Algoritmisk tenking og undervisningsmodellen PRIMM

Eit studium av algoritmisk tankegang hos elevar når dei programmerer i Scratch

VIVIAN MÆLAND KROGSÆTHER

RETTLEIAR
Kjetil Damsgaard

Universitetet i Agder, 2023
Fakultet for teknologi og realfag
Institutt for matematiske fag

Forord

Denne oppgåva markerer slutten på tre år som deltidsstudent ved Universitetet i Agder. Det har vore tre kjekke og lærerike år, men også tre tøffe år. Særskilt siste året, der kombinasjonen mellom å vere kontaktlærar i 100 % stilling og skrive master på fritida til tider har vore knalltøff. Eg gjorde det ikkje lettare for meg sjølv ved å velje eit emne eg på førehand ikkje meistra, men på ei anna side har eg jobba med noko eg som lærar i matematikk er nøydd til å setje meg inn i, programmering. No er rett nok ikkje Scratch så komplisert, men likevel var det ei føresetnad at eg meistra programmet til ei viss grad med tanke på å lage oppgåver som fungerte for elevane. Det same med algoritmisk tankegang. Eit omgrep eg trudde eg hadde ei forståing for, men som dess meir eg las, dess meir usikker vart eg. Ikkje eingong forskarar definerer omgrepet likt.

At eg no er ferdig trur eg det er mange rundt meg som set pris på. Tusen takk til heile storfamilien, og Knut Inge, no skal du få lov å ta feriedagar når eg har skulefri, eg treng ikkje lenger huset for meg sjølv desse dagane. Takkar og vener og kollegaer som har måtta høyre på all frustrasjonen min, Elise, som har fått mykje ansvar for klassen, og elevar som har bidrege med materiale til studien.

Takk til Kjetil Damsgaard for ærlege og konstruktive tilbakemeldingar! Du har stilt deg disponibel både kvardag og helg, langt meir enn eg venta.

Og sist men ikkje minst, takk for no til alle medstudentar og lærarar ved Lærarspesialiststudiet i Matematikkdidaktikk, UIA. Lykke til vidare, alle saman!

Bremnes, 6. mai 2023

Vivian Mæland Krogsæther

Samandrag

Innføringa av LK20 hausten 2020, gir klare føringar på at algoritmisk tenking og programmering er ein del av kompetansen elevane skal inneha i framtida. Målet med denne studien er å sjå korleis algoritmisk tankegang kjem til uttrykk hos elevar på 8.trinn gjennom programmering, og meir spesifikt korleis algoritmisk tankegang kjem til uttrykk i dei ulike fasane av PRIMM når elevar programmerer i Scratch .

Studien tek utgangspunkt i sosiokulturell læringsteori, då PRIMM-modellen vektlegg diskusjonar omkring tolking av kodar, og overføring av ferdigheit og kunnskap frå det sosiale til det kognitive planet (Sentance, Waite, & Kallia, 2019). Programmet elevane brukar er Scratch. Eit program som også har vist å støtte algoritmisk tenking i ein sosiokulturell kontekst (Jiang, Zhao, Gu, & Yin, 2021).

Studien er ein kvalitativ casestudie av 19 elevar på 8.trinn, og vart gjennomført i to økter på til saman 2,5 timar. I tillegg vart to elevpar tekne ut av klassen for observasjon, der bandopptak vart nytta. Elevpara vart tekne ut kvar for seg, og desse øktene vara om lag ein time kvar med meg til stades. Datainnsamlinga bestod av innlevert arbeid, observasjon og fokusgruppeintervju med bandopptakar.

Resultatet viste at elevane gjennom PRIMM-modellen får øving i algoritmiske konsept og praksisar, som *abstraksjon, behandling av algoritmar, automatisering, dekomposisjon, generalisering, testing og feilsøking, samarbeid* og *evna til å handtere opne problem/uthalding* (Bocconi, Chioccariello, & Earp, 2018)**.** Dette viste seg ikkje berre i sluttfasen der dei endrar kode og lagar kode sjølv, men også i dei første fasane der fokus er på tolking av kodar.

Abstract

The introduction of LK20 in the autumn of 2020 provides clear guidelines that algorithmic thinking and programming are part of the competence the students should possess in the future. The aim of this study is to see how algorithmic thinking is expressed by eighth-grade students through programming, and more specifically, how algorithmic thinking is expressed in the various phases of PRIMM when students program in Scratch.

The study is based on sociocultural learning theory, as the PRIMM model emphasizes discussions about the interpretation of codes, and the transfer of skills and knowledge from the social to the cognitive level (Sentance, Waite, & Kallia, 2019). The program the students use is Scratch. A program that has also been shown to support algorithmic thinking in a sociocultural context (Jiang, Zhao, Gu, & Yin, 2021).

The study is a qualitative case study of 19 eighth-grade students and was conducted in two sessions totaling 2,5 hours. In addition, two student pairs were taken out of the class for observation, where tape recordings were used. The student pairs were taken out separately, and these sessions lasted about an hour each with me present. The data collection consisted of submitted work, observation, and focus group interviews recorded on tape.

The result showed that through the PRIMM model, the students get practice in algorithmic concepts and practices, such as *abstraction, algorithmic thinking, automation, decomposition, generalization, testing and troubleshooting, collaboration* and *the ability to handle open problems* (Bocconi, Chioccariello, and Earp, 2018). This was not only evident in the final phase where the students change code themselves, but also in the first phases where the focus is on interpreting codes.

