problemløsende aktivitet, helt spesifikt er denne studiens problemstilling som følger:

På hvilken måte kan programmering tilrettelegge for problemløsende aktivitet i matematikk?

Programmering omfatter mer enn bare å skrive programkode. I rapporten «Programmering i skolen», beskrives programmering som hele prosessen fra å identifisere et problem og tenke ut mulige løsninger på problemet, til å skrive kode som kan forstås av en datamaskin, samt å feilsøke og kontinuerlig forbedre koden (Utdanningsdirektoratet, 2018). Dolonen et al. (2019) konkluderer i sin forskning med at innføringen av programmering i skolen gir betydelige institusjonelle utfordringer. Lærere har stort sett lite kompetanse i å undervise programmering, noe som gjør det nødvendig å finne gode læringsressurser og undervisningsopplegg. Dolonen et al. (2019) konkluderer også med at det finnes lite adekvat forskning på undervisningsopplegg, men at det finnes to lovende opplegg. Ett av disse er PRIMM-modellen. Den kan være en god metode for lærere i Norge når de skal designe undervisningsopplegg og jobbe med programmeringsundervisning. For å si noe om dette må jeg først undersøke om elever, gjennom arbeid med programmering i PRIMM modellen, samtidig jobber problemløsende og på den måten møter intensjonen til læreplanen i matematikk. Arbeidet med programmeringen må ikke bli en ekstra belastning på elever og lærere i grunnskolen, men heller et verktøy for å utvikle matematisk forståelse.

For å svare på problemstillingen om hvordan programmering kan tilrettelegge for problemløsende aktivitet vil jeg se på elevene som jobber med et PRIMM opplegg. Her vil jeg ha følgende forskningsspørsmål for å hjelpe meg på veien:

Hvilke problemløsningsstrategier kommer til syne når elevene jobber etter PRIMM-modellen?

1.3 Oppbygging av studien

I denne studien vil jeg først legge frem relevant teori på programmering og programmeringens plass i skolen (kap 2.2). Jeg gir en innføring i PRIMM som modell og grunnlaget denne står på (kap 2.3), deretter vil jeg se på problemløsning både i matematikk (kap 2.4) og programmering (kap 2.5) og hvordan disse henger sammen. Denne teorien vil danne et bakteppe for undersøkelsene jeg vil gjøre, og skape det faglige grunnlaget når jeg senere vil diskutere mine funn i undersøkelsene.

I metodekapittelet (kap 0) vil jeg snakke om hvilke metodiske valg jeg har gjort, hvilke konsekvenser dette har for dataen jeg kan samle inn og hvilken relevans oppgaven har for andre som er interessert i feltet.

Til slutt vil jeg legge frem en analyse (kap 4) av funnene jeg har gjort gjennom arbeidet med denne studien, og en drøfting (kap 5) rundt disse funnene. Gjennom dette håper jeg å kunne konkludere innenfor de rammene jeg setter, om hvordan programmering og programmeringsundervisning kan tilrettelegge for problemløsende aktivitet, og om PRIMM kan være et godt verktøy, slik at elevene tenker problemløsende, og utvikler matematisk kompetanse i programmering slik Utdanningsdirektoratet hevder.

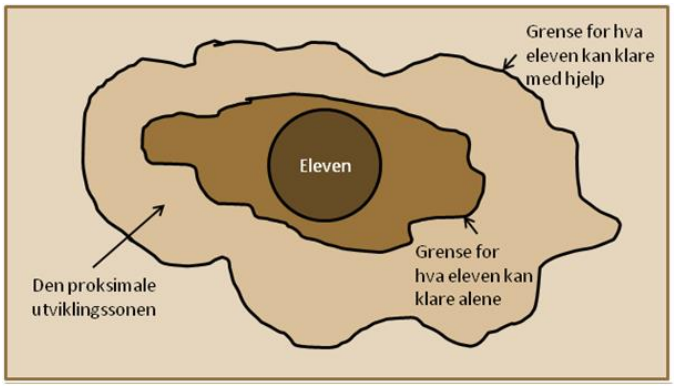
2 Teori

I denne studien undersøker jeg en teoretisk metode for oppbygging av programmeringsundervisning, PRIMM. Jeg undersøker om elevene arbeider problemløsende når de jobber med programmering i undervisningsøkter laget med denne modellen. Sentrale begreper i min studie som blir forklart i teorien er derfor programmering, PRIMM, problemløsning og *computational thinking*. Jeg skriver litt om innføringen av programmering i skolen og hvor skolen står i dag, som et fundament for resten av teorien som omhandler programmering i matematikk. Jeg gir også en kort innføring i programmeringsspråket Scratch, som ble brukt i denne studien.

2.1 Sosiokulturell læringsteori

Et viktig grunnlag i mange læringsteorier i dag er den sosiokulturelle læringsteorien. Spesielt er Vygotskys teorier om den proskimale utviklingssonen og mediering viktige. I denne studien er disse teoriene viktige fordi de støtter opp om og ligger som et grunnlag for de teoriene jeg ser på i denne oppgaven, nemlig problemløsning, *computational thinking* og PRIMM.

I det sosiokulturelle synet på læring er barns læring i samhandling med andre sentralt. I Vygotskys bok «*Mind in society*» fra 1978 beskriver han hvordan barn mestrer gjennom interaksjon, samarbeid og samspill med andre for deretter å mestre på egenhånd på det indre plan (Vygotsky, 1978). Den proksimale utviklingssonen beskrives av Vygotsky som avstanden mellom hva en elev klarer å gjøre eller løse på egenhånd (nåværende utviklingssone), og hva de kan mestre ved hjelp av en lærer eller en «*more capable peer*» (den proksimale utviklingssonen). Vygotsky beskriver at det å jobbe i samhandling med andre kan hjelpe barn å nå sin proksimale utviklingssone. Et barn kan altså i samhandling med andre løse problemer som ikke ville vært mulig å løse på egenhånd. Vygotsky skiller mellom problemløsning eleven klarer på egenhånd, og problemløsning eleven mestrer i samarbeid med en voksen eller andre mer kapable barn (Vygotsky, 1978).



Figur 1: Den proksimale utviklingssonen Imsen (2005, s.259)

Et annet viktig konsept som ble innført av Vygotsky er mediering. Han skriver at mentale prosesser var resultat av medierende artefakter der det var tre hovedkategorier, materielle verktøy, psykologiske verktøy og andre mennesker (Vygotsky, 1981). I et klasserom kan disse artefaktene være bøker, kalkulatorer, hjelp og støtte fra andre, digitale hjelpemidler,

plakater osv. Mediering setter oss i stand til å bevege oss fra det sosiale planet til det kognitive planet ved hjelp av disse hjelpemidlene. Mer spesifikt i konteksten av denne oppgaven vil viktige medierende artefakter være læreren, de andre elevene på en gruppe, programmeringsspråket Scratch og andre hjelpemidler elevene blir tilbudt gjennom opplegget.

En nøkkelfaktor i Vygotskys sosiokulturelle læringsteori er at språk er en viktig medierende faktor. Det muntlige og skriftlige språket muliggjør bruk av mange ulike medierende artefakter. Vygotsky beskriver flere grunner til at det skriftlige språket er vanskeligere enn det muntlige, blant annet de formelle reglene i oppbygningen av et skriftlig språk (Vygotsky, 2004). I denne oppgaven er en av de viktigste komponentene et skriftlig språk, nemlig programmeringsspråket. Vi kan anta at det å skrive en programmeringssekvens, vil være vanskeligere enn det å muntlig forklare hva en programmeringssekvens gjør (Sentance et al., 2019b).

2.2 Programmering

Programmering er for eksempel å gi en datamaskin instrukser om hva den skal gjøre. Da må vi bruke et språk som datamaskinen forstår. Dette gjøres ved å sette sammen flere kommandoer som sier noe om hvordan maskinen eller programmet skal reagere på en innputt som et museklikk eller tastetrykk. Programmering kan foregå på mange nivåer, fra datamaskinens maskinkode med 0 og 1, til lavnivåspråk og opp til høynivå språk som C++ og Python. I dag finnes også visuelle programmeringsspråk som baserer seg på blokker i stedet for tekst. Her representerer hver blokk flere linjer med tekst kode som hjelper den som programmerer til å sette sammen kode som fungerer sammen. I dette prosjektet brukte elevene et slikt program, Scratch, og jeg kommer tilbake til hvordan dette språket fungerer, senere i teori- og metodedelen i kapittel 2.6. Utdanningsdirektoratet skriver i sin kompetansepakke til lærere i norsk skole at programmering handler om algoritmisk tenking sammen med kommunikasjon, både med en datamaskin og mellom mennesker (Utdanningsdirektoratet, 2020b).

Figur 2: Programmering= Tenking + kommunikasjon (Utdanningsdirektoratet, 2020b)

2.2.1 Et historisk perspektiv på programmering i skolen

I skrivende stund har fagfornyelsen 2020 trått i kraft og med det har programmering blitt en del av skolehverdagen til mange tusen barn, unge og ikke minst lærere. For skoler

rektorer og lærere over hele landet fører dette til nye muligheter og utfordringer i skolehverdagen. Likevel var ikke programmering et ukjent emne før fagfornyelsen ble innført. Tidlig forskning om bruk av programmering i skolen fant sted allerede på 1980 tallet. En av de ledende forskerne den gang var Seymour Papert (Dolonen et al., 2019).

Programmering og matematikk kan sies å ha hatt en grunnleggende sammenkobling helt siden programmeringens spede begynnelse. Det var matematikeren Alan Turing som fant opp den første programmerbare datamaskinen og programmering som språk. Senere på 1980 tallet var teknologien på full fart fremover i USA og verden for øvrig. International Business Machines corporation (IBM) lanserte sin første Personal Computer i 1981, og teknologien ble nå mer og mer tilgjengelig. Papert så på denne utviklingen og var en av dem som så nærmere på hvordan datamaskinene kom til å påvirke vår læring. I boken *«Mindstorms: children, computers and powerful ideas»* skriver Papert: *The child programs the computer. And in teaching the computer how to think, children embark on an exploration about how they themselves think* (Papert, 1980, s. 19). Han beskriver her hvordan elever kan lære gjennom bruk av programmering. Han håpet at elevene kunne være aktive deltagere i egen læring, og bli gode problemløsere gjennom programmering.

2.2.2 LOGO – starten på programmering i skolen

Papert og kollegaene utviklet samtidig det første programmeringsspråket som var tiltenkt bruk i opplæring av elever, LOGO. I dette programmet programmerer elevene en skilpadde til å flytte seg rundt på skjermen og tegne streker. På denne måten kunne elever arbeide med programmering og geometri samtidig. Gjennom forsøk med elever viste han at det var mulig for elever i tidlig alder å lage algoritmer. Elevene arbeider undersøkende, for selv om datamaskin og LOGO ikke er nødvendig for å tegne geometri, ville det hjelpe elevene å undersøke, prøve og feile og finne løsninger på problemer som oppstår på en ny måte. Han hadde stor suksess med elever som arbeidet på denne måten, og argumenterte derfor for at denne typen opplæring burde være et ideale i skolen. Han mente det hjalp elevene med å forstå noe nytt i lys av noe de kunne fra før. Og gjennom å arbeide på denne måten over tid mente han at elevene kunne ta med seg det de lærer over i andre situasjoner også utenfor programmeringen i LOGO (Papert, 1980).

Etter at IBM lanserte sine «personal computers» til hjemmebruk utover 80-tallet, og teknologien ble mer tilgjengelig, hevdet Papert i 1993 i sin bok: «*The children's machine: Rethinking school in the age of the computer»,* at man måtte gjennomføre en reform i skolen på bakgrunn av at elevene fikk en generell problemløsningsevne og kognitiv utvikling som følge av programmeringen (Papert, 1993). Der Papert (1993) hadde stor suksess med utprøving blant elever viste det seg likevel å være vanskelig å få støtte for påstandene om at programmering forbedret ikke fagspesifikke evner hos elevene i internasjonal forskning. Utover 90-tallet var det også færre og færre skoler som hadde programmering på timeplanen. Man lærte seg å bruke nettlesere, printere, skriveprogrammer og søkemotorer, men ikke hvordan det hele hang sammen og fungerte (Dolonen et al., 2019).

I dag er forutsetningene for programmering i skolen noe helt annet. Den teknologiske utviklingen har skutt fart og teknologien er med oss i alle deler av hverdagen og arbeidslivet. Også verktøyene for å lære seg programmering har utviklet seg i stor grad. Spørsmålet om hva programmering kan gi elever i skolen er på nytt blitt høyaktuelt (NOU 2015:8).

2.2.3 Ludvigsenutvalget: Fremtidens skole

21.juni 2013 ble et utvalg ledet av Stein Ludvigsen oppnevnt for å *«vurdere grunnopplæringens fag opp mot krav til kompetanse i et fremtidig samfunns- og arbeidsliv»* (NOU 2015:8, s. 5). Resultatet av denne utredningen ble NOU 2015:8 Fremtidens Skole, Fornyelse av fag og kompetanser. I denne rapporten beskriver utvalget hvordan samfunnet utvikler seg, og hvordan utdanningen må følge med for at elevene skal få de beste forutsetningene videre i livet. Kunnskapsmålene endrer innhold og form, og nye vilkår for elevenes læring skapes, samtidig som fremtidsrettede kompetanser utvikles (NOU 2015:8, s. 7). Rapporten beskriver 8 slike sentrale fremtidsrettede kompetanser som er viktige i det 21 århundre. 2 av disse 8 er «IKT kompetanse» og «Kritisk tenking og problemløsning». Disse kompetansene blir her anerkjent som viktige for elevene videre i deres liv (NOU 2014:7, s. 116). Disse to kompetansene kjenner vi igjen fra den nye læreplanen i matematikk og notatet «*hva er nytt i matematikk*» (Utdanningsdirektoratet, 2020a). Her skriver Utdanningsdirektoratet at algoritmisk tankegang er synliggjort fordi det er en viktig problemløsningsstrategi, og at elevene gjennom bruk av programmering kan utforske og løse problemer.

2.2.4 Programmering i skolen i dag

Ser vi tilbake på 80-tallet og programmering, er mye endret. Nå som det vi kan kalle 2. bølge av programmering skyller inn over skoler over hele Europa har programmeringen helt andre forutsetninger for å bli en suksess. Tilgjengeligheten til programmering har blitt stor gjennom et stort antall programmerbare objekter og programmeringsspråk tilpasset læring. Der man før hadde LOGO, har man i dag blokkbaserte programmeringsspråk som Scratch (Scratch, u.å.), Blockly (Blockly, u.å.) og micro:bit (Micro:bit, u.å.). Der man tidligere programmerte en skilpadde på en skjerm, har man i dag fysiske gjenstander å jobbe med som Spero, Bitbot, Beebot, Raspberry Pi og mange flere (O'Brien, 2019). Felles for disse er at de er utviklet spesifikt med tanke på opplæring av barn i programmering.

Det fantes allerede i 2016, før programmering ble en del av læreplanen, 98 ulike kodeklubber fordelt i hele Norge, og det var over 125-tusen elever som deltok på kodetimen i regi av kidsakoder.no i 2020 (Lær Kidsa Koding, 2021). Mange barn og unge har allerede et forhold til det å programmere (Utdanningsdirektoratet, 2018). Programmering og gode verktøy for å lære programmering finnes det altså mengder av. Så hva er da utfordringen? I sin gjennomgang av programmeringsundervisning i skolen skriver Waite (2018): *«Teachers are currently presented with a plethora of educational*

technology resources that lack pedagogical instruction». Ifølge Waite er det mangelen på gode pedagogiske ressurser som hemmer undervisningen av programmering i skolen i dag. I en liten undersøkelse gjort av lærere i barneskolen, kommer det frem at lærerne mener de kan for lite om programmering, likevel viser undersøkelsen at lærerne er positive til innføring av programmering i matematikk. Samtidig legger de vekt på at de selv ønsker å lære å programmere, og hvordan de skal undervise, heller enn å få ferdige opplegg til bruk i klasserommet (Berggren & Jom, 2019).

Utdanningsdirektoratet har anerkjent utfordringene innføringen av programmering i skolen fører med seg. De har derfor kommet med flere tiltak for å lette overgangen for skolene og lærerne. Utdanningsdirektoratet har lagt på bordet ekstraordinære midler som skolene kan søke på så fremt de prioriterer videreutdanning og kompetanseheving i programmering for lærere. Pengene skal gå til utstyr som kan brukes i undervisning som digitale enheter, roboter og tilleggsutstyr for programmering. I 2020 ble det delt ut 20 millioner i slike ekstraordinære midler, og de neste 4 årene skal det deles ut 15 millioner hvert år (Utdanningsdirektoratet, 2020d).

Utdanningsdirektoratet skriver at *«Mange lærere er usikre på hvordan de skal undervise i programmering og algoritmisk tenking i sine fag.»* (Utdanningsdirektoratet, 2020b). Som en respons har de laget en kompetansepakke tilpasset lærere i skolen. Pakken er ment som en kompetanseheving hvor opplæringen er rettet direkte mot de ulike fagene i skolen som har kompetansemål i programmering. Her legger de vekt på at det å undervise elever i programmering handler om å gi dem flere verktøy til å løse oppgaver og problemer. I kompetansepakken for programmering i matematikk legger Utdanningsdirektoratet (2020b) frem at det er fire argumenter for hvorfor programmering skal styrke undervisningen i matematikk:

1. Problemløsning: *Elevene skal lære å bli problemløsere. Med programmering kan de gå løs på helt nye og mer virkelighetsnære problemstillinger, med flere strategier og effektive verktøy. For eksempel kan elevene samle egne datasett med mobilen og deretter utforske datasettene ved hjelp av programmering.*
2. Eksperimentering, prøving og feiling: *Programmering åpner for eksperimentering, prøving og feiling på en annen måte enn vi kanskje er vant med fra før. Hvis programmet ikke fungerer, får du vite det med en gang. Da er det bare å lete etter egne feil og prøve på nytt. Slike umiddelbare tilbakemeldinger kan føre til større utholdenhet hos elevene dine.*
3. Læring for flere: *Elevene skal få lov til å finne sin vei inn i matematikken. Noen elever liker å tenke og abstrahere, andre liker å fikle, teste og eksperimentere. Programmering og algoritmisk tenkning åpner for mange ulike tilnærminger til matematikk. Hvis du som lærer behersker mange ulike metoder, kan du lettere tilpasse opplæringen til hver enkelt elev.*
4. Dybdelæring: *En av nøklene til dybdelæring er bruk av ulike representasjonsformer. Matematiske problemer kan ofte løses med og uten programmering. Å klare begge deler kan gi en dypere forståelse for matematikken. Samtidig utvikler elevene en bevissthet om hvilke verktøy som egner seg for hvilke problemer* (Utdanningsdirektoratet, 2020b).

I notatet programmering i skolen står det *«Programmering gir også en systematisk tilnærming til problemløsning»* (Utdanningsdirektoratet, 2018). Her menes det at programmeringen kan gi en kompetanse i å lære. Gjennom strukturerte aktiviteter med prøving og feiling, systematisk feilsøking, samt utforsking for å finne de beste løsningene og generalisering av løsninger, mener de at elevene lærer seg å lære. De lærer seg å bli bedre problemløsere (Utdanningsdirektoratet, 2018).

Til tross for nyskapningen innen programmeringsverktøy og en voksende kunnskap om programmering og programmeringsverktøy, er det stadig en spenning mellom denne kunnskapen og den pedagogiske kunnskapen i programmering. Lærere og elever har ofte et begrenset syn på programmering knyttet til steg for steg instruksjoner i læringsverktøyene som finnes (Benton et al., 2016). For å løfte programmeringsundervisningen til neste nivå, må vi finne et pedagogisk verktøy som kan hjelpe lærerne å designe egne undervisningsopplegg som kan tilpasses de ulike temaene i matematikkfaget. Benton et al. (2016) sier at lærerne må sette pris på det viktigste målet med programmering, som er kraften i algebraisk tenking, og hvordan det kan brukes i matematikk undervisningen. Programmering i skolen har vist seg å ha potensiale til å hjelpe elever med å utvikle et høyere nivå av matematisk tenking både når det kommer til abstraksjon, algebraisk tenking og problemløsning (Clements, 1999).

2.3 PRIMM modellen

Det finnes i dag få svar på hva som er effektiv pedagogikk i undervisning av programmering. Hva som er god pedagogikk avhenger av sammenhengen hvor programmeringen brukes. Waite (2018) skriver at «R*esearch is needed to develop and evaluate teaching and learning pedagogies for cross-curricular computing in primary and secondary schools»*. Det er per i dag gjort lite forskning på feltet. En av de få forskergruppene som jobber med dette er Sentance et al. (2019a, 2019b). Med reformasjonen som foregår rundt programmering i England mente de at det var viktig å skape en pedagogisk modell å jobbe ut ifra når lærere lager undervisningsopplegg for elever. Dette kan støtte lærere i arbeidet med programmering, som kan være et utfordrende emne for noen elever. (Sentance et al., 2019a).

PRIMM modellen bygger på fire tidligere teorier. Først vil jeg kort ta for meg grunnlaget for PRIMM, deretter se nærmere på hvordan PRIMM modellen er tenkt brukt, og hva som ligger i hvert punkt i modellen.

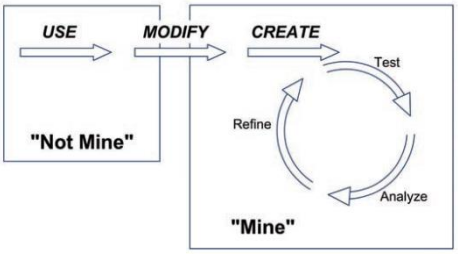
2.3.1 Grunnlaget til PRIMM

I startfasen av utviklingen av PRIMM-metoden undersøkte Sentance et al. (2019b) elever i den engelske skolen og hva som gjorde at de strevde med begynneropplæringen i programmering. Her identifiserte de at den tradisjonelle metoden for programmeringsundervisning hvor elevene kopierer koden læreren skriver, ikke fungerte

godt for barn i barneskolealder. Med dette som grunnlag har Sentance et al. (2019b) satt sammen fire teorier, som til sammen utgjør PRIMM-modellen. Disse teoriene er som følger:

- *Tracing and reading before you write* (Lister et al., 2004). Dette er en rapport utarbeidet til den årlige Innovation and Technology in Computer Science Education konferansen. Rapporten ble utarbeidet som et resultat av at mange elever ikke kunne å programmere på slutten av introduksjonskurset i programmering. Rapporten peker i sin konklusjon på at elevene ikke er i stand til å lese og forstå koden de jobber med, og at elevene mangler evne til å drive problemløsning. Rapporten forteller at lesing og forståelse av kode må være en viktig del av arbeidet med å lære koding på lik linje med problemløsningskompetanse. Lister beskriver at elever må være i stand til å lese kode med mer enn 50% nøyaktighet før de selv er i stand til å produsere kode.

- *Use modify create* (Lee et al., 2011) er en annen pedagogisk modell for programmering utviklet av Lee et al, publisert i 2011. Ideen i denne modellen er at den som skal lære kode, først kjører en eksisterende kode for å se hva koden gjør, for deretter å modifisere koden. Først når eleven er trygg på koden kan hen skrive egen liknende kode fra bunnen av.



Figur 3: Use- Modeify- Make. Lee et al., (2011)

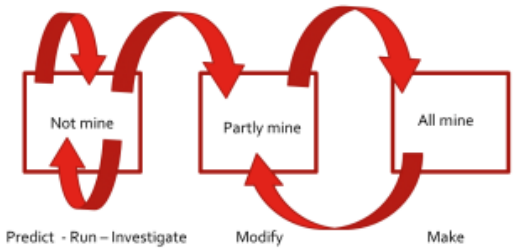
- Abstraction Transition (AT) taxonomy (Cutts et al., 2012) er en egen studie som ser på forståelse av abstraksjoner ved koding. Studien peker på tre hovednivåer av abstraksjon: Koding, programmeringsspråk og språk (engelsk). De mente at gjennom bevist bruk av alle nivåene kunne elevene styrke programmeringsferdighetene sine

- Block Model (Schulte, 2008) er en modell som beskriver elevers utvikling av kunnskap om programmering. Modellen deler opp forståelsen i fire nivåer: Enkeltlinjer med kode, grupper med kode, kode som henger sammen og hele programmet. For å få fullstendig forståelse må eleven forstå koden på hvert nivå. Dette finner vi spesielt igjen i *investigate* delen av PRIMM.

2.3.2 PRIMM modellen.

Primm modellen består av 5 faser i en undervisningsøkt. Intensjonen i det pedagogiske verktøyet er å kunne lage undervisningsøkter og undervisningsperioder med innhold som strekker seg over de 5 fasene av læring. Disse fasene er:

- *Predict*: Forutse hva koden kommer til å gjøre.
- *Run:* Kjøre koden for å sjekke antagelsene fra *Predict*.
- *Investigate:* Undersøke strukturene i koden.
- *Modify*: Modifisere koden for å endre eller legge til funksjonalitet.
- *Make*: Lage et nytt program med de samme eller liknende strukturer.



Figur 4: PRIMM modellen. Scentance et al (2019)

Intensjonen bak modellen er at læreren skal kunne sette seg inn i modellen og det læringsmateriellet som er produsert, for å kunne lage sine egne undervisningsopplegg ut fra denne modellen. Hvert steg i modellen har sin egen rolle, og skal hjelpe elevene med å lære seg å programmere. (Sentance et al., 2019a)

2.3.2.1 Predict

I den første fasen forteller Sentance et al. (2019b) at elevene skal undersøke et kort program eller et utsnitt av et program for å prøve å avgjøre hva dette programmet gjør. Oppgaven til elevene blir å diskutere sammen i par hva hver del av programmet gjør, for deretter å skrive ned hva de tror resultatet av å kjøre koden blir. Denne aktiviteten oppfordrer elevene til å se etter ledetråder i programmet som antyder hva kodens funksjon er (Sentance et al., 2019b).

2.3.2.2 Run

Etter *predict* fasen sier Sentance et al. (2019b), at læreren skal diskutere elevenes svar sammen med klassen. Deretter skal elevene laste ned koden delt av læreren og kjøre denne. Her er et viktig poeng i PRIMM at elevene ikke skal kopiere koden fra smartboard eller liknende, da kopiering av kode er en helt annen prosess enn det Sentance et al. (2019b) ønsker å oppnå i dette steget av PRIMM. Det å se på ferdigskrevet kode har mange fordeler, det flytter ansvaret for alle feil i koden over på læreren, slik at elevene kan fokusere på forståelse. Det sparer også tid, da det å kopiere kode kan være tidkrevende og vanskelig spesielt for elever som sliter med andre utfordringer som lesing og skriving. Når elevene kjører koden, får de se om deres spådom stemte (Sentance et al., 2019b).

2.3.2.3 Investigate

I denne delen forteller Sentance et al. (2019b) at læreren skal stille spørsmål som utfordrer elevenes kodeforståelse. For eksempel spørre hva som skjer med koden hvis vi utelater en spesifikk del. Dette krever at eleven har forståelse av den underliggende algoritmen og kjøringen av koden. Læreren kan også stille spørsmål om enkeltdeler av koden i programmet. Disse spørsmålene skal hjelpe elevene å utvikle forståelse om enkeltnivåer i programmering (Sentance et al., 2019b). Ideelt sett skal diskusjoner i denne fasen foregå

mellom enkeltelever eller i grupper, slik at elevene utvikler et vokabular for å snakke om programmering. I denne fasen er det viktig at læreren har god forståelse for både programmering og elevers misoppfatninger i programmering, for å kunne stille gode spørsmål. Disse tre delene av økta kan gjentas flere ganger, og elevene kan jobbe seg igjennom flere liknende koder til de forstår hvorfor koden får akkurat dette resultatet når den kjøres. Det kreves gjerne mange gjentakelser av denne aktiviteten for elevene for å forstå de underliggende konseptene i en kode som læreren forsøker å lære bort (Sentance et al., 2019b).