Innhald

Forord III
Samandrag V
Abstract VII
Innhald IX
1 Innleiing 1
2 Teori 3
2.1 Ulike perspektiv på læring 3
2.2 Programmering i skulen 4
2.3 Computational thinking 6
2.4 Scratch 10
2.5 PRIMM 11
3 Metode 15
3.1 Forskingsdesign 15
3.1.1 Enkeltinstrumentelt casestudie 15
3.1.2 Kvalitativ metode 16
3.2 Elevutval og organisering 17
3.3 Datainnsamling 18
3.3.1 Elevarbeid 18
3.3.2 Observasjon 18
3.3.3 Fokusgruppeintervju og bandopptak 19
3.4 Val av programmeringsspråk 20
3.4.1 Kva er eit programmeringsspråk? 20
3.4.2 Blokkprogrammering 20
3.5 Undervisning 20
3.5.1 I forkant av undervisningsøktene 20
3.6 Oppgåver 22
3.7 Etikk 23
3.8 Kvalitet på studien 23
4 Resultat og analyse 25
4.1 Predict og Run – Forutsei kva som kjem til å skje i koden, og sjekk om hypotesen er rett 26
4.2 Investigate – undersøk koden 36
4.3 Modify – endre koden 39
4.4 Make – lage eigen kode 43

5 Drøfting 61
6 Konklusjon og avsluttande del 67
7 Referansar 69
8 Vedlegg 75
8.1 Vedlegg 1, NSD-skjema 75
8.2 Vedlegg 2, Spørsmål før prosjektet 77
8.3 Vedlegg 3, Oppgåve 2c) 79
8.4 Vedlegg 4, Samtykke 80

1 Innleiing

Den auka digitaliseringa i samfunnet gjer at teknologi og programvare er viktig for folk flest. Ein treng kunnskap for å forstå den digitale verda både for å sjå moglegheiter men og risikoar (Sanne, et al., 2016). I tillegg vil mange av elevane me har i dag vere med på å utvikle teknologi i framtida (Kaufmann & Stenseth, 2021). «Samfunnet endrer seg raskt og det elevene lærer skal være relevant og framtidsrettet. Læreplanene er derfor endret slik at kompetansen elevene utvikler skal kunne brukes også på områder som i dag er ukjent» (Utdanningsdirektoratet, 2021). Bruk av teknologi i matematikkfaget er ikkje nytt. Programmering vart forsøkt innført på 80-talet, utan å lukkast. Bruk av rekneark og dynamisk programvare har gradvis blitt integrert i faget frå slutten av 90-talet og utover, og er no noko elevane vert prøvd i til eksamen (Det kongelige kirke- utdannings- og forskningsdepartement, 1996; Utdanningsdirektoratet, 2006).

At samfunnet endrar seg har påverka læreplanar i mange land, og dei siste åra er programmering innført i grunnopplæringa. Måten ein integrerer programmeringa på er derimot ulik (Bråting & Kilhamn, 2021).

I Noreg fekk matematikkfaget hovudansvaret for opplæringa i programmering gjennom LK20. Nytt i faget var mellom anna at programmering skal inn frå og med 5.trinn. Resten av grunnskuleløpet finn ein spesifikke kompetansemål der ein både skal lære programmeringsomgrep, og der ein skal kunne nytte programmering som eit verktøy for å løyse problem (Utdanningsdirektoratet, 2020).

Under kjerneelementet *utforsking og problemløysing* står det «Algoritmisk tenking er viktig i prosessen med å utvikle strategiar og framgangsmåtar for å løyse problem og inneber å bryte ned eit problem i delproblem som kan løysast systematisk» (Utdanningsdirektoratet, 2020). Gjennom å jobbe med algoritmisk tenking kan ein lære tankegang og konsept som ligg til grunn for mykje av informasjonsteknologien, mellom anna programmering (NOU, 2020, s.43).

Dersom intensjonane om at norske elevar skal lære både å programmere og å nytte programmering som eit verktøy for å løyse problem skal realiserast, må lærarar i grunnskulen inneha denne kompetansen (Bueie, 2019, s.28). Våren 2022 var me tre studentar frå lærarspesialiststudiet ved Universitetet i Agder som undersøkte korleis det stod til med programmeringskunnskapen hos lærarar på skular i ulike delar av landet. Resultatet var nedslåande, men ikkje overraskande. 73% av lærarane i undersøkinga følte seg ikkje kompetente til å undervise i programmering, inkludert her var også nyutdanna lærarar. Dette kjem også fram i andre artiklar (NRK- innlandet, 2021). Konsekvensen var at undervisninga bestod av ferdige kurs frå a til å, kurs ein fann på nettstadane til

ulike læreverk, på det digitale læreverket Kikora eller gjennom Kodetimen. Verken elevar eller lærarar hadde problem med gjennomføring av kurs, men tilbakemeldinga var at ein etter fullenda kurs mangla forståing for programmering, og at ein dermed ikkje klarte å løyse oppgåver på eiga hand.