2.3.2.4 Modify

Sentance et al. (2019b) snakker i det fjerde steget i PRIMM om elevenes eierforhold til programkoden. Frem til nå har ikke elevene hatt noe eierforhold til koden det jobbes med, men i dette steget skal eleven gradvis få eierskap over programkoden. I denne fasen skal elevene bygge på koden de har jobbet med frem til nå, for så å endre og skape nye funksjoner i koden. Ved hjelp av gradvis oppbygging av oppgavene i denne fasen, kan man skape progresjon fra enkle endringer til mer grunnleggende endringer i koden (Sentance et al., 2019b). Det at elevene har en kode som de forstår, gjør at de får mere selvtillit i arbeidet med å endre koden. Mulige oppgaver i denne fasen kan være å fjerne tydelige innlagte feil i koden de jobber med, endre resultatet eller hvordan du gir programmet instrukser. Denne aktiviteten kan flytte eierskapet av koden eleven jobber med fra å «ikke være min» til å bli «delvis min». Denne delen fungerer som stilas for elevene for at de senere skal kunne mestre å lage egen kode innenfor de samme konseptene (Sentance et al., 2019b).

2.3.2.5 Make

I det siste steget i PRIMM forteller Sentance et al. (2019b) at elevene skal lage egen kode. Her kan oppgaven være å designe egne program ut ifra et problem gitt fra læreren. Her er poenget at elevene kan bruke konseptene de har lært i tidligere økter sammen med konseptet fra denne økta til å lage kode som kan løse nye problemer. Innlæring av ett enkelt konsept i programmering gjennomføres i løpet av en økt hvor alle delene av PRIMM er til stede, mens et mer avansert konsept kan kreve flere økter og flere gjennomføringer av PRIMM-modellen som helhet. Det å skape et eget program er en avansert prosess og en viktig ferdighet. Prosessen bør starte med at eleven lager en passende algoritme for å løse problemet. Dette er vanskelig, men gir eleven muligheter til å være kreativ, og til å få tilfredsstillelsen av å lage sitt eget program (Sentance et al., 2019b).

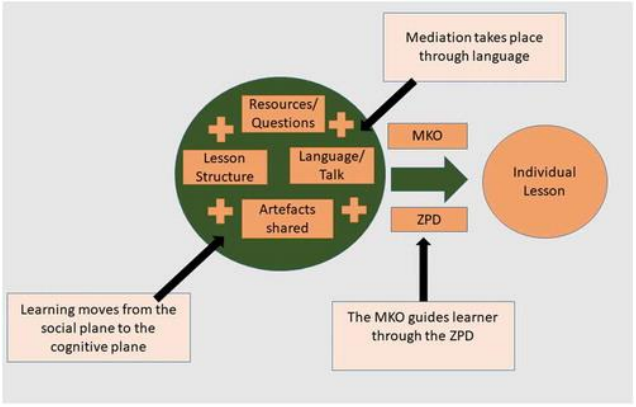
2.3.3 En modell for å lære bort programmering.

PRIMM bygger tett på de sosiokulturelle læringsteoriene til Vygotsky. Fra disse teoriene er det i hovedsak 3 nøkkelprinsipper som skal guide lærere og elever gjennom læring av og med programmering.

Mediering gjennom muntlig språk: her skal elevene oppfordres til å snakke sammen om programmering. Elevene skal jobbe sammen for å forstå ulike deler med kode, og utrykke hva de tror kommer til å skje (Sentance et al., 2019b).

Læring flyttes fra det sosiale til det kognitive planet: Gjennom at elevene mottar eksisterende kode vil koden eksistere i det sosiale planet først før koden etter hvert kan forstås internt hos eleven. Etter hvert som eleven har internalisert flere konsepter vil eleven kunne jobbe med mere komplekse programmeringsoppgaver, og drive uavhengig problemløsning (Sentance et al., 2019b; Vygotsky, 1981).

Læreren eller eleven som støtte for å nå den proksimale utviklingssonen: Elevene kan få hjelp av læreren som støtte for å nå den proksimale utviklingssonen, gjennom å modellere for elevene hvordan de kan løse problemer i programmering. Andre elever kan også støtte denne utviklingen gjennom parprogrammering eller gruppearbeid, hvor arbeidet er konsentrert om noen oppgaver eller strukturer som er innenfor elevenes proksimale utviklingssone (Sentance et al., 2019b; Vygotsky, 1978).



Figur 5: Modell for design av en programmeringsøkt (Sentance et al., 2019b)

2.3.4 Forskning på og utprøving av PRIMM

PRIMM ble prøvd ut i 2 runder med studier i 2017. Pilotstudien besto av 6 lærere og 80 elever. I hovedstudien deltok 14 lærere og 493 elever. Lærerne underviste programmering ved hjelp av PRIMM gjennom 10 undervisningsøkter fordelt på 8 til 12 uker. Lærerne fikk tilgang på fullt undervisningsmateriell for de 10 øktene, med presentasjoner, arbeidsark, program og fasit til oppgavene. Elevene i studien, pluss 180 elever som deltok som en kontrollgruppe, gjennomførte en pre- og posttest. Studiene viste at elevene som ble undervist gjennom undervisningsopplegg laget med PRIMM, hadde signifikant bedre resultater i en posttest enn kontrollgruppen. Her ble forståelse og måloppnåelse målt både gjennom kvantitative og kvalitative undersøkelser. Lærere og elever ble også intervjuet gjennom prosessen. I intervjuene fortalte lærerne at elevene hadde et godt utbytte av en strukturert måte å arbeide med programmering på, og flere mente at elevene hadde lært mer gjennom denne metoden enn slik de hadde undervist tidligere (Sentance et al., 2019b).

I og med at PRIMM modellen er forholdsvis ny når denne masteroppgaven skrives, finnes det få publiserte artikler eller forskningsprosjekter som har brukt denne tilnærmingen til å undervise programmering. Søk gjort i Google Scholar gir 1050 treff for «PRIMM programming». En gjennomgang av de første 10 sidene, viser at bare 10 av disse er faktiske studier som har tatt i bruk PRIMM modellen.

En av de få som har publisert noe er Law. Han har brukt PRIMM som modell gjennom en hel modul i programmering av spill i høyere utdanning. Hans konklusjon er at det tar en betydelig mengde tid å lage undervisningsopplegg etter denne modellen, og at elevene må

være engasjert i undervisningen for å få utbytte. Han skriver at elevene virker mindre bekymret for å sette i gang med koding generelt, og at de produserte programmering på et godt nivå (Law, 2020). Law skriver også at PRIMM bygger på en solid pedagogisk plattform som vises gjennom en godt definert struktur og hvor lett det er å implementere modellen i undervisningen (Law, 2020).

Udenze har sett på PRIMM modellen, og undersøkt hvordan det fungerer for mer avanserte biter med kode. Han peker på at PRIMM er en «state of the art» modell for begynneropplæring i programmering, men at modellen har noen mangler når det kommer til arbeid med programmer som har flere kodeprosesser som foregår simultant. Han peker han på at man med fordel kan bytte ut *modify* steget med «*Decompose og Arrange*», for å få mere oversikt over ulike komponenter i kode. Han foreslår derfor en oppgradering av PRIMM til PRIDAM, for å møte utfordringene som oppstår ved koding på høyere nivå (Udenze & Elfallah, 2020).

2.4 Problemløsning i matematikk

I 1945 ga Polya ut «*How to solve it*». I boka definerte Polya problemløsning som et didaktisk domene. Han skrev at *«Solving problems means finding your way out of a difficulty» (Polya, 2004).* Gjennom tidene har begrepet blitt behandlet av mange og er et komplekst begrep med mange ulike definisjoner. I dag bruker vi begrepet om å løse et problem du ikke umiddelbart har en metode for å løse. Matematikkdidaktikere har i lang tid vært interessert i å forstå hva problemløsning er, og hvordan det skal undervises i dette. Problemløsning har vært nevnt i læreplanene i matematikk siden mønsterplanen i 1987. Den forrige utgaven av læreplanen hadde en egen kompetanse for problemløsning, mens problemløsning i læreplanen i dag har blitt et eget kjerneelement i matematikkfaget. Her står det at problemløsning i matematikk handler om at elevene utvikler en metode for å løse et problem de ikke kjenner til fra før. Videre blir algoritmisk tenking fremhevet eksplisitt fordi det er en viktig problemløsningsmetode (Utdanningsdirektoratet, 2020c). Dette kommer jeg tilbake til i neste delkapittel.

2.4.1 Problemløsningsprosessen

Tidligere handlet problemløsning ofte om løsningsprosessene. Polya skriver i sin bok «*How to solve it*» fra 1945, hvordan man kan løse et problem. Dette handlet i utgangspunktet om de fire stegene «*Understanding the problem, devising a plan, carry out the plan og looking back*» (Polya, 2004). Det finnes mange varianter av Polyas strategi, og Woods skriver i en artikkel fra 2000 at det finnes så mange som 150 publiserte problemløsningsmetoder på tvers av ulike fagfelt. De fleste ligner på strategien til Polya, og inneholder et sted mellom 2 og 7 steg. Det som skiller disse er i stor grad hvordan de er tilpasset ulike fagfelt, mens hovedinnholdet er det samme (Woods, 2000). De fleste moderne problemløsningsmodellene baserer seg på stegene til Polya, og inneholder som regel stegene 1: Les og forstå problemet 2: Velg en passende strategi 3: Løs problemet og 4: Se deg tilbake og reflekter over svaret. Terminologien som blir brukt kan være

forskjellig, men ideene er de samme (Posamentier & Krulik, 2015). Et eksempel på en alternativ inndeling er Schoenfelds inndeling i de 6 delene: *Read, analyse, explore, plan, implement* og *verify.* Dette brukte han til å peke på forskjellene på hvor mye tid elever og matematikere brukte på ulike deler av et problem (Schoenfeld, 2016). I en problemløsningssetting er løsningene på slike problemer ikke bare selve svaret, men hele prosessen fra man begynner å lese oppgaven til man har et svar som man har reflektert over. Noen mener også at selve prosessen er viktigere enn det endelige svaret for det er her læringen ligger (Posamentier & Krulik, 2015). I informatikk blir *computational thinking* sett på som en problemløsningsmetode. Denne metoden blir beskrevet av Wing (2017) som tankeprosessen i å formulere et problem, og å utrykke dette slik at et menneske eller en maskin effektivt kan løse det.

2.4.2 Problemløsningsstrategier

Posamentier og Krulik (2015) skriver i sin bok «*Problem solving, Strategies in mathemathiks*» at det å velge en passende strategi (Polya sitt andre steg, lag en plan) er nøkkelen til å arbeide effektivt med problemløsning. Polya lister i sin bok opp 67 ulike heuristics eller tommelfingerregler for dette steget. Problemløsningsmetoden eller prosessen følger stort sett det samme mønsteret, derimot kan problemløsningsstrategiene for å løse selve oppgaven variere basert på hvilke fagspesifikt område metoden skal støtte elevene i å jobbe med. Posamentier & Krulik presenterer i sin bok det de ser på som de 10 viktigste problemløsningsstrategiene i møte med matematiske problemløsningsoppgaver (Posamentier & Krulik, 2015). Dette er de strategiene som forfatterne fant viktigst i problemløsning i matematikk. Definisjonen av slike strategier er viktig i problemløsning da ulike problemer kan løses på ulike måter og med ulike strategier. Posamentier & Krulik understreker at det er sjeldent at bare en problemløsningsstrategi brukes, men at problemløseren ofte tar i bruk flere (Posamentier & Krulik, 2008). *Computational thinking* fra Csizmadia et al. (2015) definerer også noen konsepter eller problemløsningsstrategier som omhandler hvordan et problem kan ordnes og presenteres slik at det kan løses av en person eller datamaskin. Csizmadia et al. (2015) beskriver 6 slike strategier, som jeg kommer tilbake til i kapittel 2.5.

I denne studien ønsker jeg å se på problemløsende aktivitet i programmering i matematikkfaget. Derfor har jeg valgt å se på et sett strategier som hører matematikken til, og et sett strategier som hører til i informatikken, for å se om noen av disse blir brukt når elevene programmerer. Under følger Posamentier og Krulik (2015) sine 10 strategier, og eksempler på hvordan de ser ut i bruk. Senere i kapitel 2.5 ser jeg på Csizmadia et al. (2015) sine strategier. Dette gjør meg i stand til å sammenlikne elevenes metoder, og å avgjøre om de bruker noen av disse strategiene i sine arbeider med programmering i matematikk.

2.4.2.1 Logisk resonement

Logisk resonnement er en av de vanligste strategiene som også benyttes regelmessig i dagliglivet. I en hverdagssituasjon er et logisk resonnement ofte bygd opp av en rekke

logiske argument. I matematikk gjør lærerne elevene oppmerksomme på de logiske sammenhengene som finnes i matematikk, gjennom oppgaver og spørsmål som for eksempel «hvis den er x hva vil det være logisk at den neste er da?». Det å løse en oppgave med logisk resonnement kan spare eleven for mye algebraisk manipulasjon hvis hen kan finne en måte å omgå de matematiske problemene (Posamentier & Krulik, 2008).

Et eksempel kan være: *Hilde har 200kr i 20 kroninger, hun har også 4 ganger så mange 5ere som 20 kroninger. Hvor mye penger har hun i 5ere?*

Her kan en vanlig løsning være å dele 200 på 20 for å få antallet 20 kroninger for deretter å gange med 4 for å få antall 5ere som er 40 stk 5ere. Til slutt må eleven regne ut at $40kr \cdot 5 = 200\,kr$. Med et logisk resonnement kunne elevene i stedet sagt at siden det er 4 ganger så mange 5-erer som 20 kroninger så må summen være den samme altså 200 kr.

2.4.2.2 Se etter mønster

En av de vakre tingene i matematikk er logikken og ryddigheten som matematikken uttrykker. Denne logikken kommer frem fysisk gjennom mønster eller rekker med mønster. Matematikeren W.W.Sawyer sa en gang at matematikk er søken etter mønstre (Posamentier & Krulik, 2008).

Hvis vi finner mønster i geometrien eller i matematikken for øvrig, kan det støtte oss i å løse problemer i geometrien eller andre områder. Et eksempel kan være oppgaven: Finn ut den indre vinkelsummen i et regulært Ikosagon (20-kant). Her kan det hjelpe oss å vite at vinkelsummen i en trekant er 180, et kvadrat er 360 og et pentagon er 540. Hver av disse figurene kan sees på som oppbygd av henholdsvis en, to eller tre trekanter. Ser vi på mønsteret som danner seg kan vi se at 6 kanten vil være 720 osv. Ved å se systematisk på mønsteret som danner seg ser vi at antallet grader er 180 * antallet trekanter som bygger opp figuren.

Antall sider	3	4	5	6	7	10	20
Antall trekanter	1	2	3	4	5	8	18
Sum antall grader.	180	360	540	720	900	1440	3240

2.4.2.3 Jobbe bakover

I begynnelsen kan denne strategien virke noe forvirrende. Elever vil i de fleste tilfeller jobbe seg fra et spørsmål og frem til et svar. Likevel vil elevene der det finnes et unikt endepunkt, og mange ulike måter å angripe problemet på, med fordel kunne jobbe seg bakover for å løse problemet (Posamentier & Krulik, 2008). Et eksempel på en slik strategi kan være ved oppgaven: *Med en 11 liter kanne og en 5 liter kanne, hvordan kan du måle opp nøyaktig 7 liter.*

Ved å jobbe seg bakover her kan man tenke seg at den eneste måten å få 7 liter på, er å ha dem i 11 liters kannen, og den er da altså 4 liter fra å være full. Hvordan kan vi helle vekk nøyaktig 4 liter, jo ved å helle de over på 5liters kannen som da må ha igjen 1 liter i seg. For å oppnå det kan vi fylle 11 liters kannen helt opp og så helle vekk 5 liter to ganger før den siste literen helles over. På denne måten kan eleven jobbe seg bakover til et svar i denne typen oppgaver hvor det endelige resultatet er kjent.

2.4.2.4 Se problemet fra andre synsvinkler

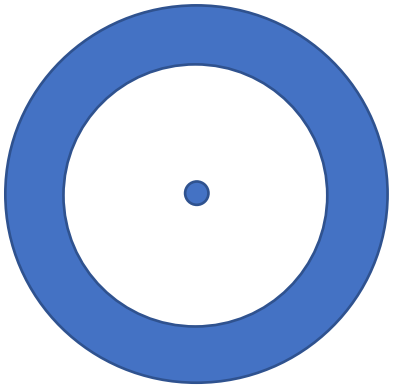
Denne strategien handler om hvordan elevene når de møter på problemer, kan se på problemet fra en annen synsvinkel og finne en ny måte å løse problemet på. I skolesammenheng er det vist at hvis en elev ikke klarer å løse et problem, vil eleven prøve å løse problemet på den nøyaktig samme måten en gang til (Posamentier & Krulik, 2008). Denne problemløsningsmetoden krever at eleven aktivt gjør en vurdering av fremgangsmåten når den første måten å løse problemet på ikke fører frem, eller er lang, vanskelig og tidkrevende.

Et eksempel kan være: Gitt utrykket $(\boldsymbol{x-10})(\boldsymbol{x-9})(\boldsymbol{x-8})\ldots.(\boldsymbol{x-3})(\boldsymbol{x-2})(\boldsymbol{x-1})$. Løs for $\boldsymbol{x=4}$. En vanlig løsning ville her vært å bytte ut x med 4 i alle leddene i utrykket for deretter å gange ut svaret. $(\boldsymbol{4-10})(\boldsymbol{4-9})(\boldsymbol{4-8})\ldots(\boldsymbol{4-3})(\boldsymbol{4-2})(\boldsymbol{4-1})$. Ser vi derimot problemet fra en annen synsvinkel kan vi med en gang se at i leddet $(\boldsymbol{x-4})$ vil summen bli $\boldsymbol{0}$ og dermed vil utrykket for $\boldsymbol{x=4}$ være lik $\boldsymbol{0}$.

2.4.2.5 Vurdere ekstremaltilfeller

Når vi ser på enkelte situasjoner både i matematikk og ellers i livet kan det å vurdere ekstremaltilfeller være en måte å løse et problem på. I matematikken kan en utforske eller se på ekstreme tilfeller og bestemme noen egenskaper ved problemet, ved å la noen variabler være de samme mens vi endrer enkelte variabler til det ekstreme (Posamentier & Krulik, 2008).

Et eksempel på denne typen tilnærming til et problem kan være: 2 konsentriske sirkler er 10 cm unna hverandre, hva er differansen mellom omkretsen i disse sirklene?



I dette tilfellet kjenner vi ikke diameteren i oppgaven, og for å regne ut differanse mellom omkretsen i figurene må vi altså bestemme at diameteren for sirklene er $\boldsymbol{d}$ og $\boldsymbol{d+20}$. Forskjellen på omkretsen av sirklene er altså $\boldsymbol{\pi(d+20)-\pi d=20\pi}$. Hvis vi i stedet ser på et ekstremt tilfelle der vi krymper sirklene helt til den innerste sirkelen krymper til et punkt i sentrum av den andre sirkelen, ser vi at forskjellen mellom omkretsen i sirkelen er $\boldsymbol{20\pi}$. Her sparer vi oss mye arbeid ved å se på det ekstreme tilfellet.

2.4.2.6 Forenkle oppgaven

Å løse problemer handler ofte om å finne den beste og enkleste veien til en løsning. Noen ganger kan denne veien være å jobbe med en enklere versjon av problemet først, for å få innsikt som kan hjelpe deg å løse den vanskeligere oppgaven. Gitt problemet: Er summen av 1287 og 1385 et partall eller et oddetall, vil en løsning være å først jobbe med, «enerne», 5 og 7 og prøve å finne en løsning på det problemet, for deretter å bruke denne løsningen for å finne et svar på hele oppgaven. Gjennom å løse et enklere problem og

mestre steg for steg, kan eleven etter hvert mestre den komplekse helheten (Posamentier & Krulik, 2008).

2.4.2.7 Organisere data

En annen måte å gjøre problemet enklere på er å organisere dataen i problemet slik at den blir presentert på en mer oversiktlig og tydelig måte. Et eksempel kan være å finne medianen i en lang rekke prøvesvar. Hvis testscoren er «rotete» kan det være vanskelig å finne medianen.

72,43,98,57,87,89,67,23,56,89,91,88,72,75,66.

Hvis vi derimot organiserer tallene i stigende rekkefølge, vil det være mye enklere å finne medianen. Da vil tallet stå i midten.

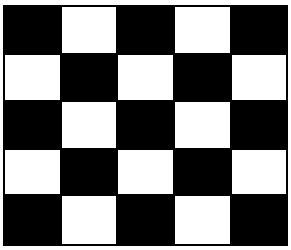
23,43,56,57,66,67,72,72,75,87,88,89,89,91,98.

Det å organisere data kan ofte lede til gjenkjennelse av egenskaper ved datamaterialet, og dermed kan det hjelpe de som arbeider med problemet til å se enkle løsninger (Posamentier & Krulik, 2008).

2.4.2.8 Lag en tegning eller ilsutrasjon

Det å lage en tegning eller en visuell representasjon av problemet er noe som kommer veldig naturlig når man jobber med geometriske problemer. I mange andre deler av matematikken kan det også være lønnsomt å visualisere problemet. Mange er visuelle i læringsprosessen og forstår lettere når noe blir tegnet (Posamentier & Krulik, 2008).

Et eksempel kan være oppgaven: I et klasserom står det 25 stoler stilt opp i et kvadrat på 5 rader med 5 stoler. Elevene som sitter på stolene får beskjed om at de skal flytte seg en stol til høyre, venstre, frem eller tilbake. Kan elevene gjøre det? Eventuelt hvorfor/hvorfor ikke? Her er det vanskelig og knotete å prøve seg frem til en løsning, men med en tegning blir det mye enklere. For å kunne flytte seg ser vi nå at elevene må flytte seg fra et svart sete til et hvitt sete eller motsatt. Siden det er 12 hvite og 13 svarte seter er det ikke mulig å gjennomføre.



2.4.2.9 Gjør rede for alle muligheter

Å gjøre rede for alle muligheter er en problemløsningsmetode som kan være effektiv i å løse komplekse problemer. Denne brukes ofte i sannsynlighet, men kan også brukes i andre sammenhenger. Denne måten å løse problemer på er ikke den mest sofistikerte måten, men kan noen ganger være den enkleste. Strategien krever som regel at det er mulig å organisere alle mulighetene i en tabell eller liknende (Posamentier & Krulik, 2008).

Et eksempel er følgende oppgave. Per inviterer sine 17 klassekamerater på besøk. Hver gjest får et nummer mellom 2 og 18 og blir bedt om å finne sammen i par. Etter at det er gjort oppdager Per at summen av alle parene er kvadrattall. Hvilket nummer er Per sin partner.

Denne oppgaven kan løses gjennom prøving og feiling eller den kan løses ved å liste opp alle muligheter. Ved å ramse opp alle nummer som gir kvadrattall kan vi raskt etablere hvilke par som må høre sammen.

1&3, 1&8, 1&15	6&3, 6&10	11&5, 11&14	16&9
2&7, 2&14	7&2, 7&9, 7&18	12&4, 12&13	17&8
3&1, 3&6, 3&13	8&1, 8&17	13&3, 13&12	18&7
4&5, 4&12	9&7, 9&16	14&2, 14&11	
5&4, 5&11	10&6, 10&15	15&1, 15&10	

Her ser vi at 3 av parene er bestemt allerede og fjerner vi 7,8 og 9 fra de andre mulighetene, vil vi raskt komme frem til det neste paret som bare har igjen 1 mulighet, nemlig 2&14. Slik fortsetter prosessen til vi raskt står igjen med parene 1&15, 2&14, 3&13, 4&12, 5&11 og 6&10. Per sin partner er altså nummer 15.

2.4.2.10 Intelligent gjett og sjekk

Denne teknikken blir ofte sett på som en prøve og feile metode, men det er en litt for enkel forklaring. Selve metoden er svært sofistikert, og det er stor forskjell på et gjett og et intelligent gjett. En som gjetter intelligent, bruker gjettingen for å smalne ned antallet muligheter og hvert gjett bygger på det forrige (Posamentier & Krulik, 2008).

Et eksempel kan være oppgaven: På en bankett ble det servert et fat med kylling for hver 2. gjest, et fat med ris for hver 3. gjest og et fat med grønsaker for hver 4. gjest som var til stede. Til sammen ble det servert 65 fat med mat. Hvor mange gjester var til stede? Denne oppgaven kan løses ved hjelp av en likning $\frac{x}{2}+\frac{x}{3}+\frac{x}{4}=65$ Her kan vi løse for x, og få at antallet gjester er 60.

Ikke alle elever og spesielt ikke yngre elever, vil ha nok kunnskap om ligninger til å løse denne. De kan i stedet forsøke å gjette og sjekke for å finne svaret. Vi vet ut ifra forutsetningene at antallet gjester er et multiplum av 12, siden det er felles for 2,3 og 4 fat som ble servert. Da kan vi prøve oss frem fra 12 og oppover

Antall gjester	Antall kylling fat	Antall ris fat	Antall grønsaks fat	Totalt antall fat
12	6	4	3	13 (ikke nok)
24	12	8	6	26(ikke nok)
48	24	16	12	52(nærme)
60	30	20	15	65(riktig)

På denne måten kan elever uten kunnskap om likninger løse problemet og komme frem til rett svar.

2.5 Algoritmisk tenking og Computational thinking

Når studien min handler om elever som løser matematiske oppgaver ved hjelp av programmering blir det viktig for meg ikke bare å lene meg på rammeverket for problemløsning i matematikk, men også se mot teori spesifikk for programmering. I programmering er begrepet *computational thinking* viktig. I Norge bruker vi algoritmisk tankegang om det engelske *computational thinking.* Dette kommer jeg tilbake til. Wing satte søkelyset på *computional thinking* i en artikkel fra 2006. I denne artikkelen argumenterer hun for at dersom vi jobber med å forstå programmering, vil vi oppnå en overføringsverdi til den virkelige verden. Vi vil bli bedre på å forstå problemer, og vi kan bli bedre problemløsere. Videre skriver hun i artikkelen at *«Computational thinking is a way humans solve problems, it is not trying to get humans to think like computers»* (Wing, 2006). Som vi vet er ikke dette en ny tanke, men en tanke som fikk mye oppmerksomhet rundt programmeringens første inntog i skolen og Papert sine publikasjoner fra 1980. Likevel er dette nå mer aktuelt enn noen gang med de fremskrittene som er gjort på den teknologiske fronten.