Dersom det er slik fleire lærarar melder, at elevar og lærarar ikkje klarar å bruke programmering som eit verktøy for å løyse problem etter å ha jobba med programmeringskurs, korleis kan ein då jobbe med programmering for å fremje forståing og algoritmisk tankegang? Korleis kan ein førebu elevane på ei digital framtid i stadig endring?

Sjølv om eg ikkje var inkludert i denne undersøkinga, kunne ein av desse 73% like gjerne vore meg. Ein lærar utan kompetanse i programmering, og dermed ein lærar som ikkje visste anna enn å undervise ved hjelp av ferdiglaga kurs. I løpet av studien las eg forsking på undervisning ved hjelp av modellane Use – Modify – Create (UMC) og Predict – Run – Investigate – Modify – Make (PRIMM), meir om desse i avsnitt 2.4 og 2.5, og klara ikkje heilt å leggje det i frå meg. Kan metodikk som dette gjere det lettare å undervise i og ved hjelp av programmering for ein lærar med lite erfaring? Kan elevane gjennom denne metodikken jobbe med kjerneelement og kompetansemål i LK 20 allereie frå startfasen av opplæringa?

Med utgangspunkt i ovanståande ynskjer eg dermed å observere elevar som jobbar med programmeringsoppgåver i Scratch, der undervisninga er lagt opp etter undervisningsmodellen PRIMM.

Forskingspørsmålet mitt vert som følgjer:

Korleis kjem algoritmisk tankegang til uttrykk hos elevane i dei ulike fasane av PRIMM?

Først vil eg presentere aktuell teori knytt til forskingsspørsmålet. Vidare skriv eg om metoden eg har nytta, før eg i kapittel 4 og 5 analyserer og drøftar resultata. Siste kapittelet er ein konklusjon på forskingsspørsmålet og tankar om vidare arbeid.

2 Teori

I dette kapittelet vil sentral teori bli presentert. Først vil eg seie litt om ulike perspektiv på læring, og kvifor dette studiet er knytt opp mot sosiokulturell læringsteori. Vidare ser eg på utviklinga av programmering i skulen dei siste 40 åra og forklarar kva som ligg i *computational thinking*, eller på norsk oversett til *algoritmisk tankegang* (Bocconi, Chioccariello, & Earp, 2018, s.8), eller *algoritmisk tenkning* (Utdanningsdirektoratet, 2019). Til slutt kjem eg inn på korleis tidlegare forsking knyter bruk av Scratch og PRIMM opp til å jobbe med utvikling av algoritmisk tankegang hos elevane.

2.1 Ulike perspektiv på læring

Tradisjonelt har det sentrale i matematikkfaget vore å lære seg metodar. I seinare tid, og ved innføring av ny læreplan, LK20, har prosessen fått ein sterkt auka fokus. I staden for å leggje mest vekt på produktet, som å lære algoritmen for divisjon, har prosessen i faget vorte framheva (Skott, Skott, Jess, & Hansen, 2019). Å ha matematisk kompetanse er kjenneteikna ved "å ha viten om, å forstå, utøve, anvende og kunne ta stilling til matematikk og matematisk virksomhet i et mangfold av sammenhenger" (Røsseland, 2005a, s 14). Matematisk kompetanse er altså noko meir enn å arbeide med ferdige algoritmar, og gjennom å arbeide med fokus på både produkt og prosess vil desse støtte kvarandre fram mot denne kompetansen (Skott et al.,2019). I min studie vert prosessen sentral. Det blir ikkje undervisning med oppskrift i korleis programmere, for så å bruke dette i oppgåveløysing. I staden blir elevane aktive i læringsprosessen. Tankane mine er at dei jobbar med å styrkje algoritmisk tankegang gjennom oppgåveløysing, samstundes som dei utviklar kompetanse i programmering.

Det finst fleire definisjonar på kva læring er, men det som synast å gå att er at det er ei endring hos den lærande gjennom ein prosess. Sjølv om definisjonane er relativt samanfattande, er synet på korleis ein lærer ulikt. Læring som tileigning og læring som deltaking er to syn som stammar frå Lave og Wenger, og som i matematikkdidaktikk vart vidareutvikla av Sfard (Skott et al.,2019). Den (radikale) konstruktivismen høyrer til under tileigningsmetaforen. Ein kan ikkje ta i mot kunnskap passivt, men må sjølv konstruere og byggje denne opp. Alle konstruerer si forståing ut frå dei erfaringane ein har, og etter kvart som ein har fått ei individuell forståing kan ein jobbe saman med andre.

Deltakingsmetaforen har eit anna perspektiv, og ligg til grunn for studien min. Gjennom å delta i sosiale samanhengar vil ein gradvis lære. I starten er ein berre observatør eller marginalt deltakande, men etter kvart kan ein gå frå å engasjere seg sosialt til å delta på eiga hand. Dette blir omtala som

the social turn, og har fått meir plass dei siste par tiåra (Skott et al.,2019). Inn under dette kjem sosiokulturell læringsteori. Teorien byggjer på Vygotskij, som hevdar at utviklinga til mennesket har utgangspunkt i miljøet ein veks opp i (Postholm & Moen, 2018) og at barn lærer gjennom å delta i aktivitetar som ligg innanfor deira *Zone of Proximal Development* (ZPD) (Skott et al.,2019). ZPD kan omsetjast til den *næraste utviklingssona*, og beskriv dei oppgåvene som er for vanskelege for den einskilde å klare åleine, men som ein kan klare med støtte frå andre. Seinare er dette også omtala som *schaffolding,* eller stillasbygging (Wood, Bruner, & Ross, 1976). Mediering er eit viktig omgrep her. Det er ikkje berre sanseinntrykk frå omgjevnadane mennesket responderer på, men og reiskap og symbol for å tilarbeide og forstå desse inntrykka (Kozulin, 2003). I denne studien er medieringa sosial gjennom diskusjonar med medelevar, symbolsk gjennom programmeringsspråket og fysisk gjennom digital teknologi (Sentance, Waite, & Kallia, 2019).