2.5.1 Computational thinking

I 2017, over et tiår etter at Wing skrev artikkelen som satte fokus på *computational thinking,* skrev hun en ny artikkel hvor hun beskriver hvordan fokuset på dette begrepet har påvirket forskning og undervisning på området. Her kommer hun med en presisering av hva *computational thinking* er:

> *Computational thinking is the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out (Wing, 2017).*

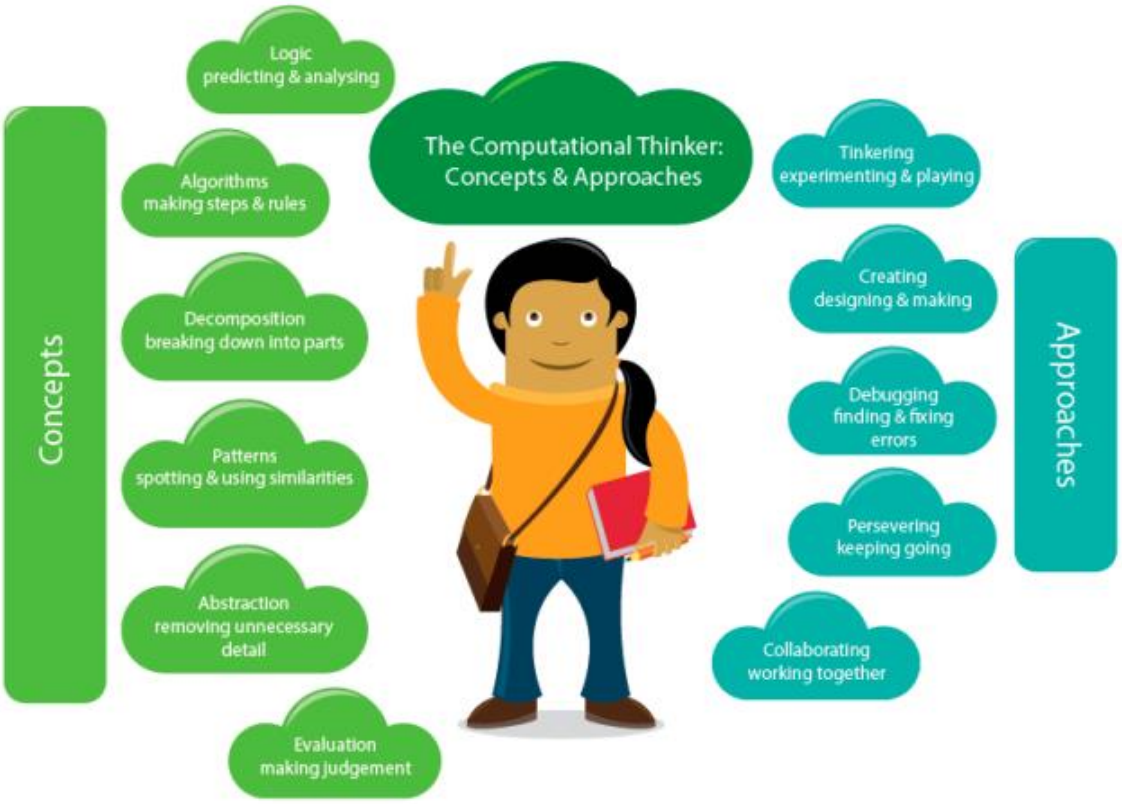
Skal vi følge Wings definisjon av *computational thinking*, så er det en problemløsningsmetode. Selv om Wing har sin definisjon av *computational thinking,* finnes det også her mange ulike tilnærminger til begrepet. Selv om de fleste tilnærminger innbefatter problemløsning, må det også nevnes at enkelte forskere mener at problemløsning faller utenfor begrepet. Et eksempel på dette er Selby & Woollard som i sin gjennomgang av begrepet *computational thinking* peker på at problemløsning faller under begrepet i deler av litteraturen, men at enkelte de selv inkludert, mener at problemløsning er et for vidt tema til at det kan underlegges *computational thinking* (Selby & Woollard, 2013).

Csizmadia et al har gjennom «*Computing at schoo*l» presentert et rammeverk for *computational thinking* i det engelske skoleverket. Her skriver de*:*

> *Computational thinking skills enable pupils to access parts of the Computing subject content. Importantly, they relate to thinking skills and problem solving across the whole curriculum and through life in general* (Csizmadia et al., 2015).

I min kontekst betyr dette at *computational thinking* og problemløsning henger tett sammen, og at vi kan se på begrepet som en problemløsningsmetode med tilhørende strategier når jeg skal vurdere om elevene arbeider problemløsende. At *computational thinking* er en problemløsningsmetode støttes også av Utdanningsdirektoratet i modellen de bruker. Dette ser jeg nærmere på i kap 2.5.2.

I artikkelen presenteres en oversikt over «*the computational thinker*» sammen med inngående beskrivelser av konseptene og arbeidsmåtene som brukes av *computational thinkers*. Figuren under viser en oversikt over disse.



Figur 6:Computational thinkers, et rammeverk for undervisning i *Computational thinking* (Csizmadia et al., 2015, s. 8)

Dette rammeverket er en guide for lærere i skolen, og har som mål å støtte lærere og skoler i undervisningen i det nye faget Computing. I rammeverket omtales *computational thinking* som en kognitiv prosess som hjelper eleven med å løse problemer, forstå prosedyrer og skape systemforståelse. Denne kognitive prosessen inkluderer

- Logisk tankegang
- Evnen til å tenke algoritmisk
- Evnen til å dekomponere
- Evnen til å generalisere og identifisere mønster
- Evnen til å abstrahere og velge gode representasjoner
- Evnen til å evaluere

Hver av disse konseptene kan identifiseres gjennom distinkt elevadferd i klasserommet. Med elevadferd menes det i denne sammenhengen det elevene sier og gjør som andre kan observere. I tillegg til en inngående forklaring av hvert enkelt konsept, følger også en liste av slike adferder eller læringsmønster lærerne kan se etter for å identifisere om eleven arbeider med *computational thinking* (Csizmadia et al., 2015). Denne modellen er identisk med modellen Utdanningsdirektoratet bruker i sin kompetansepakke for lærere i den norske skolen(Utdanningsdirektoratet, 2020b). Denne skriver jeg mer om i kapittel 2.5.2.

Videre vil jeg presentere en beskrivelse av konseptene eller strategiene og hvilke elevadferder som kan identifisere at elevene jobber med dette spesifikke konseptet. Disse beskrivelsene vil jeg videre benytte når jeg analyserer elevarbeidet i denne oppgaven, for å se om elevene bruker noen av disse konseptene når de jobber med programmering gjennom PRIMM.

2.5.1.1 Logisk tankegang

Logisk tankegang eller resonnering gjør elever i stand til å resonnere seg fra en antagelse, og kan gjennom en analyse forutsi og forstå hva som kommer til å skje. Her kan elevene bruke tidligere kunnskap og innlærte metoder til å verifisere antagelser, og å komme til en konklusjon. Dette konseptet blir i stor grad brukt gjennom testing, feilsøking og retting av algoritmer. Logisk tankegang brukes i stor grad sammen med de andre konseptene som algoritmisk tankegang, dekomponering, abstraksjon, generalisering og evaluering. (Csizmadia et al., 2015)

2.5.1.2 Algoritmer

Algoritmer, og det å tenke algoritmisk, er en måte å komme seg til en løsning gjennom en klar definering av stegene. Enkelte problemer har komponenter som kan brukes til å løse andre liknende problemer. Når slike situasjoner oppstår kan man bruke algoritmer for å finne en løsning som fungerer hver gang. Algoritmer for oppsett av multiplikasjonsoppgaver er et eksempel på en slik algoritme. I slike algoritmiske oppgaver lager man en strukturert plan, og finner løsninger gjennom steg for steg prosedyrer. Denne måten å forstå og løse problemer på er en kjerneferdighet som elevene utvikler gjennom programmering av dataprogrammer (Csizmadia et al., 2015).

I klasserommet er det mange elevaktiviteter som viser at elevene arbeider med dette konseptet. Det kan blant annet være om eleven:

- Formulerer instruksjoner for å oppnå ønsket effekt
- Formulerer instruksjoner som aritmetiske og logiske operasjoner
- Bruker variabler
- Bruker instruksjoner som tar valg
- Bruker gjentagelser, sløyfer eller løkker
- Bruker algoritmer for å teste en hypotese
- Bruker algoritmer for å beskrive virkelige prosesser
- Lager algoritmer som tar hensyn til hva eleven ønsker å bruke dem til.

2.5.1.3 Dekomposisjon

Dekomposisjon handler om å bryte ned komplekse problemstillinger til mindre delproblemer som er enklere å løse. Hver enkelt del kan bli forstått, løst, utviklet og

evaluert separat. En del av prosessen er også å vurdere hvilke delproblemer som kan løses av datamaskiner (Csizmadia et al., 2015).

I klasserommet kan man se etter om elevene:

- Bryter ned problemer til enklere bestanddeler for å kunne jobbe med hver del.
- Bryter ned problemet til en enklere versjon av det samme problemet som kan løses på samme måte

2.5.1.4 Generalisering av mønster

Generalisering er en måte å løse nye problemer basert på tidligere løsninger på liknende problemer. Dette innebærer å kunne identifisere, gjenkjenne og utforske ulike mønster og sammenhenger. Videre må eleven evne å benytte seg av elementer fra tidligere løsninger, som kan være til hjelp i det nåværende problemet (Csizmadia et al., 2015).

I klasserommet kan man se etter om eleven:

- Identifiserer mønster og fellestrekk
- Tilpasser løsninger eller deler av løsninger slik at de gjelder flere liknende problemer.
- Overfører ideer fra en løsning fra et problemområde til et annet.

2.5.1.5 Abstraksjon

Abstraksjon er en prosess hvor man avgrenser og skiller ut det som er viktig fra uviktige detaljer, slik at man kan fokusere på den delen som er viktig for å løse et spesifikt problem. Et eksempel kan være et T-bane kart. Der vil bare informasjon som er viktig for å planlegge en tur være synlig, mens alt annet er tatt vekk. En viktig ferdighet i abstraksjon er å velge de rette detaljene som skal skjules slik at problemet blir lettere å løse. En måte å ta med bare de viktigste detaljene er å velge gode representasjoner for et system eller problem (Csizmadia et al., 2015).

I klasserommet kommer dette til syne gjennom at eleven:

- Reduserer kompleksiteten ved å ta vekk unødvendige detaljer
- Velger en god representasjon som kan manipuleres
- Skjuler den fullstendige kompleksiteten
- Identifiserer forholdet mellom to abstraksjoner
- Filtrerer informasjon når de skal utvikle en løsning.

2.5.1.6 Evaluering

Evaluering er prosessen der man kontrollerer løsningen, og sørger for at løsningen er god. Et viktig poeng fra Csizmadia et al. (2015) i evalueringen er å vurdere om løsningen kan forbedres. I *computational thinking* er det et stort fokus på detaljer, og om det finnes noen måte å gjøre programkoden enda bedre (Csizmadia et al., 2015).

I klasserommet kan man se etter om elevene:

- Vurderer om løsningen er passende for formålet?
- Vurderer om løsningen gjør det den var tiltenkt å gjøre
- Eleven tester og vurderer ulike løsninger

- Vurderer om resultatet er godt nok, for eksempel effektivt nok
- Sammenlikner resultater for å se hvem som er best

2.5.2 Algoritmisk tenking

Algoritmisk tenking er den norske oversettelsen av *computational thinking,* og brukes av Utdanningsdirektoratet gjentatte ganger i begrunnelsen for innføring av programmering i skolen. De skriver at undervisning i programmering ikke bare handler om at elevene skal forstå datamaskiner og kunne bruke digitale hjelpemidler, men også om å gi eleven gode verktøy for å løse problemer og oppgaver. Videre sier Utdanningsdirektoratet at de *«har valgt å oversette det engelske computational thinking med algoritmisk tankegang»* og at det er en problemløsningsprosess som innebærer å bryte ned store komplekse problemer, organisere og analysere data på en logisk måte, lage algoritmer, abstraksjoner og modeller samt å generalisere løsninger (Utdanningsdirektoratet, 2018).

Utdanningsdirektoratet skriver selv at de bygger på Barefoot Computing (Computing at school) sin modell, og definisjonen til Wing når de presenterer algoritmisk tankegang inn i skolen i Norge. På sine nettsider og i kompetansepakkene laget for lærere i skolen skriver Utdanningsdirektoratet:

> *Algoritmisk tenkning er en problemløsningsmetode. Algoritmisk tenkning innebærer å tilnærme seg problemer på en systematisk måte, både når vi formulerer hva det er vi ønsker å løse og når vi foreslår mulige løsninger. Litt forenklet kan vi si at det er 'å tenke som en informatiker' når vi skal løse problemer eller oppgaver (Utdanningsdirektoratet, 2019b).*

Det å tenke algoritmisk er å vurdere hvilke steg som skal til for å løse et problem, samt å kunne formulere problemet slik at datamaskiner kan løse hele eller deler av problemet. Algoritmisk tenking innebærer å bryte ned komplekse problemer i delproblemer, organisere og analysere informasjon, å kunne abstrahere og lage modeller av den virkelige verden, fjerne unødvendige detaljer og å kunne generalisere en løsning (Utdanningsdirektoratet, 2019a).

Figur 7: Den algoritmiske tenkeren slik Utdanningsdirektoratet bruker den i sine kompetansepakker og hjemmesider

Hvis man sammenlikner denne figuren (Figur 7) med (Figur 6) «*The computational thinker*» i kap 2.5.1, ser man at den norske figuren er kopiert ord for ord fra den engelske. For min studie betyr det at jeg kommer til å ta utgangspunkt i de originale beskrivelsene og forklaringene fra *computational thinking,* da de er mer inngående og går enda mer i dybden på hvilke tegn man kan se etter hos elevene for å avdekke arbeide med algoritmisk tenking.

2.6 Scratch

Scratch er et gratis visuelt programmeringsspråk. Språket er utviklet av Massachusetts Institute of Technology (MIT) og har som mål å lære barn i alderen 8-16 år å programmere (Scratch, u.å.). Språket er et såkalt blokkbasert programmeringsspråk som kan gjøre inngangen inn i programmering for barn enklere. Gjennom forskningen på 80 og 90 tallet som utforsket de mulige fordelene av å lære seg programmering, kom det også frem at det var betydelige utfordringer knyttet til begynneropplæringen i programmering og blant annet syntaksen i programmeringsspråk. På dette området virker det som om blokkbaserte programmeringsspråk som Scratch, med sine fargekodede blokker og blokker som passer sammen på bestemte måter, gjør noen avanserte konsepter i programmering mer tilgjengelig (Resnick, 2012b).

Målet med Scratch var at det skulle bli enket for alle å programmere og dele egne fortellinger, spill, animasjoner og simuleringer. Gjennom å utvikle Scratch ønsket utviklerne å senke terskelen for å programmere så langt ned at hvem som helst i teorien kunne kode. Prinsippene i Scratch var at programmeringsspråket skulle være mer

utforskende, meningsfylt og sosialt enn andre programmeringsspråk (Resnick et al., 2009). Gjennom å tilby et programmeringsspråk som var lett tilgjengelig håpet utviklerne av Scratch å nå unge barn og å gi dem et verktøy i for læring, kreativitet, mestring og deling. Når man lærer å lese kan man bruke lesing for å lære, på samme måte når man lærer å kode kan man kode for å lære (Resnick, 2012a).

De ulike brikkene i Scratch hjelper brukeren til å identifisere hva hver del av koden gjør gjennom fargekoding. Hver kategori har sin fargekode, og alle brikkene i den fargen hører til den samme kategorien.

Category	Notes	Category	Notes
Motion	Movements of sprites like angles and position	Sensing	Sprites can interact with the surroundings
Looks	Controls the visuals of the sprite	Operators	Mathematical operators, comparisons
Sound	Plays audio files and effects	Variables	Variable and List usage and assignment
Events	Event handlers	My Blocks	Allows defining functions which have no return value
Control	Conditionals and loops etc.	Extensions	Explained below

Figur 8: Fargekoding i Scratch (Wikipedia contributors, 2121)

3 Metode

Som tidligere nevnt ønsker jeg i denne studien å se på elever som jobber med programmering gjennom PRIMM metoden. Målet er å finne ut om elevene arbeider problemløsende når de arbeider med programmering på denne måten. Det er vanskelig å vite hvordan elevene tenker underveis i oppgaveløsningen, men ved å observere, analysere og tolke elevenes arbeid, kan man likevel danne seg et visst inntrykk av hvordan de tenker. På denne måten kan jeg prøve å identifisere ulike trekk som tyder på at de arbeider problemløsende. I dette kapittelet vil jeg presentere mitt teoretiske perspektiv, mitt metodevalg og det undervisningsopplegget og de oppgavene jeg velger å gi elevene. Jeg vil også presentere hvordan jeg har valgt å analysere datamaterialet, samt undersøkelsens relabilitet og validitet.

Innledningsvis i arbeidet mitt med denne masteroppgaven hadde jeg et ønske om å se på programmering i matematikkfaget. Etter å ha satt meg inn i de mange diskusjonene rundt innføringen av programmering i skolen, som til slutt ga matematikkfaget ansvaret for opplæringen i programmering, bestemte jeg meg for å se om programmeringsundervisning kunne implementeres på matematikkfagets premisser. Jeg identifiserte at det fantes mange ressurser for arbeid med programmering, men få pedagogiske verktøy for lærerne å støtte seg på i utarbeidingen av undervisningsopplegg. I arbeidet med å finne et pedagogisk rammeverk som kan støtte lærere i undervisning, leste jeg det pågående forskningsarbeidet med en modell kalt PRIMM i England. Forskningen på emnet så lovende ut, og jeg ønsket derfor å se om undervisning ved hjelp av PRIMM kunne hjelpe elevene med å arbeide undersøkende og problemløsende med programmering og matematikk på samme tid slik som Utdanningsdirektoratet hevder. «*Når elevene bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse*» (Utdanningsdirektoratet, 2020a, s. 1). På denne måten kan kanskje programmeringsundervisningen bedrives på matematikkens premisser.

3.1 Teoretisk perspektiv

Det finnes flere ulike epistemologiske utgangspunkt eller ståsted en forsker kan ha. Postholm og Jacobsen (2018) sier at det epistemologiske ståstedet handler om hvordan man får kunnskap om virkelighet. I denne forskningen forsøker jeg å finne ut noe om sammenhengen mellom programmering og problemløsende matematikk. Min tilnærming til denne oppgaven kan sies å være preget av et post-positivistisk syn. Jeg mener at det finnes en virkelighet som gjelder på tvers av personer og kontekster. Jeg mener at om man studerer og forsker på en type opplegg kan man, om ikke bestemme om, så sannsynliggjøre om denne typen pedagogisk opplegg påvirker elevene til å lære matematikk bedre. For at en slik sannsynliggjøring skal kunne finne sted må det forskes på tvers av elever, klasser og land over tid. Så selv om jeg i min case-studie kanskje bare kan finne en sannhet som gjelder lokalt, vil den i sammenheng med annen forskning kunne

bidra til å finne en sannhet som gjelder på tvers av alle klasser, nasjonalt eller globalt (Postholm & Jacobsen, 2018).

Et eksempel på slik forskning er forskningen rundt en type pedagogiske opplegg som «undersøkende matematikk», som viser at elevene ser ut til å lære matematikk bedre gjennom denne undersøkende formen for matematikkundervisning. Dette funnet ser ut til å til å være gyldig på tvers av elever, klasser og land (Wæge, 2007). Til tross for en slik sammenheng, kan man ikke si at et slikt pedagogisk opplegg vil ha nøyaktig den samme effekten i alle sammenhenger, men det vil kunne øke sannsynligheten for at resultatet vil bli det samme (Postholm & Jacobsen, 2018).

I et post-positivistisk ideal skal man først skape en forventning om hvordan virkeligheten ser ut. Denne forventningen bygger på tidligere teori og empiriske funn (Postholm & Jacobsen, 2018). I og med at Utdanningsdirektoratet legger til grunn at bruken av programmering til å utforske og løse problemer, kan bidra til å utvikle matematisk forståelse, var dette utgangspunktet for min oppgave. Når jeg arbeidet videre fant jeg en litteraturgjennomgang av programmering i skolen av Dolonen et al. (2019). I denne gjennomgangen peker Dolonen et al. (2019) på et behov for en type pedagogisk verktøy for å støtte elevene i programmeringen, og som skiller seg fra de tradisjonelle undervisningsmetodene. Dolonen et al. (2019) peker på PRIMM-modellen som en start mot en forskningsbasert undervisning i programmering. Videre fant jeg teori fra Sentance et al. (2019a, 2019b), som brukte PRIMM som en pedagogisk måte å arbeide med programmering på (Sentance et al., 2019a, 2019b). Jeg går inn i forskningen med en hypotese om at denne didaktiske modellen bidrar til at elevene arbeider undersøkende og problemløsende i programmering i matematikk. Denne hypotesen var også med på å forme forskningsspørsmålet mitt, som skulle hjelpe meg å svare på problemstillingen i oppgaven. Forskningsspørsmålet ble:

Hvilke problemløsningsstrategier kommer til syne når elevene jobber etter PRIMM-modellen?

3.2 Forskningsdesign

Når man forsker er det vanlig å skille mellom en kvalitativ og en kvantitativ tilnærming til forskningen. I en kvantitativ metode er fokuset ofte å få informasjon om en virkelighet gjennom tall, som kan bearbeides i statistiske analyser. I en kvalitativ metode derimot, vil det ofte hentes informasjon om virkeligheten gjennom språk og ord. Informasjonen hentes gjerne inn gjennom nedskrivning av hva folk sier eller forskerens observasjon (Postholm & Jacobsen, 2018).

For å velge riktig forskningsdesign er det viktig å huske på hva som er målet med forskningen. I og med at jeg ønsker å finne ut om programmering kunne tilrettelegge for problemløsende aktivitet, setter dette noen rammer for hvilke forskningsdesign jeg bør velge. For å kunne studere nærmere hva som skjer med en gruppe elever som blir utsatt for en type undervisning, PRIMM, ble det naturlig å velge en kvalitativ tilnærming til

datainnsamlingen og forskningen. Jeg kan da følge elevene i en begrenset tidsperiode, og se hvordan de arbeider med innholdet og oppgavene i en økt som er lagt opp etter noen spesielle forutsetninger og regler. For å finne ut om elever som arbeider etter et PRIMM undervisningsopplegg arbeider problemløsende i programmering, ble det derfor naturlig å betrakte gruppen elever som en case og utføre en case-studie om dette.

3.3 Oppgaven

I arbeidet med å forske på PRIMM metoden i England har forskerne utviklet et undervisningsopplegg som går over 10 uker med en økt på ca. 60 minutter hver uke. I dette opplegget er øktene lagt opp på en måte som følger prinsippene i en PRIMM oppgave. For å være sikker på at min oppgave også følger alle prinsippene i PRIMM valgte jeg å hente mitt undervisningsopplegg og oppgave til elevene, rett ut ifra dette ferdiglagde opplegget. Den første utfordringen jeg fikk i arbeidet med å oversette undervisningsopplegget til norsk, var at oppgavene i det engelske undervisningsopplegget var tilpasset tekstspråket Python. I samråd med lærerne på barneskolen der jeg gjennomførte min datainnsamling, tok vi en avgjørelse om at elevene var best tjent med å programmere i et blokkbasert programmeringsspråk, da det var dette elevene skulle møte i pensum i løpet av året. Jeg måtte derfor i tillegg til å oversette oppgaven til norsk, skrive om koden fra Python-kode til det blokkbaserte programmeringsspråket Scratch, med Scratch-tillegget «Penn». Jeg oversatte oppgaven ved at jeg, så nøyaktig som mulig, prøvde å replisere alle linjene i Python-koden med Scratch blokker. Deretter gikk jeg gjennom de ulike oppgavene elevene skulle ha, samt det tilhørende støttematerialet og PowerPoint presentasjonen, og tilpasset og oversatte disse til norsk.

Activity Sheet 4b: Loops and turtles

Before you start download the starter program for this

```
def square():
    for counter in range(4):
        forward(100)
        right(90)


square()
left(45)
square()
```

Figur 10: Orginal kode

Aktivitetsark B – Løkker.

Før du starter åpne linken til programmet under fra
bruker.

Figur 9: Oversatt kode

Dette er et eksempel på oversettelsen. Resten av oppgavene, planleggingsdokumentet og PowerPointen ligger som vedlegg til slutt i oppgaven. Se vedlegg Vedlegg 1: Undervisningsplan, Vedlegg 2: PowerPoint for undervisnings økt, Vedlegg 3: Aktivitetsark AVedlegg 4: Aktivitetsark B – Løkker.

Undervisningsopplegget som ble brukt i den endelige PRIMM studien var godt gjennomarbeidet gjennom en førtest og pilotstudie av Sentance et al. (2019a). Undervisningsøkta jeg valgte å bruke i min datainnsamling, har altså allerede vært igjennom 3 runder med testing (Sentance et al., 2019b). Økta jeg valgte ut var økt nummer 4 i en rekke av 10 økter på ca. 1 times varighet. Grunnen til at jeg valgte denne var at de første øktene i undersøkelsen bar sterkt preg av å være introduksjon til det tekstbaserte språket Python. I og med at blokkene i Scratch er mer selvforklarende, mente jeg at elevene ville være i stand til å kunne hoppe rett inn i økt 4.

3.4 Pilotundersøkelse

I forkant av datainnsamlingen min valgte jeg å gjennomføre en pilotundersøkelse for å være sikker på at opplegget lot seg gjennomføre i praksis. En av tilbakemeldingene til forskingsprosjektet som ble gjennomført i England var at hver økt var litt for lang, sammenliknet med tiden som var satt av (Sentance et al., 2019a). Jeg valgte derfor å utvide tiden for økta fra 60 til 90 minutter. Gjennom pilotundersøkelsen identifiserte jeg noen punkter som jeg ønsket å rette på før selve gjennomføringen av datainnsamlingen. For det første trengte elevene en bruker på Scratch for å kunne laste ned og endre den delte programkoden. I tillegg fant jeg ut at den siste oppgaven i originalopplegget var et frempek til neste undervisningsøkt, noe det ikke var behov for da jeg ikke skulle gjennomføre flere økter. Disse to punktene justerte jeg før gjennomføringen av undervisningsopplegget og datainnsamlingen.

3.5 Utvalg av elever

Mitt utvalg av elever er i en viss grad preget av pragmatiske hensyn. Likevel har jeg innenfor de rammene jeg har hatt, gjort et strategisk utvalg av elever til min datainnsamling. All forskning preges av begrensede ressurser, og hva slags metoder og analyser man benytter seg av bestemmes derfor ikke bare av faglige hensyn, men også i noen grad av praktiske hensyn (Thagaard, 2018; Tjora, 2017). I mitt tilfelle bestod de praktiske hensynene i at jeg er i 100% jobb som lærer på en barneskole ved siden av å gjennomføre denne forskningen. Dette har både fordeler og ulemper. Jeg har på min skole hatt god tilgang på klasserom og elevgrupper. Jeg har derfor hatt gode muligheter til å gjennomføre det faglige opplegget, og å gjøre observasjoner. Samtidig gjør det at jeg er i jobb det mer krevende å få gjennomført den samme undersøkelsen med elevgrupper på andre skoler. Rent praktisk betydde dette at det var lett for meg å skaffe elever til en studie, men mer utfordrende å skaffe nok elever til for eksempel å sette opp en kontrollgruppe, og å studere forskjeller over tid.