Vygotskij såg på språket som avgjerande for at barnet skal ta del i sosialt samspel. Dette kan skje både i form av samtale med andre, men og i indre dialogar. Språket utviklar seg frå rein kommunikasjon i starten, via egosentrisk tale der ein snakkar med seg sjølv, til den indre talen eller tankane (Imsen, 1984, s.210). I studien min er dette sentralt då programmeringsspråket er eit symbolspråk. Ved å nytte PRIMM som metode får elevane kodar dei gjerne ikkje forstår i starten, men ved å samarbeide om oppgåver og diskutere med andre i klassen vert programmeringsspråket eit språk dei etter kvart kan utvikle til tankestadiet, og dermed bruke aktivt for å løyse problem på eiga hand (Sentance et al., 2019). «Through artefacts existing in the social plane that are shared by learners, we are able to use language to understand them and thus internalize that understanding over time» (Sentance et al., 2019, s.145).

2.2 Programmering i skulen

Matematikkfaget var eitt av faga som fekk ansvar for programmering i skulen gjennom LK20. Programmering i skulen er derimot ikkje nytt. Seymour Papert gav ut boka *MINDSTORMS: Children, computers, and powerful ideas* i 1980, og programmeringsspråket han arbeida med, LOGO, kom så tidleg som i 1967. LOGO var eit programmeringsspråk der ein skulle styre ei skjelpadde ved hjelp av kodar. Papert var opptatt av å skape gode læringsmiljø for barna for å stimulere for læring, og han meinte at LOGO var med på å gi barna eit slikt læringsmiljø; ei mikroverd der ein kan lære (Papert, 2020, s.12). Gjennom forsøka sine oppdaga Papert at unge elevar, også elevar med ulike utfordringar, oppnådde gode resultat, og at dei var i stand til å programmere relativt avanserte algoritmar (Papert, 2020, ss.13,16). I dei fleste undervisningssituasjonane der datamaskinar var i bruk, vart denne brukt til å lære elevane framgangsmåtar, jobbe med oppgåver på ulike nivå, gi

tilbakemeldingar og informasjon. Logo, derimot, hadde ei motsett rolle i følgje Papert. I arbeidet med å programmere maskinen, lære han å tenkje, krev det at elevane utforskar si eiga tenking. «Thinking about thinking turns the child into an epistemologist, an experience not even shared by most adults." (Papert, 2020, s.19).

Forsking viste at kompetansen elevane fekk gjennom programmering i lita grad lot seg overføre til andre område og problemstillingar (Dolonen, Kluge, Litherland, & Mørch, 2019) , og saman med kompliserte programmeringsspråk, manglande kompetanse blant lærarar og lite tilgjengeleg utstyr vart ikkje forsøket med programmering nokon suksess.

Også i Noreg har programmering tidlegare vore mål i læreplanar. Mønsterplanen for grunnskulen, M87, omtala spesifikke område som eigna seg godt for datamaskin, som løysing av likningar og berekning av areal (Norge Kirke-og undervisningsdepartementet, 1987). Likevel fekk programmeringa inga sentral rolle i matematikkfaget, og i L97 og K06 er ikkje programmering vektlagt i det heile. I L97 står det at elevane må få møte ulik teknologi, og lommereknar og datamaskin er nemnt som hjelpemiddel til å forenkle arbeid og til presentasjon. Rekneark vert sett på som eit godt verktøy i matematikk, slik tekstbehandling er det i andre fag (Det kongelige kirke-utdannings- og forskningsdepartement, 1996). I K06 står det under grunnleggjande ferdigheitar at elevane skal kjenne til, bruke og vurdere digitale verktøy til berekningar, problemløysing, simulering og modellering (Utdanningsdirektoratet, 2006).

Digitutvalget vart oppnemnd i statsråd 24.juni 2011 og kom med ei utreiing 7.januar 2013 (NOU, 2013 : 2). I oppsummeringa viser dei uro for den digitale kompetansen i befolkninga, og kjem med framlegg om å innføre programmering som valfag i grunnskulen. «Ved kun å lære elevene hvordan de skal bruke digitale medier for å formidle og kommunisere, oppdrar vi i praksis barn og ungdom til konsumenter og videreformidlere, ikke skapere av digitale tjenester» (NOU, 2013 : 2, s. 105).