En annen faktor ved utvelgelsen av elever til undersøkelsen, var å få levert samtykkeskjemaene fra foreldrene i tide. Siden en del elever ikke leverte skjemaene i tide, valgte jeg å sette sammen tre nye grupper av 18 elever med elever fra tre ulike klasser. Elevene i undersøkelsen gikk alle i 6.klasse. To av disse sammensatte gruppene ble filmet og observert, mens den tredje gruppa gjennomførte økta uten å være en del av studien.

Dette medførte at elevene måtte jobbe på tvers av sine vante klasser, så for at elevene skulle være så trygge som mulig, valgte jeg så langt det lot seg gjøre å beholde grupper som allerede jobbet sammen fra de ulike klassene, når jeg skulle sette sammen grupper til prosjektet.

Gruppene som var en del av undersøkelsen, besto altså av 36 elever. Den strategiske delen av utvalget kom når jeg skulle velge hvilke skoletrinn jeg ønsket å samle data fra, samt hvilke elever i de observerte gruppene som skulle være i fokus for kameraene og lydopptagerne i klasserommet. Det er flere måter å velge ut disse elevene på, men fordi jeg hadde kjennskap til elevene fra før, kunne jeg gjøre visse strategiske utvalg for å begrense datamengden noe. Siden jeg kjente elevene visste jeg også hvilke elever som hadde meget høyt mestringsnivå i matematikk, og hvilke som hadde lavt mestringsnivå. Gruppene som ble valgt ut til å ha lydopptager og kamera rettet mot seg hadde derfor en sammensetning av elever på ulike nivå som samarbeidet godt i matematikk. Jeg vet av erfaring at ingen av elevene i utvalget er på et spesielt høyt eller lavt nivå (de har ikke tilpasset matematikk for høyere eller lavere trinn) Det var to lydopptagere og ett videokamera i hver gruppe. I tillegg var det ett oversiktskamera. På hver gruppe var 9 elever som ble tatt opp på film eller lyd, altså 18 stykker til sammen. I tillegg har jeg observert og notert ned enkelte hendelser og uttalelser fra andre grupper der opptaksutstyret ikke var til stede. Dette utvalget var strategisk, og basert på informasjon jeg hadde om utvalget som ble undersøkt (Thagaard, 2018). Som en konsekvens av utvalget, ser jeg på hvordan faglig gjennomsnittlige elever arbeider med i mitt datamateriale. I en større undersøkelse kunne man sett på alle elevene og undersøkt om det var forskjeller på hva slags utbytte faglig svake og sterke elever får av det samme opplegget. Dette er noe jeg i min studie ikke hadde ressurser til å gjennomføre.

3.6 Observasjon

I denne studien ser jeg etter om elevene arbeider problemløsende i en bestemt undervisningsstruktur. For å finne ut av dette trenger jeg å observere elevene igjennom en slik økt. Observasjon som datainnsamlingsstrategi blir sett på som den mest grunnleggende måten å samle inn data på (Postholm & Jacobsen, 2018). Ifølge Tjora er det liten tvil om at observasjonsstudier er noe av det mest krevende og samtidig mest potente en kan utsette seg selv for som forsker (Tjora, 2017). Observasjon er mer enn å bare se på hva som foregår. Observasjon er en måte å skaffe data fra en situasjon direkte, slik det skjer, istedenfor gjennom annenhånds informasjon fra de som har vært i situasjonen, eller som forteller om hva de selv har gjort. Observasjon kan være av fakta, hendelser, noens adferd, kvaliteter og valg av strategier (Cohen et al., 2011). I denne undersøkelsen ønsker jeg å observere og sette søkelys på elevenes valg av strategier. Det å velge tid og sted for observasjon kan være utfordrende, og ikke minst og få muligheten til å observere på en måte som gjør at man kan få tilgang til observasjonsdata som er relevant for forskningsspørsmålet. Målet må være å observere på en måte som ikke er med å påvirke utfallet av dataen som skal benyttes i etterkant (Tjora, 2017).

I mitt tilfelle kunne jeg selv styre tid og sted for undervisningen i emnet, og på den måten også styre tid og sted for observasjon. For de 36 elevene som deltok i undersøkelsen ble økta som en hvilken som helst annen time på ukeplanen, med det unntaket av det var videokameraer og lydopptagere til stede i rommet. Jeg ønsket å observere elevene i så naturlige omgivelser som mulig, derfor fikk alle elevene de samme oppgavene, alle satt i samme rom og arbeidet på de samme gruppene som de vanligvis gjorde i denne perioden. Ettersom observasjonen ble gjort på denne måten kan vi si at elevene ble observert i sin naturlige setting (Cohen et al., 2011). Video og lydopptak utgjør hovedtyngden av min observasjon, da jeg ved hjelp av opptakene kunne gå tilbake og observere funn som jeg ikke fikk med meg i øyeblikket. Likevel utgjorde observasjon underveis i undervisningen også en del av materialet. Her valgte jeg metoden for ustrukturert observasjon. I en ustrukturert observasjon er man åpen for alle løsninger elevene kan ha, og alle innfallsvinkler de kan komme med (Cohen et al., 2011). Underveis tok jeg bilder og notater i situasjoner der jeg observerte ulike måter elevene løste oppgavene på. Dette kunne jeg senere legge sammen med videoopptak for å få et bedre bilde av hva elevene gjorde, og hvorfor de valgte disse strategiene.

3.7 Min rolle som lærer og observatør

I litteraturen beskrives et skille mellom ulike roller en observatør kan ha. Gold beskriver en observatør som fullstendig deltager, observerende deltager, deltagende observatør eller fullstendig observatør (Gold, 1958). Senere har andre som (Tjora 2017) og Postholm (2017), tatt tak i de samme kategoriene og beskrevet de videre med sine forslag til endringer. Viktigst for meg i denne sammenhengen er Postholms (2017, s.64) beskrivelse av rollen fullstendig deltager. Fullstendige deltagere er personer som tilhører settingen hvor det forskes på i utgangspunktet, eller som blir fullstendige deltagere i løpet av prosessen (Postholm, 2017; Tjora, 2017). Dette omfatter meg som lærer som underviser i egne klasser og klasserom. Min rolle underveis i undersøkelsen var primært å være lærer for klassen. Selv om noen grupper ble filmet og tatt opp på lyd, var fokuset mitt å undervise alle elevene i det samme materialet uavhengig av dette. På denne måten fikk elevene en genuin opplevelse av hvordan det var å bli undervist i programmering ved hjelp av modellen PRIMM. Selv om undervisning var hovedfokuset i øktene, hadde jeg undersøkelsen i bakhodet hele veien. Når grupper som ikke ble filmet kom opp med en løsning som var unik for klassa, dokumentere jeg dette ved å ta bilde av skjermen og å samle inn digital og fysisk kode etter økta. Postholm snakker også om hvordan det er en fordel å notere underveis i observasjonen, men anerkjenner at dette er vanskelig når man er en fullstendig deltager i det som foregår (Postholm, 2017). Dette er en av grunnene til at jeg valgte video og lyd-opptak som en del av datainnsamlingen for lettere å kunne se tilbake på timen og notere ting som foregikk i etterkant.

3.8 Video og lyd opptak

Dersom forskeren ønsker å få med seg alt som blir sagt i en gruppe eller på steder han ikke er til stede til enhver tid, kan en lydopptager eller et videokamera være til god hjelp (Postholm, 2017). En slik bruk av videoutstyr kan også hjelpe forskeren med og fokusere på funn som han vanligvis ikke ville fokusert på, for eksempel funn som ikke ofte gjentar seg, eller som lett havner i bakgrunnen av det som tar fokus (Cohen et al., 2011). For meg ble kamera og lydopptager til uvurderlig hjelp i innsamlingen av data. Jeg hadde ikke klart å samle inn og få med meg mer enn en brøkdel av det som ble sagt på de ulike gruppene, om det ikke hadde vært for disse hjelpemidlene. Samtidig må vi huske at bruken av kamera og opptaksutstyr kan være fremmedelementer som virker forstyrrende på elever som blir observert. Kameraene har også den begrensningen at de bare kan fokusere på en ting av gangen (Cohen et al., 2011). I mitt konkrete tilfelle forsøkte jeg som foreslått av Cohen et al. (2011), å løse disse utfordringene ved å sette opp et panoramakamera som observerte hele klasserommet, mens jeg hadde enkeltkameraer og lydopptagere som tok opp situasjonene på hver enkelt gruppe. Når det kommer til kameraene som fremmedelementer i klasserommet kan det etter gjennomgang av dataene, virke som om de ble raskt glemt. Elevene oppførte seg på samme måte som de ville gjort ellers, og er trygge i sine vante omgivelser i klasserommet.

3.9 Å se etter tegn til problemløsning

For å kunne avgjøre om elevene arbeider problemløsende i programmeringsøkten laget ved hjelp av PRIMM, må jeg sammenlikne med ett allerede etablert rammeverk for problembasert læring. Som beskrevet i teoridelen, bestemte jeg meg for å vurdere elevene ut ifra to ulike rammeverk for problemløsning. Både Posamentier & Krulik (2015) sine problemløsningsstrategier, og den algebraiske tenkeren fra Csizmadia et al.(2015). Her er målet å se elevene både i lys av et matematisk problemløsningsrammeverk og ett rammeverk med røtter innfor informatikken. Det er verdt å merke seg at Utdanningsdirektoratet selv velger å støtte seg på rammeverket fra informatikk når de begrunner innføringen av programmering i matematikken (Csizmadia et al., 2015; Posamentier & Krulik, 2015; Utdanningsdirektoratet, 2019a).

Etter observasjonen satt jeg igjen med et stort datamatriale som måtte bearbeides for å kunne vurdere hvilke problemløsningsmetoder elevene brukte i arbeidet. For å sortere materialet ble alt transkribert før det ble kodet ved hjelpt av Nvivo som er et databehandligsverktøy. Det ble opprettet tre hovedkategorier av kode, etter de tre hovedområdene av teori som jeg observerte elevene i lys av. Disse var PRIMM av sentence et al. (2019a; 2019b), matematisk problemløsning fra Posamentier & Krulik (2015) sine 10 problemløsningsstrategier og *computational thinking* fra Csizmadia (2015) sine konsepter i *computational thinking*. Hver hovedkategori ble delt opp i underkategoriene etter sine respektive teorier. Dette kommer jeg nærmere inn på i analysedelen (kap 4.)

Etter at all innsamlet data var kodet kunne jeg sammenfatte det hele, og se på hvilke problemløsningsstrategier som kom frem i hvert av stegene i PRIMM. Alle deler av

datamaterialet ble kodet til en av de 5 kategoriene i PRIMM. Deretter så jeg på hvilke koder fra de to andre hovedkategoriene som kom sammen med PRIMM. I mange tilfeller ble materialet kodet med både underkategorier fra matematisk problemløsning og *computational thinking*. Jeg har også sortert resultatet i en tabell for å visualisere i hvilke steg og hvilke strategier kom til syne.

Når det gjelder transkripsjonen av data, er den transkribert så ordrett og lydrett som mulig fra det elevene sa. Jeg har brukt noen ulike grep for å markere hendelser som ikke kommer frem i klartekst i transkripsjonen. Jeg bruker et punktum for å markere kort pause og tre punktum for å markere en lengre pause i snakkingen til en elev. Hvis det blir et betydlig opphold på 3-4 sekunder starter jeg på ny linje. For å få frem hva eleven gjør fysisk, der jeg har feltnotater eller video som viser dette, bruker jeg paranteser.

3.10 Relabiltet og validitet i oppgaven

I kvalitativ og kvantitativ forskning vil relabilitet, validitet og generaliserbarhet være indikatorer på forskningens kvalitet (Tjora, 2017). Når vi ser på kvaliteten i kvalitative studier spesifikt, kan man snakke om undersøkelsens pålitelighet, troverdighet, overførbarhet og bekreftbarhet (Johannessen et al., 2016).

I utgangspunktet vil en case-studie kunne gi lokal kunnskap om hvordan noe fungerer på denne ene skolen. Vi kan si at den interne gyldigheten og validiteten av denne undersøkelsen er stor (Postholm & Jacobsen, 2018). For min case-studie betyr det at funnene jeg gjør har stor intern gyldighet og vil gjelde for akkurat den skolen og det klassetrinnet hvor undersøkelsen er gjort. Overførbarheten av studien avhenger derimot av hvor godt studien er beskrevet, og om det er mulig å gjenskape den samme studien med de samme resultatene i en annen kontekst (Postholm & Jacobsen, 2018). I forskningsprosjektet mitt er undervisningsopplegget som blir brukt hentet fra Sentance et al. (2019b) sin studie, og oversatt undervisningsopplegg ligger vedlagt i sin helhet. Teoriene jeg bygger på og prosessen for datainnsamlingen er nøye beskrevet. Likevel er det faktorer som gjør det vanskelig å replisere studien med hundre prosent nøyaktighet. Læreren som underviser vil være en annen, og erfaring og forutsetning for å gjennomføre opplegget vil være en annen en min. Elevenes forkunnskaper og forutinntatthet for programmering vil også variere. Vi kan derfor si at de viktigste delene av prosjektet har en overførbarhet og ytre gyldighet, men det vil alltid være faktorer utenfor forskerens kontroll som spiller inn i en slik overføring. Min studie kan være med å bidra til å belyse forskningsfeltet, og hvis det blir gjennomført flere liknende studier kan disse sett i sammenheng vise om funnene er gyldige i flere kontekster, og dermed også ha større validitet i feltet.

4 Analyse

Analysen har som mål og sette lys på de delene av undervisningen og elevarbeidet som kan bidra til å svare på problemstillingen min: **På hvilken måte kan programmering tilrettelegge for problemløsende aktivitet i matematikk?**

I analysen ser jeg helt spesifikt etter problemløsningsstrategier som kommer frem hos elevene for å svare på forskningsspørsmålet:

Hvilke problemløsningsstrategier kommer til syne når elevene jobber etter PRIMM-modellen?

Oppgavene som er gitt til elevene er hentet fra PRIMMs oppgavebank med tilhørende PowerPoint og undervisningsplan for undervisningsøkta. Hver side i PowerPoint-presentasjonen som guider læreren og elevene gjennom økta er merket med symboler som forteller i hvilket steg i PRIMM modellen akkurat denne aktiviteten hører til. Analysen er strukturert i kronologisk rekkefølge etter kategoriene fra PRIMM-modellen. Under hver kategori tar jeg for meg hvilke problemløsningsstrategier fra matematikk og hvilke konsepter fra algoritmisk tankegang som kommer til syne hos elevene i arbeidet og diskursen. I prosessen med å analysere data ble det derfor opprettet 3 hovedkategorier fra teorigrunnlaget. I disse hovedkategoriene er det plassert 21 underkategorier. Koding av alt materiale, både lyd, video, bilder og oppgaveark ble lagt inn i Nvivo og kodet her.

De tre hovedkategoriene er

1) PRIMM
2) Matematisk problemløsning
3) *Computational thinking*

Hovedkategorien PRIMM inneholder de fem underkategoriene *Predict, Run, Investigate, Modify* og *Make*. Kategoriene kommer fra Sentance et al., (2019a,2019b) og er de samme som kategoriene i PRIMM. Koding av matematisk problemløsning ble delt inn i underkategoriene etter Posamentier & Krulik (2015) sine 10 problemløsningsstrategier. Underkategoriene ble da: Logisk resonnement, se etter mønster, jobbe bakover, se problemet fra andre synsvinkler, vurdere ekstremaltilfeller, forenkle oppgaven, organisere data, lage tegning eller illustrasjon, gjøre rede for alle muligheter og intelligent gjett og sjekk etter. Den siste hovedgruppen *computational thinking* ble delt i undergruppene etter Csizmadia (2015) sine *computational thinking* konsepter, logisk tankegang, algoritmer, dekomposisjon, generalisering, abstraksjon og evaluering. I flere tilfeller fant jeg at elevenes arbeid kunne kodes med både matematisk problemløsning og *computational thinking* kategorier. Dette viser at arbeidet til elevene ofte faller inn under både problemløsning i matematikk og informatikk, dette konkretiserer jeg senere i analysen.

En oversikt over alle kategoriene, og hvilke problemløsningsprosesser jeg fant i mitt datamateriale vil bli presentert først, deretter vil utdrag fra hver av kategoriene i PRIMM presenteres og analyseres. I oversikten viser jeg en oppsummering av funnene fra datamaterialet. Her vises hvilke problemløsningsstrategier jeg fant igjen hos elevene i de respektive steg. *Predict og run* er slått sammen fordi det i selve steget *run* ikke oppstår noen egne problemløsningsstrategier. Her får elevene se om de har tenkt riktig i predict-fasen og en visning av hvordan en type program oppfører seg når det kjøres.

	Predict Run	*Investigate*	*Modify*	*Make*	PRIMM
Logisk resonnement	X	X	X	X	X
Se etter mønster	X	X	X	X	X
Jobbe bakover			X		X
Se fra andre synsvinkler	-	-	-	-	-
Vurdere ekstremaltilfeller			X		X
Forenkle oppgaven				X	X
Organisere data	-	-	-	-	-
Tegning eller illustrasjon	X	X	X		X
Gjøre rede for alle muligheter	-	-	-	-	-
Intelligent gjett og sjekk	X		X		X
Logisk tankegang	X	X	X	X	X
Algoritmer			X	X	X
Dekomposisjon	X	X	X	X	X
Generalisering av mønster			X	X	X
Abstraksjon			X	X	x
Evaluering		X	X	X	X

Undervisningen jeg nå skal analysere materiale fra, besto av to deler. Del 1 handlet om en introduksjon og arbeid med «gå, snu og tegn» kommandoene. Del 2 besto av en introduksjon og arbeid med «gjenta» kommandoen. Begge delene av økta fulgte PRIMM oppbyggingen. Del 1 besto av de 4 første stegene med unntak av *make.* Del 2 inneholdt alle 5 steg. I selve analysen har jeg valgt å slå sammen alle funn og sortere de etter stegene i PRIMM slik at jeg kan sammenlikne hva elevene har gjort i hvert av stegene. En fullstendig oversikt over undervisningen og PowerPoint brukt i undervisning ligger ved i Vedlegg 1: Undervisningsplan og Vedlegg 2: PowerPoint for undervisnings økt.

4.1 Predict og run

I opplegget som ble gjennomført med elevene er det to seksjoner som er tenkt å tilhøre det første steget, *predict*, i PRIMM modellen. I *predict*-delen av modellen skriver Sentance et al. at elevene skal lese, diskutere og vurdere kode for å forsøke å forstå hva som er kodens funksjon. Begge gangene en kode ble presentert for elevene i *predict*-fasen fikk de koden oppgitt i PowerPoint, og fikk i oppgave å diskutere i gruppene hva de mente koden

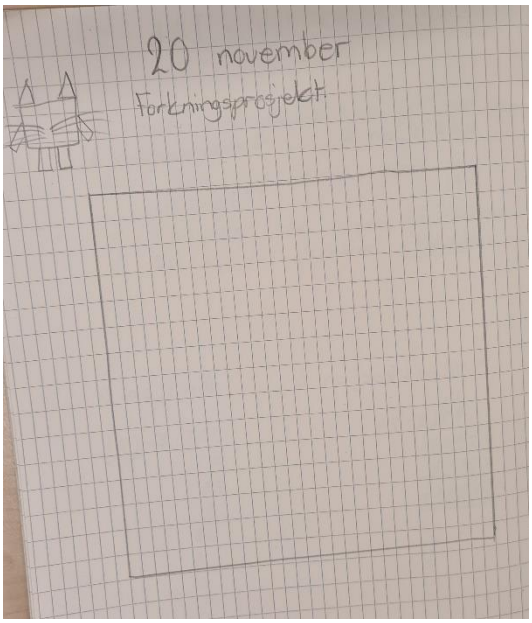
ville gjøre. Etter diskusjonen fikk elevene mulighet til å kjøre koden på egen Ipad i tillegg til at koden ble kjørt felles på smartboarden. Her følger et utdrag av samtaler og tegninger elevene brukte for å forklare hva de mente koden ville gjøre.

Figur 11: Predict del 1

Elev A1:	Nå skal vi tegn hva som kommer til å skje
Elev A2	Ja, men først skal vi snakke om det
Elev A1:	Men det blir jo veldig rart hvis den snur, (Gestikulerer frem og tilbake) går 100 steg den veien, snur og går 100 steg den veien
Elev A2:	Den kommer til å ta salto
Elev A1:	Snu seg først, så legger seg, så gå 100 steg fremover, så still seg på hode.
Elev A3:	Eller så bare går den frem og tilbake
Elev A2:	Ja sånn gå, snu, gå snu, gå snu
Elev A1:	90 grader snu, hvis den ser den veien så snur den jo ikke den jo ikke hele veien
Elev A3:	Hva er det vi tror da? At den legger seg ned på bakken?
Elev A1:	Tenk dere hvis dere ser for dere den katta, så går den og så snur den 90 grader
Elev A2:	Men hva skal vi tegne da (Elev A1)
Elev A1:	Jeg må tegne katta, eller jeg bare tegner en firkant, det her er katta

I den første *predict*-oppgaven var mange av elevene usikre på hva eller hvordan de skulle angripe problemet. Av de seks gruppene som ble tatt opp på lyd eller film var det ingen som umiddelbart identifiserte hva koden ville gjøre.

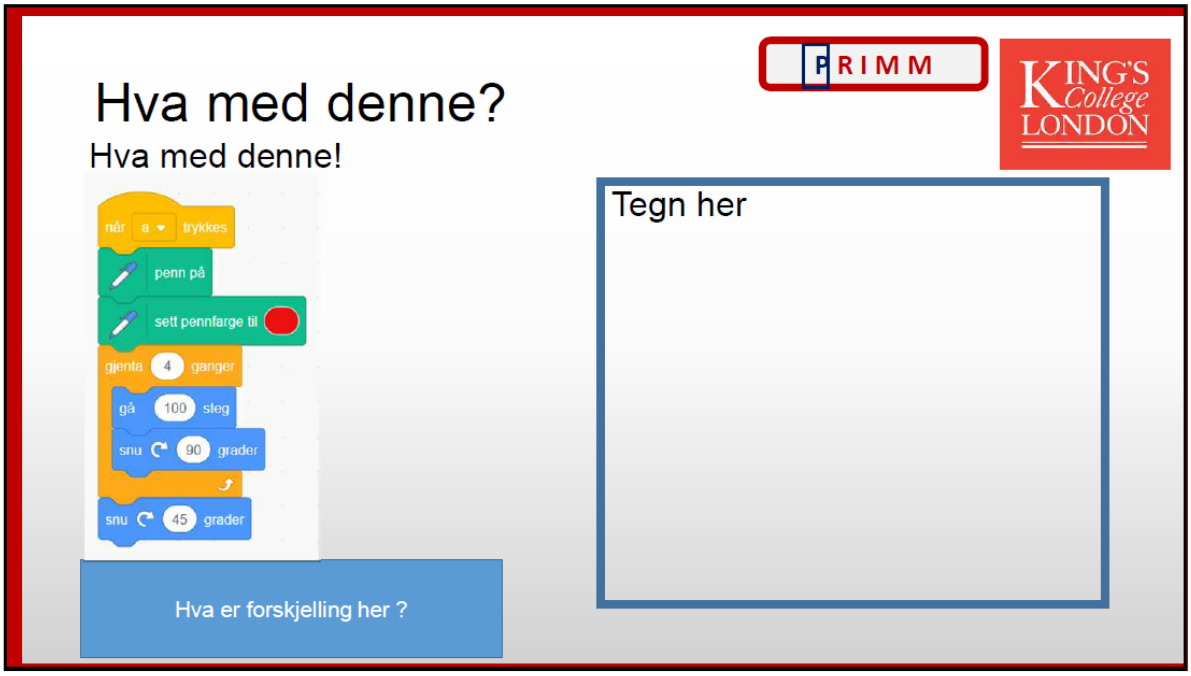
Elevene i eksemplet over prøver seg frem med **logisk tenking og dekomposisjon** for å forklare hva koden vil gjøre. De velger å ta for seg bevegelsesdelen av koden og ser på «gå og snu» kommandoene og prøver å beskrive hva «katten» i koden gjør. Elevene klarer ikke umiddelbart å bli enige, men er innom flere ulike konsepter som bevegelse, rotasjon, antall grader og avstand.



Figur 12: Kvadrat tegnet fra 1. kode Elevgruppe A

Gjennom **logisk tankegang** diskuterer elevene om «katten» (figuren som utfører handlinger i Scratch) i programmet, vil komme til å bevege seg samt å snu retningen den kommer til å gå. Fra tegningen ser vi også at elevene på denne gruppen har kommet frem til at figuren som tegnes blir en firkant. Posamentier & Krulik peker på at logisk resonement er en av de vanligste strategiene som ofte blir brukt i dagliglivet. Gjenom logiske argumenter klarer elevene å argumentere for det som stemmer (Posamentier & Krulik, 2008). I dette tilfellet er det en av elevene (Elev A1) som forstår hva koden kommer til å gjøre. Likevel sliter hen med å forklare for gruppa hva som kommer til å skje, og til slutt gir hen opp den muntlige forklaringen og tegner det i stedet. Hen bruker da **Lag en tegning eller ilustrasjon** fra Posamentier & Krulik (2015). Gjennom å bruke denne metoden kan hen visualisere for de andre på på gruppa, og få støtte i forklaringen av koden. Sentance et al. (2019a;2019b) skriver i sine publikasjoner om språket som et medierende verktøy, og hvor viktig det er for elevene som jobber med programmering. Et viktig poeng med *predict, run, investigate* fasen i PRIMM modellen er å øve elevene i å bruke språket og det nye vokabularet som følger med, slik at de kan bli sikre på dette (Sentance et al., 2019b). Elevene er her helt i startfasen av å programmere og har ikke lært seg dette vokabularet, noe som byr på utfordringer i samhandlingen og samarbeidet.

Dekomponering er en av de seks konseptene i *computational thinking* modellen til Csizmadia et al. De beskriver at elever som bruker dette konseptet i en klasseromssituasjon, vil bryte ned problemet i enklere bestanddeler for lettere å kunne jobbe med hver del (Csizmadia et al., 2015). I denne første oppgaven var dekomponering noe elevene tidlig i arbeidsprosessen benyttet seg av. I det første kodeutklippet elevene fikk se, fokuserte elevene i alle seks gruppene på en del av kodeutklippet av gangen. I gruppen i eksemplet over, fokuserte elevene på de blå brikkene i programmet først. Selv om elevene ikke ble ferdige, kunne de si noe om denne delen av koden. Grunnen til at elevene valgte å hoppe hit kan tilskrives at dette er en mer interessant del av koden, og den delen som ikke umiddelbart ble forstått av elevene som leste koden.

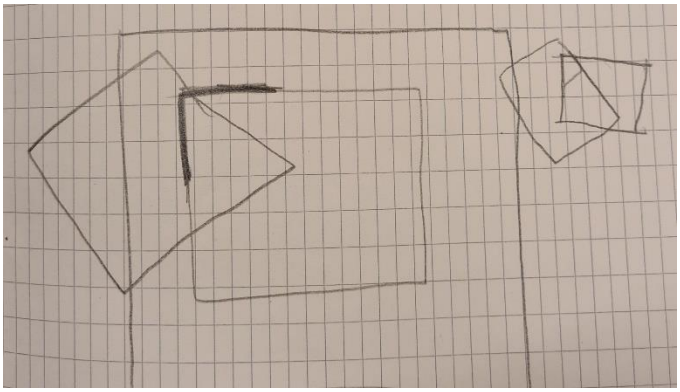


Figur 13: Predict del 2

Andre halvdel av undervisningen startet med en ny *predict* øvelse. Denne delen av undervisningen handlet om gjentagelser og løkker. Koden over ble presentert for elevene og de fikk i oppgave å utforske koden for å finne ut hva den ville gjøre.

Elev A1:	Rundt den «gå hundre og snu 90» så står det gjenta 4 ganger så det blir akkurat som i stad
Elev A2:	Ja da blir det jo fremdeles en firkant da,
Elev A1:	Ja det blir det samme bare at den står en annen vei når den er ferdig (rekker opp hånda og spør læreren)
Elev A1:	Det blir jo bare en firkant og så står den en annen vei når den er ferdig
Lærer:	Hva kommer da til å skje om du starter koden igjen da?
Elev A1:	Da blir det jo sånn her, (tegner en firkant, med en ny firkant 45 grader oppå)

Av datamaterialet kommer det frem at fem av seks grupper valgte å **dekomponere** problemet for å konsentrere seg om en del av koden først. I eksemplet over anvendte gruppa **logisk tankegang** til å forutsi hva som kom til å skje i koden basert på tidligere kunnskap. De identifiserte raskt at koden kom til å oppføre seg på samme måte som koden de startet med i den første kodedelen, og skape en firkant. Gjennom annen tidligere

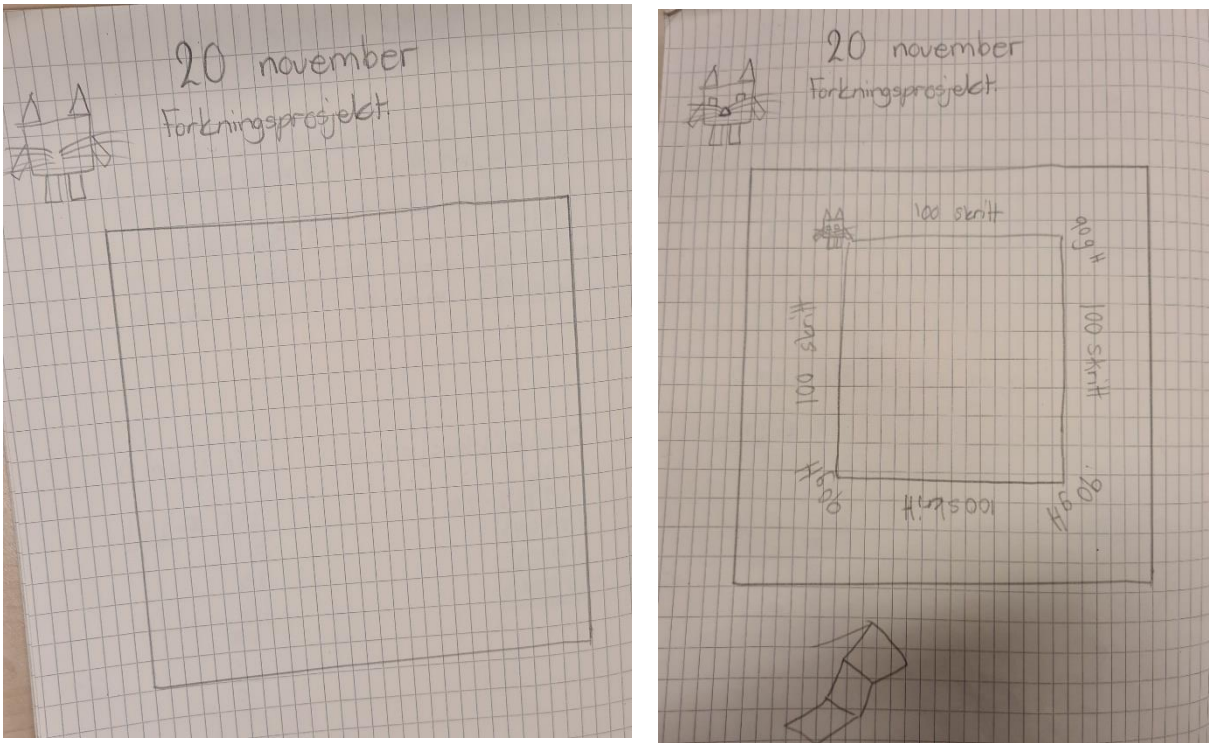


Figur 14:Kvadrat fra 2. Kode, Elevgruppe A

erfaring fra *investigate* og *modify* stegene forsto elevene i gruppa også at en siste kodebrikke, snu 45 grader, ville gjøre at neste firkant la seg på skrå oppå den forrige. Csizmadia et al og Posamentier & Krulik peker begge på at hvis elevene har tidligere erfaring med liknende oppgaver kan de resonnere seg frem til hva som kommer til å skje, basert på hva som skjedde forrige gang de møtte samme type problem (Posamentier & Krulik, 2008; Sentance et al., 2019b).

Se etter mønster er en problemløsningsstrategi som innebærer å gjenkjenne et mønster, og bruke dette til å løse problemet (Posamentier & Krulik, 2008). Gjennom å gjenkjenne mønsteret i denne oppgaven klarte elevene å identifisere at koden når den blir brutt ned i sine bestanddeler, er lik den første koden de arbeidet med. Elevene kjente igjen paret med kodebrikker «gå 100 og snu 90» fra koden tidligere på dagen, og selv om elevene ikke hadde noen erfaring med kodebrikken «gjenta 4» forsto Elev A1 at koden skulle gjentas 4 ganger, og dermed ble lik som koden hvor paret med kodebrikker sto under hverandre 4 ganger.

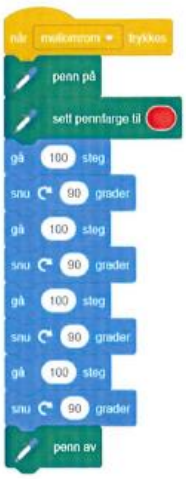
En annen matematisk problemløsningsstrategi som ble brukt av samtlige elever i begge *predict* delene av økta var **«lag en tegning eller illustrasjon».** Posamentier & Krulik skriver at denne strategien er noe som ofte kommer naturlig til elevene når de jobber med geometriske problemer (Posamentier & Krulik, 2015). Som vi ser over, illustrerte elevene sine forståelser av programmet, slik at de kunne ha noe å støtte seg på når de skulle forklare til sine medelever eller til læreren. Selv om fokuset for elevene i økta var å lære seg programmering, er også geometrien sentral for elevene. De benytter seg av den matematiske problemløsningsstrategien for å representere ikke bare hva som skjer med koden, men også hvilken geometrisk figur som kommer frem. At elevene valgte denne strategien er noe aktiviteten i PRIMM naturlig legger opp til, da elevene enda ikke har mulighet til å kjøre koden og se resultatet fysisk. Derfor er naturlig å tegne dette for hånd. Dette blir enda tydeligere videre når elevene selv modifiserer koden til å bli andre geometriske figurer. Etter at elevene eller læreren kjørte koden (*run*) var det også mange elever som valgte å rette opp, eller notere videre på illustrasjonene sine. Et eksempel på dette er tegningen under. Her forteller eleven selv at hen ønsket å notere hva koden gjorde for hvert steg slik at hen kunne huske det til senere. I tillegg til å falle inn under kategorien «lag en tegning eller illustrasjon» er dette også starten på elevens undersøkelse av kode som er kjørt, altså neste steget i PRIMM, *investigate*.



Figur 15: Kvadrat fra 1. Kode, før og etter *Run*-steget Elevgruppe A

4.2 Investigate

Det tredje steget i PRIMM-modellen er *investigate*. I dette steget skal læreren stille spørsmål som utfordrer elevenes kodeforståelse. Spørsmålene skal hjelpe elevene å utvikle forståelse for ulike nivåer av koden (Sentance et al., 2019b). I opplegget elevene gjennomførte er det markert med investigate to steder. Det vil si at det er to steder gjennom opplegget hvor det legges opp til at elevene skal undersøke kode i henhold til det tredje steget i PRIMM. Første gang var i forbindelse med den første koden elevene fikk presentert, mens andre gang var når de lærte om løkker. Eksempler fra begge steder følger under. Etter å ha kjørt den første koden fikk elevene utdelt ett oppgaveark som skulle støtte dem i å undersøke koden nærmere. Eksempler på oppgaver, svar og diskusjoner følger under:



Oppgave 1: Svar på følgende spørsmål:

1. I operasjonen Snu (høyre) 90, hva tror du 90 betyr?
   90 betyr at man skal snu seg 90 g i den retningen komandoen gis.
2. Hva gjøre gå (100)?
   A gå 100 er å gå 100 steg fram, bak, eller til sidene.
3. Hva skjer om du flytter penn av til tidligere i koden?
   Hvis du flytter (pen av) tiligere stoppes komandoene tiligere.
4. Hva skjer om du endrer alle (høyre 90) Til (høyre 45)? Svar først så prøv og se hva som skjer.
   Jeg trur det blir et rektangel. Svaret ble en Halv 8 kant.

I denne gruppa diskuterte de oppgavene og det var ulike meninger om hva koden kom til å gjøre. Her følger et utdrag fra diskusjonen rundt oppgave 4. Ett av poengene Sentance et al. (2019b) ønsker å få frem fra elevene i denne fasen, er utvikling av forståelse for de ulike delene av koden sammen med utvikling av et vokabular knyttet til programmering.

Elev A2:	Hva skjer om du endrer alle 90 til 45 i stedet
Elev A1:	Hmm, da må vi se på arket i koden, hvis vi endrer alle disse til 45
Elev A1:	Da fær han bare halveis, i stedet for at det blir en firkant blir det bare en halv 4kant (tegner først en hel firkant og så en halv firkant)
Elev A2:	Ja, men da blir det jo en sånn…
Elev A1:	Neeiii, vent
Elev A2:	Da blir det jo en sånn her firkant. (tegner et rektangel)
Elev A1:	Vent litt, blir det ikke bare en mindre firkant da? For hvis det her er 90 grader så snur man jo så mye, viser 90 grader på papiret
Elev A2:	Vi har fortsatt 100 steg da så den er like stor
Elev A1:	Ja så da blir det sånn da eller? Tegner et rektangel på skrå (tiltet 45 grader)

Elev A1:	(mumler) Eller blir det mer sånn, (tegner en trekant).
Elev A2:	Nei jeg tror det blir et rektangel
Elev A1:	Jeg tror det blir et rektangel, ok så da må vi prøv det
Elev A2:	Hæ? Skal vi lage vår egen nå da? (Til læreren som passerer)
Lærer:	Du kan bare endre den koden som jeg gav deg.
Elev A1:	(Har endret alle vinklene i koden.) Ok da sjekker jeg hva som skjer. *OI! Det blir en halv 8 kant!*
Elev A1:	Oi det var jo veldig overaskende da.
Elev A2:	Det vart en åttekant
Elev A1:	Da veit vi iallfall hva 45 grader er. (tegner på arket) Hvis den står her så går den 100 steg, så snur den 45 grader, så går den 100 steg, og så…
Elev A2:	Det gir jo faktisk mening da.
Elev A1:	Selv om det er litt vanskelig å vite hva 45 grader er nøyaktig
Elev A3:	Men det er veldig smart da.
Elev A2:	Da må vi gjøre om her da (peker på svaret fra forrige oppgave hvor de skrev rektangel)
Elev A2:	Svaret var en halv åttekant.

I denne oppgaven blir elevene ledet igjennom en undersøkelse av ulike deler av koden. Her jobber elevene med en og en del av en forholdsvis enkel kode og forsøker å identifisere hva de ulike kodebrikkene og parameterne gjør. Elevene jobber altså her med **dekomposisjon** som metode slik som vi finner hos Csizmadia et al. (2015). Ved å dekomponere koden kan de ta for seg en del av gangen, når de forstår hver del for seg kan de også lettere forstå helheten.

Oppgaven er her lagt opp slik at den spør elevene spørsmål som «hva hvis, svar så test» dette gjør at elevene må **evaluere** det de vet om koden, og hvordan den fungerer, for senere å bestemme hva de tror skjer videre. I *Investigate* har elevene også fått utdelt koden. Etter å ha svart på spørsmålene kan de kjøre koden og se om de fikk rett. Deretter kan de evaluere og eventuelt justere svaret og forståelsen. I oppgaven skal de først forsøke å undersøke koden og hva som skjer om en del av den endres. Deretter blir de bedt om å evaluere løsningen for å se om den var rett. I dette tilfellet var evalueringssteget viktig for elevene i og med at deres ulike initiale antagelse var feil. Gjennom å evaluere fikk elevene rettet opp i feilen og fikk en opplevelse av noe som «gav mening». Hos Csizmadia et al. Er evaluering en viktig prosess, her får elevene kontrollert løsningen, og vi ser at eleven tester og vurderer ulike løsninger slik som det beskrives i modellen deres (Csizmadia et al., 2015). Her får elevene også viktig kunnskap om koden kom kan benyttes videre i påfølgende steg i PRIMM modellen.

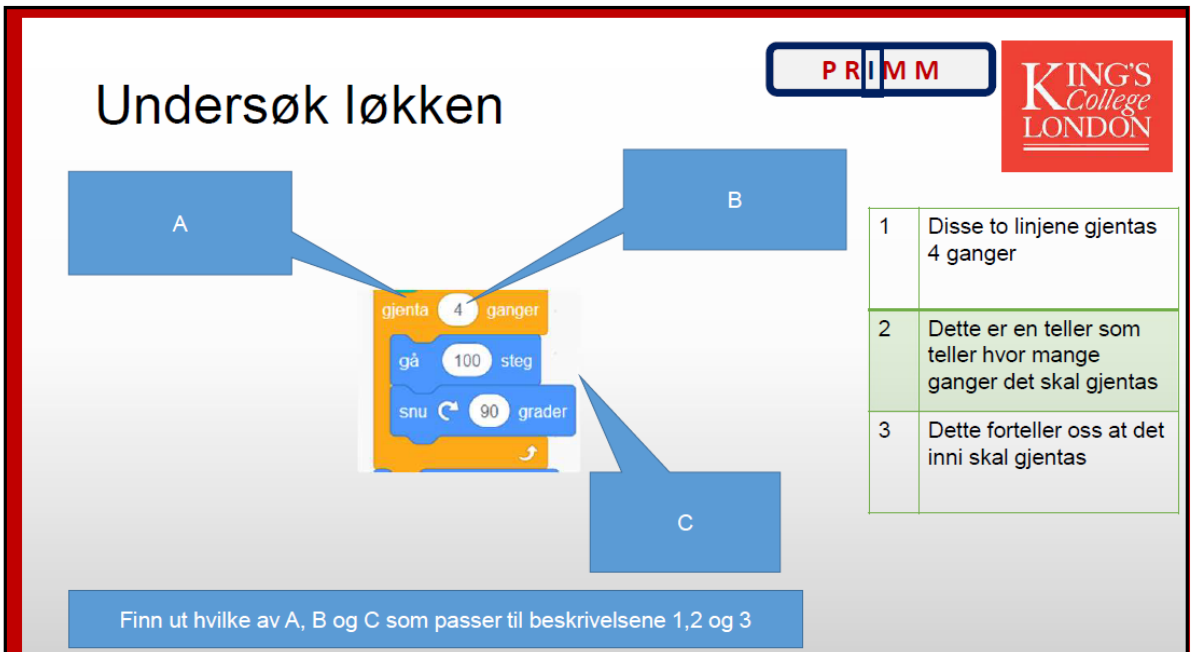
En annen gruppe diskuterer spørsmål 3 på arket.

Elev C1:	Her står det, hva skjer hvis du flytter pennen til tidligere i koden?
Elev C2:	Da skrur den seg på litt tidligere.
Elev C1:	Nei
Elev C2:	Penn av, nei åja, da skrur pennen seg av tidligere.
Elev C1:	Ja kanskje det, ja det tror jeg, men flytter? Det står jo flytter der.
Elev C2:	Ja flytter penn av, da gjør den jo det der vi putter den (kommandoen), Da skrur penna seg av

Elev C1:	Ja åja da visker den ut firkanten.
Elev C3:	Ja
Elev C2:	Nei den bare stopper å lage firkanten der vi putter den inn
Elev C1:	Men vi kan jo bare prøve da
Elev C2:	Ja hvis jeg drar den brikken midt inne i her sånn (har flyttet kodebrikken inn etter snu 90 andre gang) så blir det bare halve firkanten så da stopper den å tegne der.
Elev C3:	Åja, men vi tok den her (rett etter penn på) og da skjer det ingen ting når vi starter den.
Elev C2:	Det er jo fordi vi skrur den på og så av med en gang.
Elev C1:	Hvis vi putter den et annet sted da, hva skjer da (prøver etter første gå 100). Å ja, så uansett hvor vi putter den så blir det stopp der.

Ikke alle gruppene ble enige om svarene på de ulike oppgavene, men verdien av å diskutere med andre elever i læringsprosessen er tydelig i mange av utsagnene til elevene. I eksempelet over misforstår to av elevene (Elev C1 og C3), og mener at koden vil viske ut hele figuren basert på erfaringen de gjorde seg når de flyttet på brikken. I samarbeid med den siste eleven oppdaget de at dette kun var tilfellet for en enkelt plassering av kodebrikken i koden. Her er poenget at elevene forstår ulike deler av koding, og de kan støtte hverandre og løse problemer de ellers ikke ville kunne løst på egenhånd. Her er både kodeverktøyet og de andre elevene viktige medierende artefakter for hverandre i kodeforståelsen. PRIMM modellen bygger på den sosiokulturelle læringsteorien, og et viktig prinsipp i *investigate*-steget er at elevene ideelt sett skal diskutere med medelevene sine i denne fasen. Da vil elevene også etter hvert utvikle et vokabular å bruke i programmeringsdiskusjonene (Sentance et al., 2019b).

I den andre delen av økta hvor elevene lærte om løkker hadde elevene en tilsvarende oppgave, men denne gangen skulle de diskutere i gruppene om hvilke elementer i en kode som gjorde hva. Selve oppgaven ble vist frem på PowerPoint (Figur 16) og elevene svarte muntlig etter en kort gruppediskusjon.



Figur 16: Oppgave: undersøk løkken

Elev D1:	Disse to linjene gjentas 4 ganger
Elev D2:	Det må bli alternativ b
Elev D1:	Nei det må bli c, disse to linjene gjentas 4 ganger, siden c peker på begge linjene, jeg tror iallfall det, for de linjene er jo de samme som vi jobbet med i stad. Og c peker på de 2 linjene i koden så da tror jeg de gjentas sånn som når vi lagde firkanten.
Elev D2:	Dette er en teller som teller hvor mange ganger noe skal gjentas, det må vel være b?
Elev D1:	Æææææ ja
Elev D2:	Da må det bli 3a og 2b
Elev D2:	Ja det gir jo mening da! B) Dette er en teller som teller hvor mange ganger noe skal gjentas, A) den sier at det som kommer etter er det som skal gjentas, mens C er de 2 linjene som blir gjentatt

I denne *identify*-delen av økta har elevene jobbet, og fått flere erfaringer med bevegelseskodene. Det som er nytt for elevene her er «gjenta» blokken. Alle gruppene som ble tatt opp på lyd fullførte denne oppgaven, og gjennom diskusjon klarte de å sette alle beskrivelsene på rett plass relativt raskt. Her bruker de tidligere erfaring, og bygger på den med et **logisk resonnement** for å forutse hva som kommer til å skje, basert på tidligere liknende opplevelser, slik som både Csizmadia et al og Posamentier & Krulik beskriver (Csizmadia et al., 2015; Posamentier & Krulik, 2008).

4.3 Modify

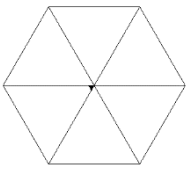
Modify er det fjerde steget i PRIMM modellen. Her er målet at elevene gjennom å endre litt og litt på koden, gradvis vil få eierskap til den. Frem til nå har elevene arbeidet med enkeltkonsepter i koden, og opparbeidet seg en forståelse av disse. Nå skal elevene bruke den forståelsen for å endre koden steg for steg slik at koden blir elevenes kode (Sentance et al., 2019b).

Modify-delen av økta startet for elevene etter den første runden med «*predict, run, investigate*». De skulle nå bruke koden de hadde fått utdelt tidligere, og endre denne i henhold til nye oppgaver. Blant gruppene som ble filmet og tatt opp på lyd var de flere variasjoner i hvordan de modifiserte koden. Videre følger oppgavene, og et utdrag av noen samtaler og koder fra denne delen av økta:

Oppgave 2 : Lag flere former

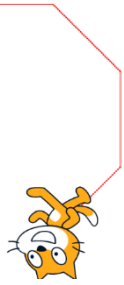
Prøv å gjøre disse oppgavene – Forklaring til noen kodebrikker du kan bruke er i tabellen nederst.

1. Endre den opprinnelige koden du fikk utdelt
   a. Endre fargen på linjene
   b. Endre lengden på linjene
2. Lag en ny kode der du tegner en trekant, tips: Du må endre vinkelen på snu kommandoen.
3. Bruk penn opp og penn ned for å tegne en firkant og trekant ved siden av hverandre.
4. Tegn figuren til høyre:

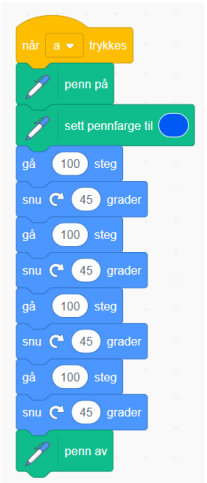


Lærer: Først skal dere prøve å endre fargen på linjene, så skal dere endre lengden på linjene og til slutt skal dere endre figuren til å bli en trekant

Elev C3: Jeg skriver bare 45 her (i kodebrikken snu) så blir den sånn (Figur 18: Halv åttekant)



Figur 17: Halv åttekant



Figur 18: Kode for halv åttekant

Elev C1: Du må ha gjort noe feil her

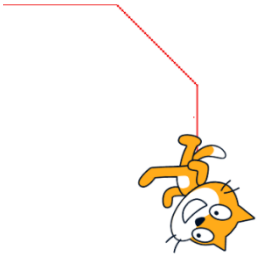
Elev C3: Oi nei, men det må kanskje ikke være 4 ganger da hvis det skal være trekant. Sånn kanskje, skal vi prøve nå

Elev C1: Ja, det blir nok bedre

Elev C3: Nei, det blir fortsatt sånn (del av åttekant: Siden antall og grader fremdeles står på 45)

Elev C1: Hvis vi tar bort en sånn gå 100 da?

Elev C3: Da ble jo den ene streka bare borte da (figur 19)



Figur 19: 3/8 åttekant

Elev C2: Men nå lager den jo bare 3 linjer da, så det er jo riktig hvis det skal bli en trekant!

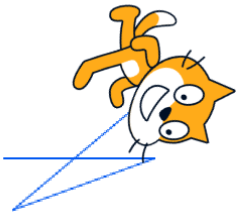
Elev C3: Da må vi fiks på de greiene her da (antallet grader)

Elev C1: Den må være 100 og … doble, hvis vi dobler 90 da?

Elev C3: Ja det var jo 90 i firkanten, ja eller prøver 160 nå jeg

Elev C1: Jaaa eller, ja den er jo trekanta da

Elev C3: Den henger jo ikke sammen da, da er jo ikke antall grader riktig da (Figur 20)



Figur 20: 160°

Elev C2: Ja det andre er jo riktig da! For den her er lengden på sidene, og det her er fargen, så da må det jo være gradene som skal endres

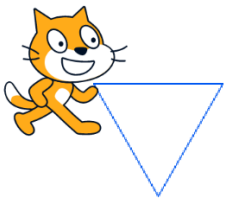
Elev C1: Vi prøver 120 da, det tror jeg kanskje kan funk

Elev C3: Ja de funka! Lærer se vi fikk det til! (Figur 21)

Lærer: Se her ja, dette funka bra! Hva er det som gjør at det fungerer da?

Elev C1: 40 + 40 + 40 er jo 120 da, jeg veit ikke, men det er dele på 3 iallfall

Lærer:	Ok Hva med i en firkant da? I koden vi brukte i stad var det jo snu 90 grader i hvert hjørne. Altså 4 ganger. Har det noen sammenheng.
Elev C1:	90 + 90 + 90 + 90 det er … regner ut ... 360 da.
Lærer:	Ok har dere hørt om 360 grader før? Hva er det?
Elev C2:	Det er helt rundt!
Elev C1:	Å ja, så det er helt rundt firkanten da?
Elev C2:	Nei, jeg tenkte på sirkel jeg.
Elev C1:	Ja, men det er jo det i firkanten og da!
Lærer:	Begge deler stemmer! Så hva med trekanten da?
Elev C1:	Da er vel det 360 og da?
Elev C2:	Ja, 360 dele på 3 er jo 120!!
Elev C2:	Vent jeg må skrive det i koden min også, snu 120…. ja det funker jo! Se trekant



Figur 21: Trekant

Elevene har frem til nå forholdt seg til ganske tette rammer og spesifikke oppgaver. I *modify*-oppgaven har elevene et klart mål de skal nå, men de må benytte seg av kunnskap og problemløsningsstrategier de har lært frem til nå for å løse problemet. De ulike gruppene gikk løs på denne oppgaven på ulike måter. Den første gruppen (se utdraget over) forsøkte først å gjøre om fra 90 til 45 grader, det samme som de gjorde i forrige oppgave. Når de så kjørte denne koden, tegnet programmet en illustrasjon (Figur 17) og elevene oppdaget at de hadde laget en halv åttekant i stedet for trekant. Her fungerer Scratch som et medierende artefakt for elevene, slik Vygotsky skriver om. Gjennom diskusjon hjelper elevene hverandre til å løse oppgaven, og ved hjelp av samarbeid og det visuelle kodeverktøyet Scratch klarte elevene å komme frem til et svar (Vygotsky, 1981).

I dette tekstutdraget var det vanskelig å kode elevaktiviteten med en spesifikk problemløsningsstrategi. Gjennom å teste koden gjentatte ganger skaper elevene **illustrasjoner** av hver utgave av programkoden som elevene lager. Samtidig er metoden en **Intelligent gjett og sjekk** metode, hvor elevene prøver seg frem med ulike antall grader og sikter seg inn på målet etter hvert som de kommer gradvis nærmere. I tillegg bruker elevene **logisk resonnering** både i prosessen hvor de finner ut hvor mange ganger koden skal gjentas, og etter koden er blitt riktig, for å forstå hvorfor koden stemte. Posamentier & Krulik (2008) understreker at det er sjelden elevene bruker bare en problemløsningsstrategi, men at elevene tar i bruk flere strategier.

I modellen til Csizmadia et al., *The computational thinker*, kan vi si at elevene bruker **abstraksjon** i kombinasjon med **logisk tankegang** for å løse oppgaven (Csizmadia et al., 2015). De elliminerer vekk all informasjon som ikke er viktig for oppgaven og fokuser på en linje med kode som gjentar seg. Her manipulerer de denne frem til de finner rett svar. Logisk tankegangkonseptet gjentar seg gjennom stort sett alle elevenes besvarelser. Dette kommer frem gjennom at elevene sjelden arbeider med bare logisk tankegang, men gjør dette i sammenheng med de 5 andre konseptene, (algoritmisk tankegang, dekomponering, abstraksjon, generalisering og evaluering) til Csizmadia et al (2015).