Omgrepa *Algorithmic Thinking* og *Computational Thinking* vart første gang nemnt i politiske dokument som førebuing til aktivitetar i skulen i 2016 (Bocconi, et al., 2022), og i 2017 la regjeringa ut ein ny digital strategi for både grunnskule og vidaregåande skule. Alle elevar skulle få opplæring i teknologi, som mellom anna inneber å forstå og å handtere algoritmisk tankegang og programmering (Norstein & Haara, 2021, s.74) . Resultatet av den nye strategien vart kjerneelement og kompetansemål i LK20. Argumenta er mellom anna at programmering kan medverke til auka digital kompetanse i befolkninga og at den overlappar kritisk tenking i fagfornyinga. Dessutan er det ei naturleg del av problemløysing i ingeniørfaga (Dolonen, 2019). «Programming/coding provides a laboratory for teaching and learning Computational Thinking – it makes CT concepts concrete» (Bocconi, et al., 2022, s.28). Programmeringsspråka har utvikla seg, utstyr er tilgjengeleg for alle, og

fokus på 21.century skills, som kreativitet, kritisk tenking og problemløysing (Gretter & Yadav, 2016) har vore medverkande til dette.

2.3 Computational thinking

Det siste tiåret har *computational thinking* (CT) vore eit sentralt tema innanfor pedagogisk forsking og praksis. CT kan førast tilbake til Seymour Papert, men vart teke i bruk som eit omgrep av Jeanette Wing i 2006 (Shute, Sun, & Asbell-Clarke, 2017). Ho argumenterte for at CT måtte undervisast i skulen på lik linje med rekning, lesing og skriving (Wing, 2006, s.33). «Computational thinking involves solving problems, designing systems, and understanding human behavior by drawing on the concepts fundamental to computer science» (Wing, 2006, s.33).

I Noreg har ein omsett *computational thinking* til *algoritmisk tankegang* (Bocconi et al., 2018) eller *algoritmisk tenkning* (Utdanningsdirektoratet, 2019). «Å tenke algoritmisk er å vurdere hvilke steg som skal til for å løse et problem, og å kunne bruke sin teknologiske kompetanse for å få en datamaskin til å løse (deler av) problemet. I dette ligger også en forståelse av hva slags problemer/oppgaver som kan løses med teknologi og hva som bør overlates til mennesker» (Utdanningsdirektoratet, 2019).

Figur 1, henta frå (Utdanningsdirektoratet, Algoritmisk tenkning, 2019).

Omsettinga til *algoritmisk tenking* kan vere misvisande sidan det kan forvekslast med det engelske *algorithmic thinking,* som seier noko om evna til å arbeide med algoritmar og lage stegvise instruksjonar for å løyse eit problem (Kaufmann & Stenseth, 2021). *Algorithmic thinking* er altså berre ein del av CT, og kan føre til at den norske omsettinga kan tolkast for snevert (Gjøvik & Torkildsen, 2019).

Plakaten (figur 1) er tilpassa frå Barefoot Computing, UK, og forklarar ikkje berre kva som ligg i CT, men får og fram sentrale arbeidsmåtar for den algoritmiske tenkjaren. Algoritmisk tenking er ein problemløysingsmetode som inneber at ein nærmar seg problem på ein systematisk måte. Dette gjeld både når ein formulerer kva ein skal forsøkje å løyse og når ein kjem med forslag om løysingar (Utdanningsdirektoratet, 2019). Ein må kunne analysere informasjon, bryte ned komplekse problem til delproblem som lar seg løyse og lage framgangsmåtar eller algoritmar for å løyse desse. Vidare må ein lage modellar av den verklege verda ved å fjerne detaljar som ikkje trengst. Vurderingar må gjerast undervegs, og løysingane kan ofte generaliserast og brukast til å løyse andre problem (Utdanningsdirektoratet, 2019). I tillegg til omgrep som logikk, algoritmar, dekomposisjon, mønster, abstraksjon og evaluering, legg utdanningsdirektoratet vekt på arbeidsmåtar som kjenneteiknar den algoritmiske tenkjaren. Ein må utforske og eksperimentere, og vere skapande og open for alternative løysingar. Å gjere feil er ein viktig del av prosessen. Feila må oppdagast og rettast, og ein må prøve på nytt og på nytt. Samarbeid og deling vert sett på som gode arbeidsmetodar i denne prosessen (Utdanningsdirektoratet, 2019).

Det er mange ulike definisjonar på kva CT er, og desse utviklar seg etter kvart som ny forsking kjem til (Shute et al.,2017; Brennan & Resnick, 2012; Bocconi, et al., 2022). Wing (2006) la vekt på at CT ikkje er det same som å tenkje som ein datamaskin, men korleis menneske løyser problem. «Equipped with computing devices, we use our cleverness to tackle problems we would not dare take on before the age of computing and build systems with functionality limited only by our imaginations» (Wing, 2006, s.33). Sjølv om CT vert sett i samanheng med programmering og bruk av teknologi er CT ein tankeprosess uavhengig av teknologi, og ein problemløysingsmetode som inneber evne som å kunne designe løysingar som kan utførast av ein datamaskin, eit menneske eller ein kombinasjon av begge (Bocconi, Chioccariello, Dettori, Ferrari, & Engelhardt, 2016). Programmering er såleis berre ein av fleire moglege verktøy til å utvikle CT.