I det neste utdraget forsøker elevene å løse oppgaven ved hjelp av strategien **Jobbe bakover** fra Posamentier & Krulik (2008). I denne strategien starter elevene med svaret og jobber seg bakover til utgangspunktet. I dette tilfellet vet elevene at svaret skal bli en trekant, de vet hvordan en trekant tegnes for hånd, og de begynner altså med å tegne opp svaret. Her benytter de seg også av problemløsningsstrategien **Lag en tegning eller ilustrasjon**. Gjennom å starte med svaret kan eleven bruke ilustrasjonen til å bestemme at de bare trenger tre gjentagleser av koden. De kunne også finne frem en gradskive og måle vinklene på figuren for å bestemme hvilke vinkler de hadde behov for i koden.

Elev A1:	Lag en ny kode der du lager en trekant, men hvordan skal vi endre vinkelen så det blir en trekant?
Elev A3:	Sånn sånn og sånn (tegner en trekant)
Elev A1:	Ja da må jeg kanskje ta bort de to nederst linjene her? (Sletter de to nederste kodelinjene så hen står igjen med 3x linjer og 3x snu kommandoer)
Elev A1:	Her skal jeg jo ha 3 sider så da må jeg ta bort de 2 nederste så det ikke blir 4 sider da, og så skal den bare snu seg 2 ganger
Elev A2:	90 grader, nei da blir det 180… nei 45 grader
Elev A1:	Nei det blir feil. Er det noen som har en gradskive?
Elev A3:	(går til læreren og finner frem en gradskive.)
Elev A2:	Hmm jeg skjønner meg ikke helt på denne gradskiven jeg. Hvis vi går sånn og sånn (viser fra 0 grader via midten på gradskiva til 90 grader), så blir det jo en trekant hvis vi tar en linje sånn: (fra 90 grader til 0 grader)
Elev A1:	Hvis jeg prøver på programmet i stedet da… nei det ble ikke en trekant, vi skal ikke ha 60 grader, vi må ha enda mere.
Elev A2:	Sammenlikner med gradskiven igjen (ok her er 60 grader, og her er 90 grader. Hva med 120 grader da?
Elev A3:	Oi det funka jo
Elev A1:	Hvor mange grader hadde du? 120!
Lærer:	Se her ja, hvorfor tror dere at det blir 120grader da?
Elev A2:	Viser på gradskiven, fordi hvis det går opp hit så blir vinkelen så stor, og så videre så må det bli en trekant med like sider, så det er logisk
Lærer:	Hvor mange grader er det i en trekant da? Har dere lært noe om det?
Elev A1,A2:	(Tenker og gestikulerer mye, prøver seg frem på gradskiven, men kommer ikke frem til noe svar)
Lærer:	Eller hva med hvor mange grader trengte dere for å snu dere rundt en hel runde når dere tok salto med figuren i sted?
Elev A1:	Hmm, 180… eller nei det var jo 360 grader, for da ble det hele runden
Elev A2:	Å ja så siden 120 gange 3 er 360 så kommer vi helt rundt!
Elev A1:	Åååååå Det gir jo faktisk mening jo! Det er derfor vi skal ha 90 på firkanten, for da blir det jo 360 dele på 4. Det gir jo faktisk mening da!
Elev A1:	Så da blir 360 utgangspunktet i alle figurene da!

På slutten av utdraget kom læreren bort til gruppa og sammen **evaluerte** de oppgaven og løsningen, slik som Csizmadia (2015) beskriver i sin modell. Gjennom evalueringen oppdaget elevene på gruppen et mønster som de tok i bruk i neste runde med *modify*, og avlutningsvis i *make*-delen av undervisningsøkta. Mønsteret de fant var at alle figurene har en sum av invendige vinkler lik 360 grader. Elevene får her støtte i læreren sine hint,

og på den måten hjelper læreren elevene med å nå sin proksimale utviklingssone. Med støtte av læreren klarte elevene å lære noe nytt som de kanskje ikke ville oppdaget på egenhånd (Vygotsky, 1978).

I denne sekvensen av programmeringsøkta ser vi at elevene lærer noe nytt i matematikk samtidig som de jobber med å lære seg programmering. Når elevene bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse skriver Utdanningsdirektoratet (2020a). I oppgaven som handler om geometri slik som denne, vil det i programmeringsarbeidet kreve at elevene også jobber med matematikken bak konseptene. Mens elevene utforsker programmeringen utforsker de også konseptet «vinkler» i ulike figurer. Her jobber altså elevene med to temaer parallelt, slik som Utdanningsdirektoratet foreslår.

I den neste sekvensen har elevene på gruppa nettopp hatt en gjennomgang av løkker. Her forsøker elevene å bruke det de nå har lært om løkker for å løse denne nye oppgaven.

Oppgaver

1. Endre koden slik at den inneholder en løkke
2. Skriv et program som kan lage en femkant med sidelengde 200
3. Skriv et program som kan laget en sekskant med sidelengde 50.

Lærer:	Nå kommer en ny aktivitet, hvor dere skal gjøre om koden, prøv å gjøre den om slik at den bruker en løkke
Elev A1:	Er dette en løkke? (trykker gjentatte ganger på kjør program)
Lærer:	Nå må du jo sitte å kjøre samme koden om igjen og om igjen for hånd, det er det vi skal prøve å få til uten å måtte gjøre det på nytt og på nytt. Men aller først må dere gjøre om koden slik at den tegner en trekant.
Elev A1:	Ok da må vi gjøre om gradene, også må vi gjenta 3 ganger og ikke 4.
Elev A1:	Æææ denne får jo 3 sider men det er ikke en trekant.
Elev A2:	Kan vi ikke bare prøve med en annen vinkel da?
Elev A1:	Hvor mange grader er det i en trekant da?
Elev A2:	Innsiden er 120, det sa vi jo i sta
Elev A2:	Jeg klart det! (Figur 22)

Figur 22: Trekant 2 med kode

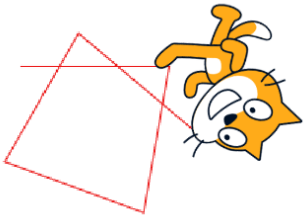
Elev A1: Vi er pro m. du må kom å lær det du også (til en annen elev)

Elev A2: Det neste er 5 kant,

Elev A1: Da blir det gjenta 5 ganger da siden den har 5 kanter)

Elev A2: Hvor mange grader er det da?

Elev A1: Hvis den går bort sånn, og så ut sånn, (viser med hendene vinklene som må til for å få en femkant)

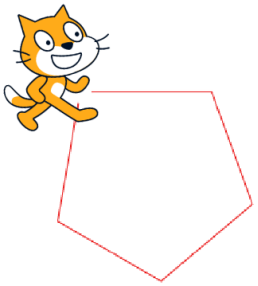


Figur 23: femkant, for stor vinkel

Elev A1: Det ble nesten da, det ble nesten en femkant, men det ble sånn (peker på en figur 23)

Elev A2: Her brukte vi 100, hva med å ta litt mindre da? Sånn 95

Elev A1: Ja ….(Figur 24) Nei det ble for mye enda, hva med 85 da?



Figur 24: Fem kant for liten vinkel

Elev A2: Skal vi se, hvis vi skriver inn 85 her …

Elev A1: Ja jeg fikk det til! … nei akkurat ikke, nå brukte jeg 70 da går det akkurat ikke

Lærer: Når dere jobbet i sted sa dere til meg at på trekant måtte vinkelen bli 120 og på firkant måtte den bli 90. Hva var det dere konkluderte med da?

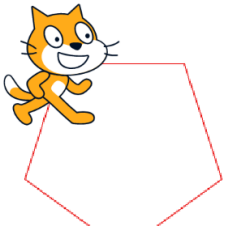
Elev A1: Selvfølgelig… 360… (mumler og begynner å skrive) da må jeg bare ta 360... dele på 5, går opp 7, 1 til overs flytter ned 0. 10 det blir 2 så… det blir 72 da!

Elev A2: Selvfølgelig!

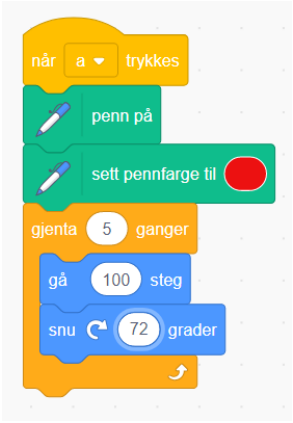
Elev A1: Åååå ja jeg fikk det til!! (Figur 25)

Elev A2: BRA Elev A1!

Elev A1: Så på den neste så blir det bare 360 dele på 6! 360 dele på 6 øm, det går opp 6 gang, så da blir det 60!



Figur 25: Femkant og kode



Elev A2: Så bra! Kult!

Elev A1: Da er det jo lett da! Vi bare deler på 360 uansett!

Elev A2: Ja nå kan vi jo lage en tikant eller noe for det er enkelt å regne ut, det blir 36 grader.

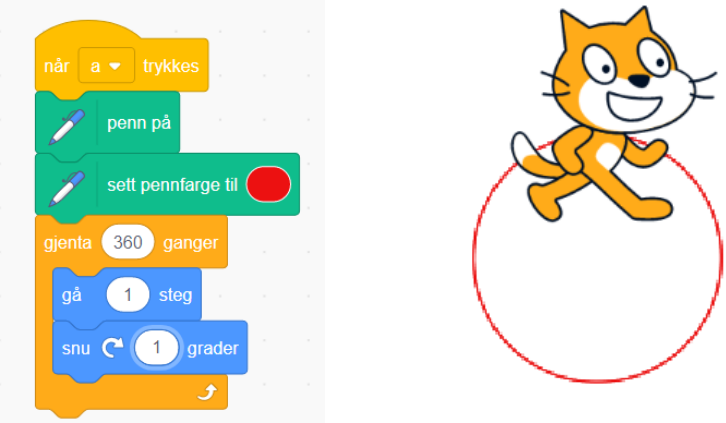
I dette utdraget har elevene gjennom tidligere kunnskap og logisk resonnement klart å finne frem til et mønster som gjelder for alle figurer av denne typen. Ved hjelp av problemløsningsstrategien **Se etter mønster,** hvor elevene ser systematisk på mønster som dannes i matematikken, identifiserer elevene i dette tilfellet at alle figurene har en indre vinkelsumm på 360 grader (Sentance et al., 2019b). Ved hjelp av denne kunnskapen klarte elevene på gruppa å forenkle oppgaven, og komme rett til svarene de var ute etter ved hjelp av et enkelt delestykke. I modellen til Csizmadia et al., «*The computational thinker*» faller elevenes arbeid og løsning inn under konseptet **generalisering av mønster** (Csizmadia et al., 2015). Elevene identifiserer mønstrene i oppgaven, tilpasser løsningen slik at den kan gjelde liknende problemer og overfører ideen fra en løsning til

den neste. Dette er alle adferder som identifiserer at elevene arbeider med generalisering av mønster.

Også her integreres matematikken tett med programmeringen, og elevene jobber med matematiske konsepter samtidig som de lærer nye konsepter i programmering.

Det neste utdraget er også fra oppgaven om å endre koden til å tegne en trekant. Her kommer også flere av de samme problemløsningsstrategiene til elevene frem, men i tillegg benytter elevene på denne gruppen seg av en annen av Posamentier & Krulik sine problemløsningsstrategier, nemlig å **vurdere ekstremaltilfeller**.

(En elev fra en annen gruppe kommer inn i bildet på kameraet mens læreren står der),
Elev E1: Se jeg fikk til sekskant, men så prøvde jeg noe annet! Jeg fikk til sirkel!
Lærer: Hva har du gjort her da?
Elev E1: Jeg bare tok at den gikk 10 steg og snudde 15 grader så da går den helt rundt.
Lærer: Så kult, hvorfor det da?
Elev E1: Nei det er vel fordi den går så korte steg da. Jeg prøvde 1000 grader først, men det gav ingen mening, for da snudde den neste ikke noe, eller jeg tror den bare snudde flere ganger rundt seg selv.
Lærer: Det er godt tenkt da, det stemmer at den bare gikk rundt seg selv, for 3 ganger 360 er jo rett over 1000 så den flyttet seg rundt seg selv 3 ganger og så litt til.
Elev E1: Ja og så prøvde jeg med 1 grad, men da kom den ikke helt rundt. Eller det er jo kanskje fordi jeg ikke tok gjenta 360 ganger da. Vent litt…..
Elev E1: Se det funka jo, eller den ble jo nesten lik som i stad bare at katta gikk saktere rundt.



Figur 26: Sirkel med kode

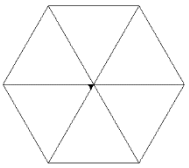
Her har eleven gjennom å se på de ekstreme tilfellene vært i stand til å undersøke og finne ut hva som er viktig for at figuren skal bevege seg en hel runde rundt, og hvilke grenseverdier som er gjeldende for antall grader rotasjon for katten i dette tilfellet. Her har gruppa også gjort seg ferdig med *modify*-oppgaven, og valgt å modifisere videre på egenhånd. Det er i denne overgangen at elevene starter på det 5. og siste steget i PRIMM, *make*.

4.4 Make

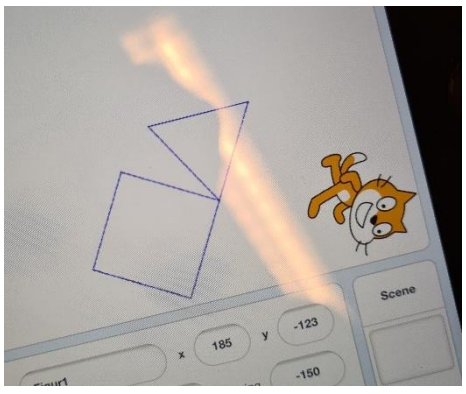
Det siste steget i PRIMM er *make*. I dette steget er målet å lage egen kode. Gjennom de fire foregående stegene har elevene blitt introdusert for, og blitt fortrolig med håndtering av nye konsepter i programmering. I *make* steget er målet at eleven skal ta i bruk disse konseptene til å løse problemer presentert av læreren, eller fra egne ideer. Prosessen med å skape et eget program er svært avansert, men det er en viktig ferdighet å lære for elevene i seg selv. Prosessen starter med at elevene bestemmer seg for et mål, eller får dette av læreren. Deretter velger elevene fremgangsmåte og algoritme som kan hjelpe dem med å skape sitt eget program (Sentance et al., 2019b).

I undervisningen for elevene var det denne gangen begrenset med tid til det siste steget i PRIMM. Elevene var helt på begynnerstadiet innen koding, og det var derfor viktig at konseptene ble tilstrekkelig innlært før vi gikk videre til neste steg. Likevel var det tre oppgaver som faller under kategorien *make*, disse tre oppgavene var markert med «*ekstra oppgave*» på oppgavearket, og de kom til slutt i en rekke av oppgaver. (Oppgavearkene ligger som Vedlegg 3: Aktivitetsark A og Vedlegg 4: Aktivitetsark B – Løkker.) Dermed var det de gruppene som jobbet raskest som gjennomførte *make* oppgavene på den begrensede tiden vi hadde. De første oppgavene følger under.

5. Bruk penn opp og penn ned for å tegne en firkant og trekant ved siden av hverandre.
6. Tegn figuren til høyre:



Elev B1:	Hvis vi skal lage en trekant og firkant må vi jo bare gjøre det vi har gjort da
Elev B2:	Ja
Elev B1:	Eller de må jo kanskje sitte sammen da, så da må vi jo lage koden på nytt
Elev B2:	Hva hvis vi bare lager to koder sånn som vi har fra før og setter de samme etter hverandre da?
Elev B1:	Ja det funker… tror jeg
Elev B2:	Ja
_____	
Elev B2:	Får du til å få den til å bli ved siden av her da? Min blir bare oppå hverandre
Elev B1:	Ja jeg bare får den til å snu seg etter den første koden.
Elev B2:	Hvordan gjør du det da? Bare putter på en sånn grader greie?
Elev B1:	Ja jeg tar 180 grader så blir det halve veien rundt
Elev B2:	Å ja det funker jo!

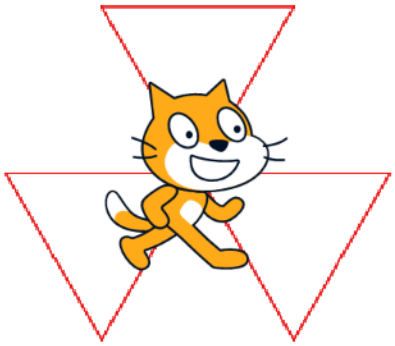


Figur 27: Firkant og trekant

I denne delen av koden bygger elevene på kode de har brukt tidligere. Det å definere om elevene jobbet med *modify* eller *make* i denne oppgaven var utfordrende. Oppgaven var ny for elevene og de måtte selv planlegge en måte å løse oppgaven på. Likevel ser vi også elementer av *modify*, i og med at elevene valgte å benytte seg av kode de har fra før og kopierte denne inn etter hverandre for å løse oppgaven. Det å bruke kode man har skrevet tidligere når man programmerer noe nytt, er ganske vanlig. Her glir disse stegene over i hverandre. Det defineres likevel under *make,* siden elevene får et nytt problem, og må skape en kode for å løse problemet, og ikke bare endre koden de har fått.

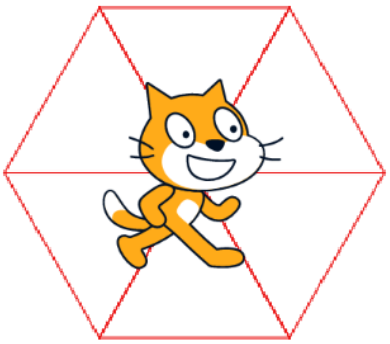
Når det kommer til problemløsningsstrategier velger elevene i denne oppgaven å **forenkle oppgaven**. I og med at elevene tidligere har laget en kodesekvens for trekant, og en kode for firkant, kan de forenkle oppgaven til: «Sett sammen to kodesekvenser og sørg for at de tegnes ved siden av hverandre». I følge Posamentier & Krulik (2008) handler problemløsning ofte om å finne den enkleste og beste veien til en løsning, og i utdraget under kan vi se hvordan elevene bruker koden for trekanten og gjentar denne seks ganger med en rotasjon på slutten av hver gjentagelse, og på den måten løser oppgaven.

Elev B2: Ok hvordan skal vi gjør denne da? Den er jo bare 6 trekanter sammen
Elev B1: Ja da kan vi gjøre som den her da, og så bare snu på dem
Elev B2: Ja så vi bare lager trekanten 6 ganger da? Men hvor mye skal vi snu da?
Elev B1: Vi bare prøver en gang først…. Sånn så her står den da likt som når vi tok den første. Da må vi bare snu så den er langs den neste linja da
Elev B2: Hææ? Hva mener du?
Elev B1: Jo vi bare må legge på en sånn snu brikke og så kan vi få den til å snu seg så mye som den trekanten er. Hvor mange grader er en sånn trekant da?
Elev B2: Er ikke det 120 da?
Elev B1: Hmm nei det ble feil. (Figur 28)

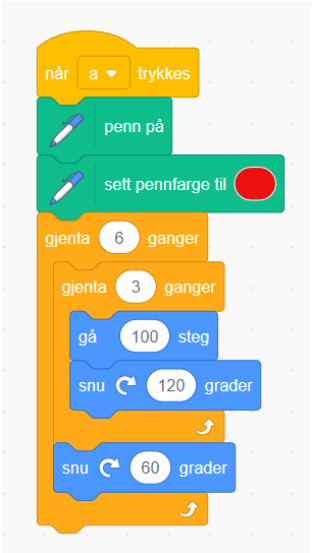


Figur 28: Firkant og trekant

Elev B2: Da er det halvparten da siden den hoppet liksom over en
Elev B1: Sånn ja det funket (Figur 29)



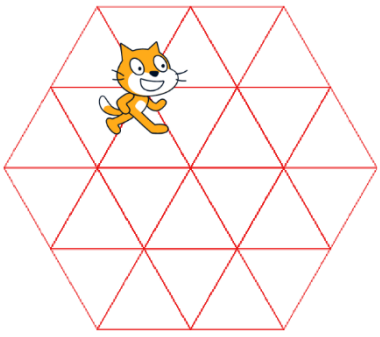
Figur 29: Sekskant med kode

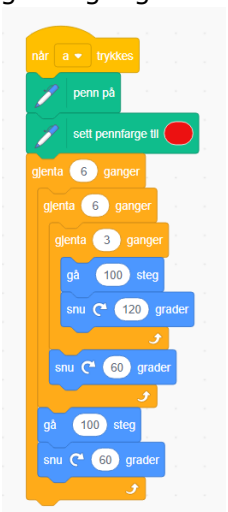


Også i denne oppgaven klarer elevene å bygge på tidligere kodesekvenser og kunnskap for å skape den nye kodesekvensen. Måten elevene benytter gjenta-blokka sammen med prøving og feiling, viser at elevene forstår hvordan en løkke fungerer og gjennom dette klarer de å løse oppgaven. I datamaterialet ser jeg at elevene bruker konseptet **algoritmer** fra Csizmadia et al (2015). Dette konseptet går ut på at elevene tenker algoritmisk for å komme frem til et svar. I klasserommet er det flere elevaktiviteter man kan se etter for å avgjøre om en elev jobber med denne strategien. I dette tilfellet så jeg at elevene brukte «gjentagelser, sløyfer og løkker». (Csizmadia et al., 2015). I alle løsningene hvor elevene programmerte for å svare på oppgaver, måtte elevene jobbe algoritmisk. Dette kom frem gjennom at elevene formulerte instruksjoner for å oppnå en bestemt effekt, formulerte instruksjoner om logiske operasjoner og brukte gjentagelser, sløyfer og løkker samt at elevene testet hypoteser.

I det neste utdraget ser vi hvordan en annen gruppe løste oppgaven. De har her blitt ferdige med å lage kodesekvensene som oppgavearket spør etter, og lurer på om det er mulig å utvide koden mer. Her har elevene blitt ferdige med alle oppgaver som de har fått tildelt av læreren, og prioritert å bruke tiden når de var ferdig til å utforske programmeringsspråket videre og la kreativiteten flyte. Her er det få ytringer som sier noe om hva eleven har tenkt underveis i oppgaven da de stort sett satt hver for seg og jobbet med kodene. De svarer likevel på noen korte spørsmål under oppsummeringen av økta, når deres kode blir vist frem og presentert.

Elev D1: Da er vi ferdige, hva gjør vi nå?
Elev D2: Jeg veit ikke, men jeg lurer på hva som skjer hvis vi tar enda en loop inni loopen på den her (oppgava som gir 6 trekanter i en sekskant)
Elev D1: Prøv da
Elev D2: Det går ikke den bare går inni seg selv igjen.
Elev D1: Kan du ikke bare få den til å gå en gang ekstra da?
Elev D2: Jo…
Elev D2: Oi se så kul den ble!





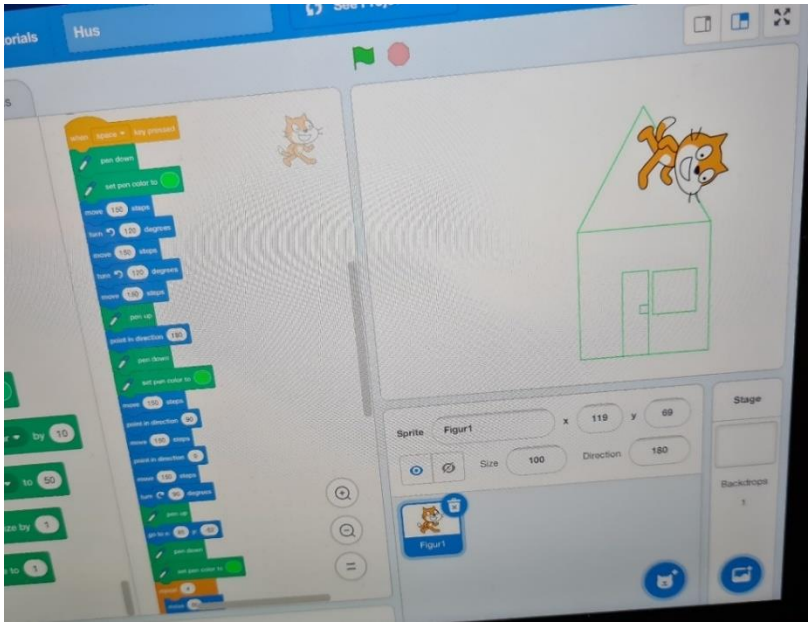
Figur 30: Stor seks kant med kode

I oppsumeringen:
Lærer: Er det noen flere som vil vise frem sin kode?
Elev E1: Ja, jeg har laget meg et hus
Lærer: Hva har du tenkt her da?

Elev E1: Nei jeg veit ikke helt, jeg bare prøvde å lage litt forskjellige figurer, og så fant jeg ut at jeg ville sette de sammen til et hus.

Lærer: Hva har du gjort her da?

Elev D1: Jeg har bare brukt gå og tegn kommandoene til å få katta til å tegne, også brukte jeg sånn gjentagelse eller loop eller hva det het på de siste firkantene etter vi lærte oss det.



Figur 31: Hus laget med en kodesekvens

Målet med PRIMM er at elevene skal bli så kompetente i konseptene som læres bort at de selv kan ta de i bruk og designe egne programmer. Dette vil gi eleven tilfredsstillelsen av å kunne være kreativ og produsere egen kode med mening for eleven selv (Sentance et al., 2019b). I det siste utdraget ser vi at elevene har tatt konseptene de har lært om bevegelse, tegning og gjentagelser og produsert sine egne kodesekvenser. I kodesekvensene de lager bruker de mange konsepter fra *computational thinking* fra Csizmadia et al. (2015). Her bruker elevene **algoritmer** gjennom å formulere instruksjoner, samt bruk av gjentagelser og sløyfer. Videre bruker elevene **generalisering av mønster** for å forstå og bygge videre på mønster i koden, og også **evaluering** for å se om koden stemte med det de hadde sett for seg, og deretter justering for å få koden til å gjøre som de ønsket.

Gjennom analysen av denne undervisningsøkta har jeg pekt på noen problemløsningsstrategier fra Posamentier & Krulik (2008), og noen konsepter og elevadferder som identifiserer disse fra Csizmadia et al (2015). I datamaterialet har jeg funnet at 7 av 10 av Posamentier & Krulik (2015) sine problemløsningsstrategier har kommet frem. Disse var logisk resonnement, se etter mønster, jobbe bakover, vurdere ekstremaltilfeller, forenkle oppgaven, lag en tegning eller ilustrasjon og intelligent gjett og skjekk. Kategorier av problemløsning som ikke ble brukt av elevene denne gangen var se

problemet fra en andre synsvinkler, organisere data og gjøre rede for alle muligheter. Fra Csizmadia et al.(2015) sine konsepter for algoritmisk tankegang kommer alle 6 konseptene frem hos elevene igjennom datamaterialet i denne økta. Dette viser at mange problemløsningsstrategier kan brukes i møte med programmering for elevene. Hvorvidt disse strategiene hadde blitt brukt av elevene i programmering hvis de ikke hadde arbeidet gjennom PRIMM modellen er vanskelig å si, men PRIMM modellen la noen rammer for hvordan elevene skulle jobbe. PRIMM fokuserer på de sosiokulturelle læringsteorier, og gjennom medieiring og støtte fra medelever, læreren og programmeringsprogrammet Scratch jobbet elevene problemløsende med mange konsepter i programmering og matematikk. Derfor kunne vi også finne mange problemløsningsstrategier i arbeidet til elevene.