> Programming is … seen as an especially useful platform for teaching CT since it brings together several of the elements – both concepts and practices – that are central to CT… Even so, it is important to observe that CT involves problem solving and thinking competencies that can be invoked in settings outside of programming. Programming, although important, cool, interesting and fun, is but one of the possible vehicles for developing CT competencies. (Grover & Pea, 2017, s. 28)

Fleire forskarar har belyst kva CT inneber og komme med forslag til rammer for korleis ein kan forstå omgrepet (Brennan & Resnick, 2012; Grover & Pea, 2017). Shute et al. (2017) har undersøkt om lag 70 ulike forskingsartiklar og publikasjonar relatert til CT, og basert på denne gjennomgangen finn ein at forskarar har komme fram til samanfallande ferdigheiter knytt til CT; *dekomposisjon, abstraksjon*, *algoritmar* og *feilsøking*. Shute et al. (2017) foreslår i tillegg til desse fire *iterasjon* og *generalisering*. Om lag det same blir lagt vekt på hos Bocconi et al. (2018) i den nordiske tilnærminga til CT. *Abstraksjon*, *algoritmisk tenking*, *automasjon*, *dekomposisjon* og *generalisering* er nemnt, relatert til ferdigheitar eller praksisar som *berekning*, *testing* og *feilsøking*, *samarbeid* og *kreativitet* og *evna til å handtera opne problem* (Bocconi et al., 2018). Uthalding medan ein jobbar med utfordrande oppgåver er også inkludert i Weintrop et al. (2015) si oversikt.
I følgje Csizmadia et al. (2015) er *logic* og *logical thinking* overordna konsepta i algoritmisk tankegang. Det er til stades når ein jobbar med algoritmar, dekomposisjon, generalisering, abstraksjon og evaluering.
At ulike forskarar nyttar ulike omgrep, og i varierande grad inkluderer både konsept, ferdigheitar og perspektiv, gjer at det kan vere komplisert å fa tak i kjernen på CT. Kva vil det seie å tenkje algoritmisk? For å gjere observasjon og analyse meir oversiktleg i denne studien, tar eg utgangspunkt i det som Bocconi et al. (2018) definerer som kjerneferdigheitar i CT; *abstraksjon, behandling av algoritmar, automatisering, dekomposisjon* og *generalisering*. I tillegg tek eg med *testing og feilsøking*, som er relatert til som ein ferdigheit eller praksis innan CT (Bocconi et al., 2018). Når eg nedanfor prøver å beskrive kva som meinast med dei ulike konsepta, inkluderer eg element frå forskarar nemnt tidlegare i dette avsnittet der eg forstår konsepta er samanfallande.

Abstraksjon (Abstraction)

I følgje Wing (2014) er datavitenskap automatisering av abstraksjonar. Dermed blir abstraksjon svært viktig i algoritmisk tankegang. Abstraksjon handlar om å forenkle. Kva er naudsynt å ta med, og kva kan utelatast? Løysing av komplekse problem blir lettare om ein reduserer detaljar som ikkje er naudsynte og fokuserer på det som betyr noko (Bocconi et al,2016). Abstraksjon inneber å kjenne

att mønster, å generalisere, og å la ein gjenstand stå for mange. Det inneber å lage algoritmar som tar inndata, utfører ulike trinn, og produserer utdata. Ein brukar med andre ord «black-boxing» for å skjule detaljar slik at ein kan fokusere på inndata og utdata (Grover & Pea, 2017). Abstraksjonar ein gjer vert automatisert, og datamaskinen utfører ønska handling kjapt og presist (Wing, 2014).

Behandling av algoritmar (Algorithmic thinking)[1]

Behandling av algoritmar er å setje opp steg for steg korleis ein kan nå eit mål eller løyse eit problem. Denne samlinga av steg blir kalla ein algoritme (Grover & Pea, 2017). Matoppskrifter og ei beskriving av korleis ein kjem seg frå klasserommet til busshaldeplassen er døme på kvardagsalgoritmar, og algoritmar for multiplikasjon er eit døme frå klasserommet. Eit program i til dømes Scratch er ein algoritme, og kvar linje i koden står for kvart steg. Når ein behandlar algoritmar innan datavitenskap, må ein vera svært presis (Grover & Pea, 2017). Algoritmane ein lagar må fungere i alle liknande tilfelle, slik at når algoritmen først er forstått treng han ikkje lagast frå botnen av for kvart nytt problem (Csizmadia et al., 2015)

«Algorithmic thinking is the ability to think in terms of sequences and rules as a way of solving problems or understanding situations. It is a core skill that pupils develop when they learn to write their own computer programs." (Csizmadia et al., 2015, s. 7). Hos Brennan & Resnick (2012) vert dette omtala som *sequences*.

Automatisering (Automation)

Automatisering sparar oss for arbeid, då datamaskinen kan utføre eit sett med oppgåver kjapt og effektivt samanlikna med om eit menneske skulle ha utført same operasjon (Lee, et al., 2011). Utfall av 1000 myntkast kan vera eit døme på dette. Det er også viktig at elevane utviklar ei forståing for når det er smart å bruke automatisering for å løyse eit problem framfor at menneske løyser det utan bruk av maskinar (Grover & Pea, 2017). Hos Brennan & Resnick (2012) kjem dette til uttrykk gjennom *events*, der når ein til dømes trykkjer pil opp, så går Scratch-katten bortover samtidig med at den seier noko.