5 Drøfting

I analysen har jeg sett etter hvordan elevene programmerer når de jobber med PRIMM, og om vi kan se noen problemløsningsstrategier hos elevene i dette arbeidet. I datamaterialet fant jeg 7 av 10 problemløsningstrategier fra Posamentier og Krulik (2015). Jeg identifiserte også at elevene brukte alle seks konseptene fra Csizmadia et al. (2015) sine konsepter for algoritmisk tankegang. Dette er svar på forskningsspørmålet mitt.

Hvilke problemløsningsstrategier kommer til syne når elevene jobber etter PRIMM-modellen?

Gjennom å svare på dette forskningsspørsmålet håpet jeg å kunne finne en sammenheng mellom problemløsningsstrategiene elevene brukte, og PRIMM som modell til å bygge opp undervisning i programmering. Deretter kunne jeg svare på problemstillingen min.

På hvilken måte kan programmering tilrettelegge for problemløsende aktivitet i matematikk?

Gjennom diskusjonen forsøker jeg å belyse på hvilken måte strategiene elevene brukte, kan kobles tilbake til PRIMM-modellen og programmering. Gjennom diskusjonen svarer jeg ut problemstillingen min, og belyser hvordan programmering tilrettelegger for problemløsende aktivitet.

5.1 Problemløsningsstrategier hos elevene

I analysen fant jeg at elevene bruker mange ulike problemløsningsstrategier når de jobber med PRIMM. De 10 problemløsningsstrategiene til Posamentier & Krulik (2015) fra boka «*Problem-solving strategies in mathematics*» er presentert med hensyn til arbeid med matematikk. Likevel viser det seg at i et relativt begrenset tidsrom, og med et begrenset antall elever, tar elevene i bruk 7 av de 10 problemløsningsstrategiene. Algoritmisk tankegang eller «*Computational thinking*» blir også definert, blant annet av Wing (2017) og Csizmadia et al.(2015), som en problemløsningsmodell for informatikk og programmering. De 6 ulike konseptene i modellen til Csizmadia et al.(2015) ble alle identifisert hos elevene som arbeidet med programmering i denne økta. Hva er det så i PRIMM som gjør at elevene tar i bruk så mange problemløsningsmetoder? Sentance et al. (2019) skriver at hvert steg i modellen har sin rolle i opplæringen av eleven i programmering. De ulike delene av PRIMM vil altså ha ulike roller og promotere bruk av ulike strategier for å løse oppgaver. Under ser jeg på hvert av stegene i PRIMM, og om det er noe som promoterer bruken av problemløsning i de ulike stegene.

5.1.1 Predict, run, investigate

De tre første stegene skiller seg ut fra tradisjonell programmeringsundervisning samtidig som de henger tett sammen i PRIMM-modellen. Sentance et al. (2019a;2019b) skriver at de tre stegene *predict, run, investigate* ofte kan gjentas flere ganger frem til eleven forstår konseptene i koden som kjøres. Ut fra analysen ser vi også at flere av de samme problemløsningsstrategiene går igjen her.

I *predict*-delen oppfordres elevene til å diskutere hva en kode kommer til å gjøre. I analysen så jeg spesielt at problemløsningsstrategier som handlet om å undersøke, lete og forstå, kom frem hos elevene. I tillegg så jeg at elevene ble engasjert i gruppene sine, og at de snakket mye om programmet og programmering. I *computational thinking* fra Csizmadia et al.(2015) er det spesielt de tre stegene logisk tankegang og dekomposisjon sammen med evaluering som gjorde seg gjeldende. Disse overlapper også en del med logisk resonnering, se etter mønster og det å lage tegninger og illustrasjoner fra Posamentier & Krulik (2015) sine strategier, som elevene også benyttet i denne fasen.

5.1.2 Modify og make

Modify og make steget i PRIMM handler om å gjøre om en kode eller bygge opp en kode fra bunnen av. Dette er det de fleste forbinder med programmering. Det som skiller *modify* og *make* fra andre måter å lære programmering på, er ikke selve stegene, men det at elevene har kjennskap til koden og konseptene i koden før de selv skal modifisere og lage kode med de samme konseptene. I denne delen av programmeringsøkta var det noen problemløsningsstrategier som gikk igjen, mens det var andre som var nye. Nye strategier jeg så i *modify* og *make* var jobbe bakover, vurdere ekstremaltilfeller og forenkle oppgaven fra Posamentier & Krulik (2015). I tillegg så jeg arbeid med algoritmer, generalisering av mønster og abstraksjon som nye innslag fra Csizmadia et al.(2015) i denne delen.

Logisk tankegang/resonnering

Fordi elevene i første steg av en PRIMM undervisning blir tvunget til å resonnere og diskutere seg frem til hva en kode gjør, tvinger den elevene inn i en logisk resonnering eller tankegang for å prøve å finne ut hva de ulike delene av koden gjør. Logisk tankegang eller resonnering gjør seg spesielt gjeldene når elevene har tidligere erfaring å basere seg på. Csizmadia et al.(2015) sier at elevene i logisk tenking skal resonnere seg fra en tidlig antagelse til å forsøke å forutse hva som kommer til å skje (Csizmadia et al., 2015). Elevene som var med i denne undersøkelsen hadde lite erfaring med å programmere. Dermed ble den logiske resonneringen i første del av opplegget basert på erfaringer fra andre områder. Elevene assosierte gå og snu komandoene med hverdagsspråk, og kunne dermed resonnere seg frem til hva koden gjør uavhengig av tidligere programmeringserfaring. At elevene klarte å assosiere koden med hverdagsspråk kan muligens komme som et resultat av at koden er skrevet i Scratch. Scratch er utviklet for å lære barn å programmere. I programmet kan elevene støtte seg på intuitive kodebrikker som er navngitt for at elevene enklest mulig skal forstå hva de gjør. I andre runde med *predict* baserte de seg på erfaringene de hadde fått fra koden som de hadde jobbet med tidligere. Sentance (2019b) beskriver at elever gjennom gjentaglese av *predict*, *run, investigate*, i den første delen av PRIMM-modellen, får større forståelse av liknende kode. Det ser jeg stemmer her, da elevene i andre runde kunne basere den logiske tankegangen på det de hadde lært i programmering. Etter gjentagelsen klarte elevene også enkelt å forutse hvordan figuren kom til å se ut når den snudde 45 grader ekstra på slutten av koden.

Dekomposisjon

Dekomposisjon av kode er noe jeg observerte hos elevene i samtlige deler av PRIMM-økten. Av analysen kan det se ut til at *predict* steget fremmer bruken av dekomponering hos elevene på en naturlig måte, slik Sentance et al. (2015b) beskriver. Hun skriver at elevene i steget skal se på hele eller deler av programmet, og lete etter ledetråder på hva programmet kommer til å gjøre (Sentance et al., 2019b). Det å ta fra hverandre koden, se på de individuelle delene og forsøke å identifisere hvordan de fungerer, ble veldig naturlig for elevene når oppgaven var å forsøke å forutse hva koden kom til å gjøre. Også videre i arbeidet når elevene kjørte koden (*run*), og når elevene undersøkte koden nærmere (*investigate*), var dekomponering en viktig problemløsningsmetode. Elevene arbeidet med enkelte deler av koden for seg, og snakket om hva den enkelte delen utførte i den fullstendige programmeringskoden. Etter hvert som elevene forsto de ulike delene med kode, var det også lettere for dem å arbeide med hele koden samlet. I min studie kan det tyde på at PRIMM fremmer bruken av dekomposisjon som problemløsningsstrategi ved flere anledninger.

Evaluering

Det er ifølge Csizmadia et al.(2015) generelt i programmering et stort fokus på detaljer, og om det finnes en bedre måte å gjøre ting på. Det å evaluere programkode og programmer er derfor en viktig egenskap elevene skal lære seg. Evaluering er en problemløsningsmetode som skal hjelpe elevene til å oppnå et best mulig resultat når de programmerer. I og med at elevene arbeidet syklisk med de tre første stegene i PRIMM, fikk de hele tiden mulighet til å evaluere og justere hvordan de forsto koden. I de tre første stegene i PRIMM er det et stort fokus på samarbeid, samtale mellom elevene og det å forklare for andre. Dermed kan elevene gjennom samtale oppklare hverandres misoppfatninger, hjelpe hverandre til å forstå ulike konsepter og hjelpe hverandre til å mestre programmering og forståelse der de hver for seg ikke ville kommet videre. I oppbyggingen av PRIMM som modell beskriver Sentance et al. (2019a;2019b), at det er lagt vekt på at modellen skal legge til rette for mediering for læring. Gjennom mediering med medelever, læreren og gjennom et program som Scratch, kan elevene få støtte til å nå sin proksimale utviklingssone. Et eksempel kan være elev A1 og A2 på side 57, som først misforstår hva koden kommer til å gjøre, deretter får de se visuelt gjennom Scratch hva som skjer. Så diskuterer de og gir utrykk for at nå forstår de hvorfor de tok feil i utgangspunktet. De får en opplevelse av at koden i det nye perspektivet gir mening. Gjennom slike prosesser lærer elevene å forstå kode i det sosiale planet. Gjennom dette kan PRIMM bidra til å redusere de innledende hindringene i programmeringsopplæring.

Når det kommer til matematiske problemløsningsstrategier var det «logisk resonnement», «lage tegning eller illustrasjon» og «se etter mønster» som kom spesielt godt frem i de første delene av PRIMM. Det var også enkelte tilfeller av gjett og sjekk.

Se etter mønster

Å gjenkjenne mønster er en strategi jeg kunne se hos elevene i dette datamaterialet, blant annet i den sykliske gjentagelsen av *predict, run* og *investigate*. Gjennom at elevene så en type kode og senere så en liknende kode, klarte de å gjenkjenne utvalgte deler av koden. På denne måten identifiserte elevene likheter mellom kodene, som derfor kom til å gi det samme eller liknende resultat når den ble kjørt. Hvis en tenker på hver gjennomgang av PRIMM som en enkelt hendelse, vil vi kanskje ikke kunne se denne typen problemløsningsmetode. Derfor er det viktig at flere økter som er utformet etter PRIMM-modellen henger sammen, slik at elevene kan bruke det de har lært til å jobbe videre med å gjenkjenne mønster fra tidligere koder. Sentance et al(2019) forteller oss at det kan ta mange gjentagelser av *predict, run, investigate* før en elev forstår de underliggende konseptene som læreren ønsker at eleven lærer. Derfor er det også ønskelig at denne delen av prosessen gjentas frem til elevene får en grunnleggende forståelse for koden (Sentance et al., 2019b). I hver nye runde kan eleven ta med seg mønster fra koden, og lettere kjenne igjen koden som blir presentert. Når mønstre så er etablert hos eleven kan de nyttiggjøre seg disse i arbeidet med modifisering av koden senere. Gjennom den sykliske naturen til de tre første stegene i PRIMM, støtter modellen opp under denne typen problemløsning. Jeg ser at elevene over tid forstår mer og mer, og gjennom diskusjon og samtale hjelper de hverandre å se mønstre i koden. Et eksempel er elev A2 (s 51) som i *predict* delen først ikke forstår hva som skjer når figuren «går 90 og snur 100». Gjennom samarbeid i gruppa identifiserer elevene hva som skjer, og senere i gjennomgangen av løkker (s53), kjenner da elev A2 igjen «gå 90 snu 100» koden. Hen forstår nå at hvis den gjentas fire ganger så vil figuren bli det samme. Senere i oppgaven hvor 90 skal endres til 45 (grader) foreslår elev A2 at siden det var firkant i stad må det bli en halv firkant nå. Hen bruker mønsteret videre for å forsøke å forstå kommende oppgaver. Selv om hen tenker feil i denne oppgaven blir det likevel læring i hvilke komponenter mønsteret inneholder, slik at eleven til slutt bruker mønsteret for å generalisere en måte å tegne femkant og sekskant senere i undervisningsøkten (s.64).

I og med at koden vi brukte ble introdusert, diskutert og undersøkt nøye før elevene selv skulle kode, førte dette til at elevene kunne begynne å se mønster i koden med en gang. Når de så skulle modifisere koden, gjorde de dette med kjent kode og kjente mønstre. De kunne da endre eller utvide koden til å gjøre det de ville. Et godt eksempel er huset (s.69) i det siste eksemplet fra analysen. Her har eleven brukt alle tidligere mønstre til å endre og lage liknende koder etter hverandre, og skapt noe helt nytt. Uten et utgangspunkt og et mønster for hvordan vi kan kode figurer er det usikkert om eleven hadde klart å lage dette på egenhånd.

Lage en tegning eller illustrasjon

Det å lage tegninger og illustrasjoner av koden er noe elevene brukte i utstrakt grad i *predict, run og investigate* delen av økta. I *predict* fasen blir det helt naturlig for elevene å tegne hva koden kommer til å gjøre for å ha mulighet til å forklare de andre elevene hvordan koden kommer til å fungere. I flere tilfeller hjalp en illustrasjon elevene å bestemme om det de forutsa for koden stemte, og på den måten støttet denne problemløsningsmetoden elevene i arbeidet med kodeforståelse. Etter *run* fasen av økta, så jeg også flere tilfeller hvor elevene noterte eller rettet på illustrasjoner ut ifra om koden gjorde det eleven trodde eller ikke. Dette viser at *predict*, *run* og *investigate* i denne typen programmering tilrettelegger for bruken av problemløsningsstrategien lage tegninger og illustrasjoner. Gjennom å planlegge programmering i PRIMM- modellen kan jeg derfor være relativt sikker på at elevene tar i bruk denne problemløsningsstrategien.

Intelligent gjett og sjekk

I datamaterialet kunne jeg se tegn til at elevene gjettet på hva de trodde kom til å skje i kodesekvensene. Det var to typer gjetting som foregikk. Noen elever gjettet helt vilkårlig, disse elevene arbeidet derfor ikke problemløsende. Andre elever gjettet og testet for å smalne ned antallet muligheter for til slutt å sitte igjen med rett svar. Et eksempel var første gang elevene skulle gjøre om programmet fra en firkant til en trekant (kap 4.3). Her gikk elevene systematisk til verks og prøvde seg frem med å justere vinklene til figuren. Denne problemløsningsmetoden viser to ting. Først og fremst viser den at elevene i *predict*- og *run*-steget har fått med seg de viktigste konseptene om hvilke brikker som gjør hva, slik at de er i stand til å justere den riktige parameteren. Samtidig viser det at elevene eksperimenter med matematiske konsepter som grader i geometriske figurer, samtidig som de lærer programmering. Eksempel på dette finner vi på side 62. Der prøver elevene seg frem til de kommer frem til hvor mange grader figuren må snu for å tegne trekanten. Etter å ha jobbet med konseptene i koden var det mange av elevene som forsto godt hvilke deler av koden som gjorde hva. Dette gjorde elevene i stand til å endre den korrekte parameteren i oppgaver de senere skulle løse. Forståelsen for programmet bidro derfor til at elevene kunne benytte problemløsningsstrategien gjett og sjekk effektivt, og i løpet av få gjett, finne frem til svaret på flere ulike oppgaver.

Algoritmer

Algoritmer er noe alle som lærer koder kommer til å jobbe med. I arbeidet med algoritmer kan elevene lage instruksjoner, bruke variabler, teste hypoteser og bruke gjentagelser. I vårt tilfelle fikk elevene en innføring i bruken av algoritmer gjennom de tre første stegene *predict, run* og *investigate* mens de jobbet med å forstå kode. Dermed var elevene allerede kjent med prinsippene i det å arbeide algoritmisk når de skulle endre eller lage kode selv. På den måten kunne de lage instruksjoner og gjentagelser i kreative koder, og oppnå de resultatene de selv ønsket. Om dette er noe elevene hadde fått til uten støtte fra PRIMM, er vanskelig å si, men ut ifra datamaterialet kan jeg se at elevene får en tidlig forståelse for disse konseptene, noe som muligens kan støtte dem i bruken av disse.

Oppsummert

Det som gjør PRIMM unikt i programmeringspedagogiske sammenheng er vekten modellen legger på tidlig kodeforståelse og kodekommunikasjon. Et prinsipp i PRIMM er at man ikke skal måtte kopiere kode blindt, men heller få tildelt en kode som skal undersøkes og diskuteres. Dette er så langt jeg har vært i stand til å finne ut, unik blant metoder som handler om å lære bort programmering. Startkodene, kodene som elevene får utdelt i begynnelsen av programmeringsøkta, blir en inngang til å forstå ulike konsepter i programmering. Gjennom gjentatte repetisjoner av de tre første stegene i PRIMM kan elevene jobbe med de samme konseptene flere ganger og i ulike kodesekvenser. Gjennom dette skaffer elevene seg mer forståelse for konseptene det undervises i. Sentance et al.(2019a;2019b) beskriver at diskusjoner rundt koden kan foregå i det sosiale planet mellom elever, men gjennom arbeidet vil konseptene flyttes til det kognitive planet. Man kan si at elevene gjennom arbeid vil internalisere og forstå koden, og dermed flytter forståelsen seg til det kognitive planet (Sentance et al., 2019b; Vygotsky, 1981). I mitt datamateriale så jeg at *predict, run, investigate* guider elevene inn i noen problemløsningsprosesser. Disse prosessene tror jeg ikke ville oppstått dersom elevene ikke hadde fått utdelt en programkode sammen med et opplegg som stiller konkrete spørsmål rundt forståelsen av denne koden. Dermed vil jeg argumentere med at *predict, run, investigate* hjelper elevene i gang med å tenke problemløsende rundt programmering. Gjennom hele PRIMM vil jeg si at elevene arbeidet med selvtillit, og var ikke redde for å gjøre feil. Jeg ser flere steder at elevene gjør feil i kodeprosessen og i de matematiske konseptene, likevel jobber de videre og retter opp feil som er gjort. Jeg synes det kan virke som om programmering og PRIMM modellen ufarliggjør det å gjøre feil. En viktig prosess i programmering er hele tiden å evaluere løsningene, og se om noe kan gjøres på en bedre måte. Siden dette er så naturlig i PRIMM modellen, virker det ikke som elevene er redde for å gjøre feil. Når de gjør feil, fører dette heller til læring og en mulighet til å forbedre programmeringskode og forståelse.

Jobbe bakover, forenkle oppgaven og vurdere ekstremaltilfeller

Problemløsningsstrategiene jobbe bakover, forenkle oppgaven og vurdere ekstremaltilfeller, er alle eksempler på strategier hvor jeg ikke har funnet noen direkte kobling til PRIMM hos elevene når de har jobbet med dem. I alle tilfellene i materialet mitt hvor disse problemløsningsstrategiene kommer opp, har elevene hatt god støtte i programmeringsspråket, og det de har lært tidligere i økta. Samtidig tror jeg at elevene ville kunne brukt disse strategiene selv om de ikke hadde jobbet med PRIMM.

5.2 Programmering og matematikk

I denne studien har jeg sett på elever som lærer seg programmering på et begynnerstadium. Likevel ser jeg gjennom datamaterialet at elevene samtidig som de lærer seg ulike konsepter i programmering, også arbeider med konsepter fra matematikken som for eksempel vinkler. Utdanningsdirektoratet (2020a) skriver at når elever bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse. I PRIMM modellen ser vi at elevene blir guidet gjennom programmeringskonseptene steg for steg, og underveis klarer elevene også å utforske matematiske konsepter. Om det er tilfellet at all bruk av programmering til å utforske problemer, er gode verktøy for å utvikle matematisk forståelse, er vanskelig å si noe om, men i den avgrensede studien min peker datamaterialet på en slik sammenheng. Elevene så ut til å mestre arbeidet med konsepter og ytret forståelse for både programmering og matematikk.

5.3 Studiens begrensninger

Alt i alt kan det være vanskelig å bedømme hva som får elevene til å velge en strategi over en annen, og hva som kan tilrettelegge for problemløsende aktivitet i matematikk og programmering. En av utfordringene jeg har støtt på i arbeidet med denne undersøkelsen, er mangelen av en kontrollgruppe. Hvis jeg skulle hevdet bastant at PRIMM legger godt til rette for problemløsende aktivitet hos elevene når de programmerer i matematikk, måtte jeg ha gjennomført tilsvarende opplegg på en tilsvarende gruppe, men da uten bruk av PRIMM i planleggingen av undervisningen. Her kunne en tradisjonell metode for undervisning i programmering blitt brukt slik som det beskrives av Benton et al. (2016). En slik metode ville i kontrast til PRIMM fulgt steg for steg instruksjoner, hvor elevene skriver en og en bit av et program instruert av læreren eller en manual mot et bestemt mål. Spørsmålet om elevene hadde brukt like mange problemløsningsstrategier i en slik programmeringstime er noe som hadde vært spennende å undersøke.

Et annet moment som kunne styrket oppgaven min er intervju med elevene, eller en mulighet til konstant å observere en gruppe underveis i arbeidet, der observatøren kunne fått svar på hva eleven selv mente var grunnen til at hen valgte akkurat de strategiene som ble valgt. I oppsummeringen av timen er det flere av elevene som svarer på spørsmålet om hva som var bra med økta. Her svarte de blant annet at det var bra at de fikk mulighet til å undersøke en kode før de selv kunne kode. I tillegg var det fint å få utdelt et utgangspunkt for videre koding, slik at de slapp å starte på null når de skulle bygge egen kode. Dette hadde vært interessante temaer å undersøke nærmere.

5.4 Tidliligere forskning på PRIMM

PRIMM er en relativt ny metode for å lage undervisningsopplegg, og det er dermed begrenset med forskning på hvordan dette fungerer i praksis. Sentance et al., (2019) testet selv PRIMM-modellen i to runder tilbake i 2017 før utgivelsen av metoden. I hovedstudien på 493 elever og 14 lærere ble elevene undervist i 10 økter fordelt på 8-12 uker. 180 elever deltok i tillegg som kontrollgruppe som ble undervist i tradisjonell programmering. Alle elevene gjennomførte en pre og post-test, og elevene som deltok i studien hadde signifikant bedre resultat på ettertesten enn elevene i kontrollgruppen. Elevene hadde en betydlig høyere forståelse for hva ulike koder gjorde, og også større måloppnåelse målt opp mot målene i det engelske computingpensumet. I tillegg raporterte lærerne i eksprimentet at elevene så ut til å lære mer gjennom PRIMM-metoden enn slik de hadde undervist tidligere (Sentance et al., 2019a).

Uavhengig forskning på PRIMM modellen begrenser seg til noen få frittstående artikler. Law (2020) beskriver hvordan elever som arbeider med programmering etter PRIMM virker mindre bekymret for å gå i gang med å programmere, og at de heller ikke var så redde for å gjøre feil. Jeg har aldri tidligere undervist i programmering. Jeg kan derfor ikke si noe om hvordan elever som programmerer på andre måter jobber. Blant elevene i min undersøkelse, var det heller ingen som var redde for å gå i gang med kodingen. Alle forsøkte å løse problemene uten å nøle. Ut i fra egen analyse, og det Law skriver kan det virke som PRIMM støtter elevene inn i nye konsepter på en måte som ufarliggjør det å ta det første skrittet inn i programmering. I stedet for å fokusere på hva som kan gå galt, fokuserer elevene på ulike måter å møte neste problem og hvilke strategi de skal bruke i møte med den.

En av få andre artikkler om PRIMM, *PRIDAM: a framework for teaching programming*, Udenze & Elfallah, (2020), beskriver hvordan *modify* steget i PRIMM har noen mangler når det kommer til arbeid på høyere nivå med koding som har flere simultane prosesser (Udenze & Elfallah, 2020). I mine funn finner jeg at *modify* steget er mer enn godt nok som støtte til elevene på det nivået de programmerte på i dette tilfellet. Her er koden kort nok til at elevene selv klarer å ha oversikt over hele koden. Vi ser fra analysen at elevene ved flere anledninger benytter seg av problemløsningsstrategien dekomposisjon, men at dette skjer i *predict*-delen, hvor elevene dekomponerer koden for å se på deler av koden om gangen. Det å dele opp *modify* i *decompose* og *arrange* slik Udenze & Elfallah (2020) foreslår, kan ha noe for seg; for å implisitt minne læreren på å støtte elevene i problemløsningsstrategien dekomposisjon. Dette må i så fall veies opp mot at modellen blir mer avansert og at det derfor blir flere hensyn å ta for læreren i arbeidet med å lage undervisningsopplegg til elevene. Etter min mening klarer PRIMM slik det står i dag, å støtte problemløsningsstrategiene elevene trenger for å arbeide med koding, inkludert dekomponering, uten at vi trenger å dele opp denne noe mer. Denne videreutviklingen av PRIMM blir da etter mitt syn unødvendig i begynneropplæringen av programmering på barneskolen.

5.5 PRIMM i skolen

Det at PRIMM ser ut å støtte elevene i arbeidet med programmering, og samtidig støtte elevenes problemløsningsstrategier, gjør det fristende å foreslå en innføring av denne metoden til bruk i programmeringsundervisning på barneskolen. Utdanningsdirektoratet har som mål for programmeringen i norsk skole at den skal gi elevene flere verktøy til å løse oppgaver og problemer med. I kompetansepakken skriver de at programmering skal styrke undervisningen gjennom problemløsning, eksperimentering, læring for flere og dybdelæring (Utdanningsdirektoratet, 2020b). PRIMM som modell, har i min studie vist at den kan støtte opp om minst to av disse direkte. PRIMM som modell bygger på det sosiokulturelle, og har samarbeid og kommunikasjon som en av sine viktigste komponenter. I tillegg har jeg vist at PRIMM støtter opp om bruken og utviklingen av problemløsningsstrategier hos elevene. Utdanningsdirektoratet skriver i sitt notat programmering i skolen at «*Programmering gir en systematisk tilnærming til problemløsning*». Elevene skal gjennom strukturerte aktiviteter med prøving og feiling, systematisk feilsøking samt utforsking, finne de beste løsningene. Gjennom dette skal elevene bli bedre problemløsere. Det kan se ut som om elevene gjennom arbeid med PRIMM kan oppnå Utdanningsdirektoratets ønske, og støtte opp om programmering som en måte for elevene å arbeide problemløsende.

5.6 Metaperspektiv

Ludviksen-utvalget skriver at de anbefaler at det legges vekt på metakognisjon og selvregulering i alle fag (NOU 2015:8). I notatet for programmering i skolen skriver Utdanningsdirektoratet (2018) at programmering som aktivitet kan være med å styrke denne læringen. Elever som programmerer må planlegge hva de skal lage og i evalueringen ser eleven på hva som kan gjøres bedre eller videreutvikles. Det pekes også på at programmering gir en systematisk tilnærming til problemløsning som innebærer å prøve og feile, finne de beste løsningene, abstrahere løsningen, og lage algoritmer for å løse denne typen problemer. I datamaterialet i studien så jeg tendenser til at elevene arbeidet på denne måten. PRIMM guider elevene igjennom en strukturert plan for læring av ulike konsepter som innebærer en støtte for elevene i denne prosessen. Hvorvidt elevene selv er klar over prosessen de står i, er vanskelig å si ut ifra datamaterialet mitt, men er et interessant spørsmål som kanskje kan undersøkes nærmere. Et annet spørsmål som oppstår, er hvorvidt en kan forvente at elevene kan overføre disse problemløsningsstrategiene til andre fag eller områder. Dolonen et al. (2019) skriver i sin litteraturgjennomgang av programmering i skolen, at det på 80 og 90-tallet eksisterte en ambisjon om at problemløsning elevene lærte i programmering kunne overføres til andre fag og situasjoner. Selv om ambisjonen ikke ble oppnådd den gangen er Dolonen et al. (2019) tydelig på at verden er en ganske annen i dag, og at argumentene for en slik sammenheng må studeres på nytt i lys av dagens situasjon. Å undersøke en slik sammenheng kan derfor være spennende i en videre studie.