Dekomposisjon (Decomposition)

CT er ein problemløysingsmetode (Wing, 2006), og ved å bryte ned problem til mindre delproblem blir løysingsprosessen meir handterleg (Grover & Pea, 2017). Når ein jobbar med programmering

[1] Behandling av algoritmar er mi omsetting av Gjøvik og Torkildsen (2019) sitt omgrep algoritmebehandling. Algoritmebehandling erstattar det engelske omgrepet *Algorithmic Thinking*, for å unngå å blande saman med algoritmisk tankegang.

bryt ein gjerne ned problemet til bitar av kodar som vert skrivne separat. Vidare skal desse koplast saman for å få det utfallet ein ynskjer. Den første delen kan vere grei å gjennomføre, men nybyrjarar får ofte problem i samankoplinga. Ei samankopling går gjerne ikkje rett fram, men må vevast saman for at programmet skal fungere (Grover & Pea, 2017). Ein treng altså ikkje berre ferdigheit til å bryte ned, men også til å byggje opp att.

Generalisering (Generalization)

Generalisering er kopla til det å kjenne att mønster og sjå samanhengar. Slik kan ein nytte tidlegare løysingar og erfaringar til å løyse nye problem. Algoritmar laga for å løyse spesifikke problem kan tilpassast og vidareutviklast til å løyse ein heil klasse av liknande problem (Csizmadia, et al., 2015).

Testing og feilsøking (Debugging)

Ting går (normalt) ikkje som ein har tenkt, og ein må utvikle strategiar for korleis ein tar tak i problem som oppstår. Dette kan vera prøving og feiling, å overføre frå tidlegare arbeid eller å få støtte frå andre (Brennan & Resnick, 2012).

Problemløysingsprosessar inneheld evaluering om resultatet er effektivt, nøyaktig og fungerer til det som er tenkt. Det handlar også om å oppdage feil og rette opp i desse (Grover & Pea, 2017).

2.4 Scratch

I min studie har eg valt at elevane skal programmere ved hjelp av Scratch. Meir om dette i avsnitt 3.6.

Internasjonal forsking (Fagerlund, Häkkinen, Vesisenaho, & Viiri, 2020) har vist at Scratch kan vere med å utvikle algoritmisk tankegang hos elevar. Det både fremmar og gjer synleg evna til algoritmisk tenking. Scratch er eit blokkbasert programmeringsspråk som er mykje brukt i grunnskulen (Utdanningsdirektoratet, 2022). Det er utvikla for å gjere programmering lett tilgjengeleg for alle (Resnick, et al., 2009), noko som gjer at inngangsterskelen er låg. Ein får språket på norsk og det krev ikkje at ein hugsar syntaksen for det ein ynskjer å gjere. Ein slepp også problem med skrivefeil, og Scratch kan dermed fungere som *scaffholding* (Wood, Bruner, & Ross, 1976). At barna kan bruke blokker, og manipulere desse, gjer at fokus kan liggje på algoritmisk tankegang, og ikkje på å hugse korleis koden skal skrivast. (Haraldsrud, Sveinsson, & Løvold, 2020). Målet bør vere å lære matematikk gjennom programmering, og ikkje berre lære å skrive kode (Gjøvik & Torkildsen, 2019). Scratch er ikkje utvikla kun for å lære barn koding, men er eit godt verktøy som gjer moglegheiter for samarbeid gjennom *Scratch online community* . Her kan dei lære å kode saman med andre, kommentere andre sitt arbeid og manipulere ferdige kodar (Resnick, et al., 2009). Læring skjer dermed i eit samarbeid med andre i eit stadig større samfunn, og støttar utvikling av algoritmisk

tenking i ein sosiokulturell kontekst (Jiang, Zhao, Gu, & Yin, 2021). Resultat av forsking har vist at studentar som fekk samarbeide om programmeringsoppgåver oppnådde signifikant høgare CT-skåre enn elevar som jobba åleine, og at samarbeid var særskilt gunstig for studentar med minimal erfaring innan programmering (Shute et al., 2017).

Studiane til Fagerlund et al (2020) koplar aktivitetar i Scratch til mellom anna abstraksjon, logikk, mønster, dekomponering og algoritmar, omgrep ein finn att i utdanningsdirektoratet si oversikt over den algoritmiske tenkjaren (Utdanningsdirektoratet, 2019), og som kjerneelement i CT (Bocconi et al., 2018). Det kjem også fram at om ein skal få best mogleg bilete av evnene til algoritmisk tenking bør ein studere elevane sine programmeringsoperasjonar framfor kodeprosjektet, prosessen blir dermed viktig (Universitetet i Stavanger, 2022). «Due to its close relationship with computing and programming, CT skills appear to be improved via computational tools, such as Scratch." (Shute et al., 2017).

2.5 PRIMM

Teorien om PRIMM er relevant for denne studien fordi han forklarar korleis ein kan leggje opp undervisning i programmering med ein plan for progresjon, og fordi han vektlegg kunnskapen om å tolke kodar. Undervisningsmodellen er influert av Vygotskij og den sosiokulturelle læringsteorien, der overføring av ferdigheitar og kunnskap frå det sosiale til det kognitive planet skjer gjennom språk og mediering (Sentence et al., 2019).