6 Konklusjon

I dette masterprosjektet har jeg gått igjennom tidligere forskning som er aktuell for problemstillingen, samt relevant teori. Sammen med analyse av empiri og drøftinger av funn, skal jeg nå trekke noen oppsummeringer som svarer på problemstillingen.

På hvilken måte kan programmering tilrettelegge for problemløsende aktivitet i matematikk?

Ved hjelp av forskningsspørsmålet mitt skulle jeg forsøke å finne et svar på dette spørsmålet.

Hvilke problemløsningsstrategier kommer til syne når elevene jobber etter PRIMM-modellen?

Fra forskningsspørsmålet viste det seg at så mange som 7 av 10 problemløsningsstrategier fra Posamentier & Krulik (2015) og alle 6 av konseptene i Csizmadia et al.(2015) modell for *computational thinking* ble tatt i bruk av elever som jobbet i en programmeringsøkt designet gjennom PRIMM. Dette tyder på at bruken av programmering kan få elever til å arbeide problemløsende.

Utdanningsdirektoratet (2018) skriver at elevene gjennom programmering «*får en systematisk tilnærming til problemløsning*». Videre at elevene gjennom programmering gjennomfører strukturerte aktiviteter med feilsøking, prøving og feiling, og generalisering av løsinger og får «*kompetanse i å lære*»(Utdanningsdirektoratet, 2018). Ikke all programmeringsundervisning gir elevene en slik systematisk tilnærming til problemløsning, og ikke all programmeringsundervisning gir slike strukturerte aktiviteter som Utdanningsdirektoratet mener er viktig for elevenes læring. Benton et al.(2016) skriver at mange lærere har et begrenset syn på programmering som er knyttet til steg-for-steg-instruksjoner for å skrive kode, mens Dolonen et al. (2019) beskriver at få lærere har kompetanse til å undervise programmering. Dolonen et al. (2019) skriver også hvilke institusjonelle problemer som oppsto ved innføringen av programmering, og behovet for et adekvat undervisningsopplegg og pedagogisk verktøy. Et slik pedagogisk verktøy for å designe programmeringsopplegg er nødvendig for å heve programmeringsundervisningen.

PRIMM er en pedagogisk modell som kan hjelpe lærere i programmering å tilrettelegge for problemløsende aktivitet i programmering og matematikk. Modellen baserer seg på sosiokulturell læringsteori og mediering, og legger opp til strukturerte programmeringsaktiviteter for elevene. Jeg har gjennom denne studien observert at elever som jobber med programmering gjennom et PRIMM opplegg, samtidig jobber problemløsende. PRIMM-modellen er en modell som setter lærere i stand til å produsere sine egne undervisningsopplegg i programmering, som gjennom fem steg potensielt sikrer at elevene arbeider problemløsende i programmering.

6.1 Videre forskning på problemløsing

I læreplanen er det lagt vekt på at elevene skal bli gode problemløsere. Problemløsning og algoritmisk tankegang er synliggjort av Utdanningsdirektoratet i læreplanen som viktige og store deler i den nye læreplanen (Utdanningsdirektoratet, 2020a). I denne studien har jeg undersøkt og funnet ut at elever arbeider problemløsende når de programmerer i PRIMM. Nye spørsmål dukker opp som en konsekvens av disse resultatene. PRIMM er en forholdsvis ny modell og mer forskning er nødvendig på området for å trekke solide konklusjoner. Utdanningsdirektoratet (2020a) skriver *«I læreplanen er algoritmisk tenkning synliggjort fordi dette er en viktig problemløsningsstrategi. Når elevene bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse»*. Hvis man kan slå fast at PRIMM-metoden for programmeringsundervisning setter lærere i stand til å drive undervisning som både gir elevene opplæring i programmering, og som tilrettelegger for problemløsende aktivitet for elevene i matematikk samtidig, ville man kunne oppnå Utdanningsdirektoratets intensjon. I en skolehverdag med stor stofftrengsel i alle fag, hadde en slik sammenheng kunne lettet arbeidet med programmering, og rettferdiggjort innføringen av programmering i allerede eksisterende fag. Hvis denne eller en annen metode for undervisning i programmering hadde oppnådd entydige resultater og annerkjennelse kunne vi sikret god meningsfylt undervisning i programmering for alle elevene i grunnskolen. Det å lage en undersøkelse med en kontrollgruppe som kan undersøke hvorvidt, og i hvor stor grad PRIMM øker kvaliteten på programmeringsundervisningen, hadde vært et godt bidrag til en slik forståelse. Det finnes også andre prosjekter som har som mål å forene matematikk og programmering. ScratchMaths fra Benton et al. (2016) er et eksempel på et slikt prosjekt. Det å se på og sammenlikne ulike prosjekt med PRIMM, for å se hvor godt egnet de er i forhold til den norske læreplanen er også en mulig vei videre.

Referanser

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

Benton, L., Saunders, P., Kalas, I., Hoyles, C. & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International journal of child-computer interaction*, *16*, 68-76.

Berggren, S. & Jom, P. (2019, 21. november 2019). Lærerne er positive til programmering - men mangler kunnskap. *Utdanningsnytt.no* https://www.utdanningsnytt.no/fagartikkel/fagartikkel-laererne-er-positive-til-programmering---men-mangler-kunnskap/220753

Blockly. (u.å.). Hentet 21.05 fra https://blockly.games/

Clements, D. H. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education Annual*, *1999*(1), 147-179.

Cohen, L., Manion, L. & Morrison, K. (2011). *Research methods in education* (7. utg.). NY Routledge

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C. & Woollard, J. (2015). Computational thinking-A guide for teachers.

Cutts, Q., Esper, S., Fecho, M., Foster, S. R. & Simon, B. (2012). The abstraction transition taxonomy: Developing desired learning outcomes through the lens of situated cognition. Proceedings of the ninth annual international conference on International computing education research,

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

Gold, R. L. (1958). Roles in Sociological Field Observations. *Social Forces*, *36*(3), 217-223. https://doi.org/10.2307/2573808

Johannessen, A., Tufte, P. A. & Christoffersen, L. (2016). *Introduksjon til samfunnsvitenskapelig metode* (5. utg.). abstrakt forlag

Kirke- utdannings- og forskningskomiteen. (2016). *Fag – Fordypning – Forståelse. En fornyelse av Kunnskapsløftet . (Innstilling 19S. 2016-2017)*. https://www.stortinget.no/globalassets/pdf/innstillinger/stortinget/2016-2017/inns-201617-019s.pdf

Kunnskapsdepartementet. (2017). *Framtid, fornyelse og digitalisering. Digitaliseringsstrategi for grunnopplæringen 2017-2021*. https://www.regjeringen.no/contentassets/dc02a65c18a7464db394766247e5f5fc/kd_framtid_fornyelse_digitalisering_nett.pdf

Law, R. (2020). A pedagogical approach to teaching game programming: Using the PRIMM approach. Proceedings of the 14th European Conference on Games Based Learning,

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, *2*(1), 32-37.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B. & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bull.*, *36*(4), 119–150. https://doi.org/10.1145/1041624.1041673

Lær Kidsa Koding. (2021). *Kodetimen*. https://www.kidsakoder.no/kodetimen/pameldte-til-kodetimen-2020/

Micro:bit. (u.å.). Hentet 21.05 fra https://microbit.org/

Mørch, A. I., Litherland, K. T. & Andersen, R. (2019). End-User development goes to school: Collaborative learning with makerspaces in subject areas. International Symposium on End User Development,

NOU 2014:7. (2014). *Elevenes læring i fremtidens skole— Et kunnskapsgrunnlag*. Kunskapsdepartementet.

https://www.regjeringen.no/contentassets/e22a715fa374474581a8c58288edc161/no/pdfs/nou201420140007000dddpdfs.pdf
NOU 2015:8. (2015). *Fremtidens skole. Fornyelse av fag og kompetanser*. Kunnskapsdepartementet Oslo. https://www.regjeringen.no/no/dokumenter/nou-2015-8/id2417001/
O'Brien, B. (2019). *How to choose the right type of robot for your classroom*. https://meetedison.com/how-to-choose-the-right-robot-for-your-classroom/
Papert, S. (1980). *Mindstorms : children, computers, and powerful ideas*. Basic Books.
Papert, S. (1993). *The children's machine: Rethinking school in the age of the computer*. ERIC.
Polya, G. (2004). *How to solve it: A new aspect of mathematical method* (Bd. 85). Princeton university press.
Posamentier, A. S. & Krulik, S. (2008). *Problem-solving strategies for efficient and elegant solutions, grades 6-12: a resource for the mathematics teacher*. Corwin press.
Posamentier, A. S. & Krulik, S. (2015). *Problem-solving Strategies In Mathematics: From Common Approaches To Exemplary Strategies*. World Scientific Publishing Company. https://books.google.no/books?id=VTY8DQAAQBAJ
Postholm, M. B. (2017). *Kvalitativ metode: En innføring med fokus på fenomenologi, etnografi og kasusstudier* (2. utg.). Universitetsforlaget.
Postholm, M. B. & Jacobsen, D. I. (2018). *Forskningsmetode*. Cappelen Damm Akademisk.
Resnick, M. (2012a). *Let's teach kids to code* [Video]. https://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code
Resnick, M. (2012b). Reviving Papert's dream. *Educational Technology*, *52*(4), 42-46.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. & Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67.
Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E. & Voll, L. O. (2016). *Teknologi og programmering for alle - En faggjennomgang med forslag til endringer i grunnopplæringen*. Utdanningsdirektoratet https://www.udir.no/tallog-forskning/finn-forskning/rapporter/teknologi-og-programmering-for-alle
Schoenfeld, A. H. (2016). Learning to think mathematically: Problem solving, metacognition, and sense making in mathematics (Reprint). *Journal of Education*, *196*(2), 1-38.
Schulte, C. (2008). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. Proceedings of the Fourth international Workshop on Computing Education Research,
Scratch. (u.å.). Hentet 21.05 fra https://scratch.mit.edu/
Selby, C. & Woollard, J. (2013). Computational thinking: the developing definition.
Sentance, S., Waite, J. & Kallia, M. (2019a). Teachers' Experiences of using PRIMM to Teach Programming in School.
Sentance, S., Waite, J. & Kallia, M. (2019b). Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education*, *29*(2-3), 136-176. https://doi.org/10.1080/08993408.2019.1608781
Thagaard, T. (2018). *Systematikk og inlevelse* (5 utg.). Fagbokforlaget
Tjora, A. (2017). *Kvalitative forskningsmetoder i praksis. 3. utgave*.
Udenze, A. & Elfallah, M. (2020). PRIDAM: a framework for teaching programming. *Research in Teacher Education*, *10*(1), 6-10.
Utdanningsdirektoratet. (2018). *Notat om programering i skolen* https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf
Utdanningsdirektoratet. (2019a). *Algoritmisk tenking* https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/
Utdanningsdirektoratet. (2019b). *Om algoritmisk tenking og programering* https://www.udir.no/kvalitet-og-kompetanse/nasjonale-satsinger/den-teknologiske-skolesekken/om-algoritmisk-tenking-og-programmering/

Utdanningsdirektoratet. (2020a). *Hva er nytt i matematikk?* . http://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-matematikk/
Utdanningsdirektoratet. (2020b). *Kompetansepakken for lærere - Programmering og algoritmisk tenkning*. https://bibsys.instructure.com/courses/387
Utdanningsdirektoratet. (2020c). *Læreplan i matematikk 1.–10. trinn* (MAT01-05). https://data.udir.no/kl06/v201906/laereplaner-lk20/MAT01-05.pdf?lang=nob
Utdanningsdirektoratet. (2020d). *Søk om midler til utstyr for programering i skolen*. https://www.udir.no/Udir/PrintPageAsPdfService.ashx?pid=147681&epslanguage=no
Vygotsky, L. (1978). *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press.
Vygotsky, L. S. (1981). The instrumental method in psychology. *The concept of activity in Soviet psychology*, *2*(3), 135-143.
Vygotsky, L. S. (2004). Imagination and creativity in childhood. *Journal of Russian & East European Psychology*, *42*(1), 7-97.
Waite, J. (2018). Literature review: pedagogy in teaching. https://royalsociety.org/-/media/policy/projects/computing-education/literature-review-pedagogy-in-teaching.pdf
Wikipedia contributors. (2121). *Scratch (programming language)*. Hentet 21.05 fra https://en.wikipedia.org/w/index.php?title=Scratch_(programming_language)&oldid=1024387749
Wing, J. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology*, *25*(2), 7-14.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.
Woods, D. (2000). An Evidence-Based Strategy for Problem Solving. *Journal of Engineering Education*, *89*. https://doi.org/10.1002/j.2168-9830.2000.tb00551.x
Wæge, K. (2007). *Elevenes motivasjon for å lære matematikk og undersøkende matematikkundervisning* [ph.d.]. http://hdl.handle.net/11250/258129

Vedlegg

Vedlegg 1: Undervisningsplan: Tegning og løkker

Vedlegg 2: PowerPoint for undervisnings økt

Vedlegg 3: Aktivitets ark A: Introduksjon til Scratch med tegning

Vedlegg 4: Aktivitetsark B – Løkker

Vedlegg 5: Samtykke skjema

Vedlegg 1: Undervisningsplan

Undervisningsplan: Tegning og løkker

Leksjonens mål
• I denne leksjonen skal vi bruke Scratch med «penn, tillegg» til å tegne former • Du vil lære om løkker i Scratch • Du vil lære gjentagelse i programmering • Alle skal kunne modifisere programmer som bruker en for løkke • Noen av dere vil kunne skrive egne programmer med en for løkker

Lærerne notater
Scratch med «penn tillegg» er i bunn og grund en type skilpadde programmering. Denne typen programmering er nyttig undervisning i løkker, konsolidering av funksjoner og innføring av parametere.

I denne økten introduseres funksjoner for å tegne figurer ganske raskt, ved hjelp av penn tillegget i Scratch. Dette kan også bidra til å konsolidere funksjoner for navngivning. Studentene skal kunne lære tegne kommandoene i rekkefølge ganske raskt, og deretter introduseres for-sløyfen halvveis gjennom leksjonen for å vise hvordan de samme formene kan gjøres mer effektivt.

Oppbygging av økten

Tid	**Aktivitet**	**Lærings ressurs**
0- 5 min	Presentasjon, kom i gang. Rask gjør nå - se om elevene kan forutsi at det vil være en form	Gjør nå aktivitet (Predict)
5-20 min	Studentene laster ned programmet og svarer på noen spørsmål parvis eller i grupper for å hjelpe dem med å undersøke koden - gå gjennom svarene	Aktivitetsark 4a
20-45 min	Programmering - elevene endrer kvadratfunksjonen og lager flere former	Aktivitetsark 4a
45-50 min	Kort pause, eleven ut å får seg luft	
50 – 65 min	En ny forutsi oppgave - å se på kvadratkoden som er en for løkke - demo til klassen	Aktivitetsark 4b
65 – 80 min	Programmering - endre trekant og andre former ved hjelp av en for løkke	Aktivitetsark 4b
80 – 90 min	Gjennomgang - fyll ut hullene i trekantsfunksjonen (elevene kan gjøre på et papir og deretter prøve i scratch.	Oppgave

Vedlegg 2: PowerPoint for undervisnings økt

Scratch Programering

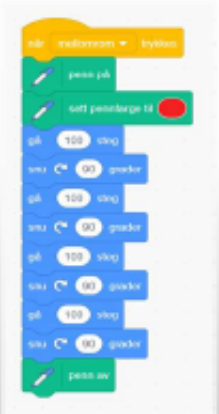
Lesson 4 of 10

1

Snakk med sidemannen

PRIMM

Hva tror du denne koden vil gjøre?



Tegn det du tror koden kommer til å gjøre

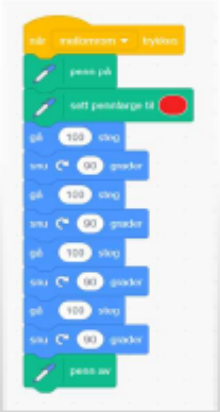
2

PRIMM

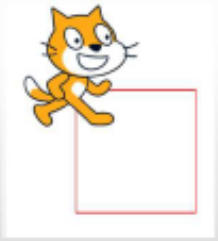
KINGS College LONDON

Svar på oppgaven:

Hva tror du denne koden vil gjøre?



Den tegnet et rødt kvadrat



3

KINGS College LONDON

I denne økta skal vi lære

- Hvordan vi kan tegne med scratch
- Hvordan du kan bruke løkker i Scratch.
- Om hvordan du bestemmer anntallet gjentagelser i en løkke.
- Hvordan endre et program som inneholder en løkke
- Noen vil lære å skrive egen program med løkker

4

Aktivitet 1: ca 15 minutter

PRIMM

KING'S College LONDON

Åpne scratch programmet som er delt på Teams.
Lagre programmet på din egen bruker
Kjør programmet og se at du får et kvadrat.
Svar på spørsmålene på oppgavearket.

5

Programming: 25 minutes

PRIMM

KING'S College LONDON

Gjør oppgave 2 på aktivitetsarket.

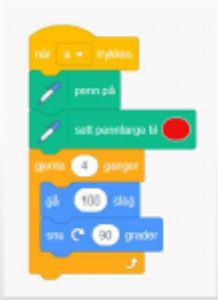
6

Neste del.

PRIMM

KINGS College LONDON

Hva gjør denne koden?



Hva er anderledes med denne?

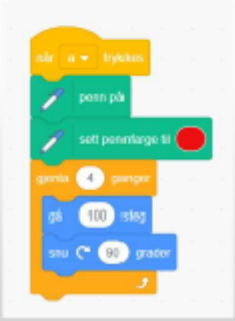
Tegn hva som skjer

7

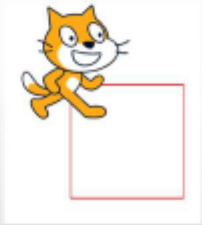
Neste del… Svaret

KINGS College LONDON

Hva gjør denne koden?



Svaret



8

Hva med denne?

Hva med denne!

Hva er forskjelling her ?

Tegn her

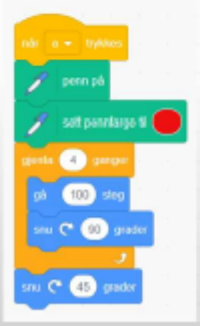
PRIMM

KING'S College LONDON

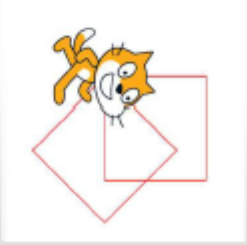
9

Hva med denne? Svar:

Hva med denne?



Tegn her



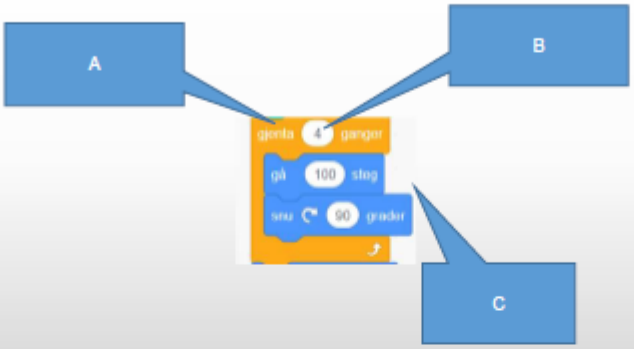
KING'S College LONDON

10

Undersøk løkken

PRIMM

KING'S College LONDON



1	Disse to linjene gjentas 4 ganger
2	Dette er en teller som teller hvor mange ganger det skal gjentas
3	Dette forteller oss at det inni skal gjentas

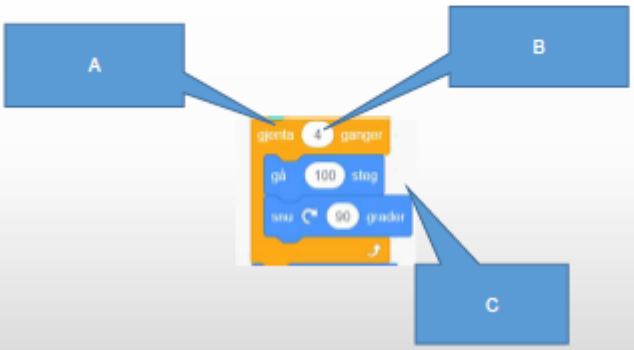
Finn ut hvilke av A, B og C som passer til beskrivelsene 1,2 og 3

11

Undersøk løkken

PRIMM

KING'S College LONDON



1 C	Disse to linjene gjentas 4 ganger
2 B	Dette er en teller som teller hvor mange ganger det skal gjentas
3 A	Dette forteller oss at det inni skal gjentas

Finn ut hvilke av A, B og C som passer til beskrivelsene 1,2 og 3

12

Nå gjør disse oppgavene

PRIMM

KING'S College LONDON

1. Åpne den neste linken til Scratch startprogrammet og lagre på din side. Kjør den
2. Endre trekant oppgaven slik at den bruker en løkke
3. Skriv et program som kan lage en 5 kant og en 6 kant

Ekstra oppgave
Lag et program som tegner en konvolutt

Vedlegg 3: Aktivitetsark A

Aktivitets ark A: Introduksjon til Scratch med tegning

Før du starter skal du åpne oppstarts programmet på scratch.

Oppgave 1: Svar på følgende spørsmål:

1. I operasjonen Snu (høyre) 90, hva tror du 90 betyr?
   _____
2. Hva gjøre gå (100)?

_____

_____

3. Hva skjer om du flytter penn av til tidligere i koden?

_____

4. Hva skjer om du endrer alle (høyre 90) Til (høyre 45)? Svar først så prøv og se hva som skjer.

_____

_____

5. Hva slags kodebrikke tror du du må bruke for å gå bakover og til venstre i stedet?

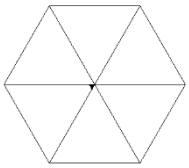
_____

_____

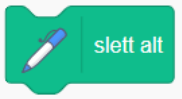
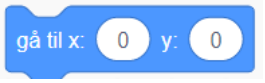
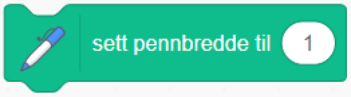
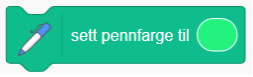
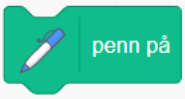
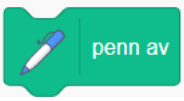
Oppgave 2 : Lag flere former

Prøv å gjøre disse oppgavene – Forklaring til noen kodebrikker du kan bruke er i tabellen nederst.

7. Endre den opprinelige koden du fikk utdelt
   a. Endre fargen på linjene
   b. Endre lengden på linjene
8. Lag en ny kode der du tegner en trekant, tips: Du må endre vinkelen på snu kommandoen.
9. Bruk penn opp og penn ned for å tegne en firkant og trekant ved siden av hverandre.
10. Tegn figuren til høyre:



Ekstra oppgave: Tegn en tegning ved hjelp av kode, for eksempel et hus, en person eller liknende.

	Gå fremover		Sletter alt som er tegnet
	Gå bakover		Flytter figuren tile n bestemt posisjon
	Snu til høyre		Endrer bredden på pennestreken
	Snu til venstre		Endrer fargen på pennen
	Løfter opp pennen slik at du kan flytte deg uten å tegne		
	Setter pennen ned slik at du tegner når du beveger deg		

Vedlegg 4: Aktivitetsark B – Løkker.

Aktivitetsark B – Løkker.

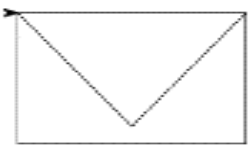
Før du starter åpne linken til programmet under fra Teams og lagre til din egen bruker.

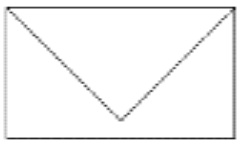
Oppgaver

4. Endre koden slik at den inneholder en løkke
5. Skriv et program som kan lage en femkant med sidelengde 200
6. Skriv et program som kan laget en sekskant med sidelengde 50.

Ekstra oppgave

Lag et program som tegner en konvolutt.





Vedlegg 5: Samtykke skjema

Vil du delta i forskningsprosjektet

«Programmering som del av matematikk faget i skolen»

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å se på programmering i matematikkfaget og hvordan programmering kan brukes som et verktøy for å oppnå matematisk forståelse og problemløsningskompetanse. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
Målet for prosjektet er å belyse hvordan programmering som en del av matematikkfaget kan brukes for å støtte den allerede eksisterende matematikk undervisningen og hjelpe elevene til å oppnå problemløsningskompetanse. Dette prosjektet er den masteroppgave som skrives i samarbeid med NTNU Institutt for lærerutdanning

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

Hvorfor får du spørsmål om å delta?
I prosjektet ønsker jeg å se på elever knyttet til mellomtrinnet. Det er for meg naturlig å benytte meg av elever knyttet til skolen der jeg har min arbeidsplass. Derfor får du spørsmål om å delta i dette prosjektet.

Hva innebærer det for deg å delta?
Deltagelse i prosjektet innebærer ikke noe ekstra arbeid for din del. Det vil bli gjennomført undervisningsøkter i matematikk hvor programmering vil spille en hovedrolle i undervisningsøkten. Økten vil bli filmet og elevenes samtaler i økta vil bli tatt opp. Dette for å analysere hvordan elevene arbeider med programmeringen og hvilke problemløsningsstrategier de velger i møte med programmeringen.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg. Elever som ikke deltar i studien vil likevel gjennomføre programmeringsøktene, men vil ikke bli filmet eller bli inkludert i studien på noen annen måte. Disse elevene vil gjennomføre økten i en av parallell klassene, hvor vi ikke har video eller opptaksutstyr til stede.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.
Det er kun Simen Nyutstumoen og Tore Alexander Forbregd som vil ha tilgang til video og lydopptakene som blir tatt opp i forbindelse med prosjektet
Alle opplysninger og filer vil være lagret på en kryptert minnepinne som låses inn på NTNU til prosjektet er over. Videre vil jeg i Masteroppgaven Anonymisere alle opplysninger som blir brukt i selve oppgaven.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?
Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er 31.07.2021 Anonyme data vil oppbevares innelåst på NTNU for eventuelle oppfølgingsstudier.

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

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

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke.
På oppdrag fra NTNU Institutt for lærerutdanning, har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

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

- NTNU institutt for lærerutdanning ved Tore Alexander Forbregd, tore.a.forbregd@ntnu.no
- Simen Nyutstumoen, Simennyu@ntnu.no
- Vårt personvernombud: Thomas Helgesen, thomas.helgesen@ntnu.no

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

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

Med vennlig hilsen

Tore Alexander Forbregd	Simen Nyutstumoen
(Forsker/veileder)	(Student)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *[sett inn tittel]*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta i lyd og videoopptak i en undervisnings økt

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

---

(Signert av prosjektdeltaker, dato)

Simen Nyutstumoen

Programmering og problemløsning i matematikk

NTNU
Kunnskap for en bedre verden