Programmering i skulen handlar om å gi elevane eit verktøy for å løyse oppgåver og problem, i tillegg til å forstå datamaskinar og få kompetanse i å bruke digitale verktøy. Dette fell inn under omgrepet computational thinking (Sevik & m.fl., 2016). Lee et al. (2011) foreslår ein undervisningsmodell for korleis ein kan engasjere elevane i CT, og kjem opp med ein modell i tre fasar: *Use, Modify, Create* (UMC). Elevane får ein ferdig kode dei skal utforske (use) før dei går vidare med å endre koden for å få den til å passe sitt design (modify). Utfordringane blir meir og meir komplekse, til ein til slutt kan lage sine eigne program (create). Denne progresjonen støttar elevane i utviklinga av CT, og inngangsporten til programmering blir lågare for både elevar og lærarar (Lytle, et al., 2019).

PRIMM er ei vidareutvikling av UMC (Sentance et al., 2019, s.148), og har fokus på korleis lærarar kan forstå læringa til elevane og få til praktiske og effektive strategiar ved programmering i klasserommet. I staden for at elevane lærer å programmere før dei kan lese program, brukar ein tid på å snakke om korleis og kvifor programma fungerer før ein redigerer og lagar eigne program. Elevane lærer altså å uttrykkje seg munnleg om programmeringsomgrep og –prinsipp. Use-delen i UMC er med andre ord vidareutvikla til *Predict, Run* og *Investigate.*

Noko av innhaldet er brukt i ei oppgåve levert på UIA våren 2022 (Haagensen, Krogsæther, & Tennfjord, 2022).

Predict (forutsei)

Elevane får sjå ein ferdig kode dei saman skal diskutere, før dei lager ei hypotese av kva som skjer når koden blir kjørt. Dette trinnet kan vere ein *startar*, eller ein kan nytte ei heil arbeidsøkt. Elevane kan jobbe åleine, i par og/eller ein kan ha diskusjonar i heil klasse.

Run (kjør)

Etter at hypotesen er laga testar ein denne ved å kjøre koden. Elevane lastar ned den ferdige koden slik at ein slepp problem med tanke på å skrive inn feil kode. Vidare kan ein diskutere om hypotesen var rett, eller kva det eventuelt var som gjorde at ein tenkte feil.

Investigate (undersøk)

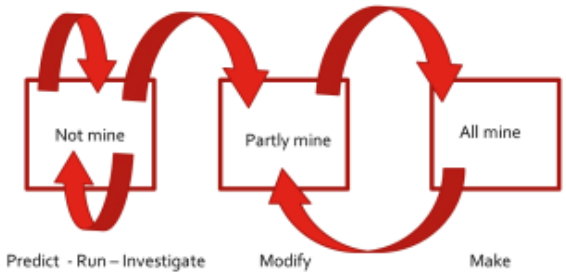
Denne fasen inneheld undersøkjande og utforskande aktivitetar med støtte frå den eksisterande løysinga. Feilsøking, setje kodelinjer i rett rekkjefølgje og finne ut kva som skjer om ein endrar på, eller tek bort, blokker eller verdiar er døme på slike aktivitetar.

Modify (endre koden)

Elevane endrar funksjonen til koden med stadig meir utfordrande oppgåver. Utgangspunktet er framleis ein ferdig kode. I denne fasen går koden over frå å vere andre sin til å vere delvis eigen kode. Slik kan ein få auka tru på seg sjølv.

Make (lage kode)

I denne fasen lagar elevane eigne program. Dei kan bruka same struktur som tidlegare, men kodane løyser andre problem. Kodane blir dermed eigne.



Figur 2. Modell for PRIMM (henta frå: https://primmportal.com/)

Ved å jobbe seg gjennom dei fem fasane på denne måten går ein frå å arbeide med oppgåver ein klarar på eiga hand, via oppgåver innanfor ZPD der støtta kjem frå læringspartnar, lærar og ferdige kodar ein kan manipulere, før ein til slutt kan arbeide uavhengig for å nå læringsmål (figur 2). I følgje Sentance et al (2019, s.170) viser testar at PRIMM gir auka læringsutbytte for elevane, og lærarane melder om ein undervisningsmodell som tilbyr rutine og læringsutbytte også for elevar som elles presterar lågt. Dei ser også at elevane får ei rikare forståing for kjernen i programmering. Sidan programmering i skulen ikkje lenger berre er for spesielt interesserte, er desse resultata av interesse.

Lee et al. (2011) og Lytle et al. (2019) kopla UMC til ein metodikk der ein engasjerte elevane i CT. Det er dermed nærliggjande å tru at også PRIMM vil gjere dette sidan det er ei vidareutvikling av UMC-metodikken. I starten får elevane ferdige kodar dei skal diskutere. På dette stadiet tenkjer eg at elevane får ei øving i å lese og forstå algoritmar og sjå korleis problem kan brytast ned i mindre delar. Dersom hypotesen elevane lagar viser seg å vere feil, kan dei få øving i abstraksjon og feilsøking. Under investigate kan elevane truleg øvast opp i ulike CT-konsept, alt etter korleis oppgåva er formulert, og kva dei undersøkjer. Generalisering kan komme til syne når elevane skal endre ein kode, og lage kode heilt frå botnen av, i tillegg til at dei andre CT-konsepta vil bli jobba med her. Som skrive i innleiinga var eg ikkje nøgd med eiga undervisning når det kom til programmering. Håpet er at eg gjennom å undervise etter PRIMM, skal finne ein måte å undervise på som engasjerer elevane, og at dei vert aktive i eiga læring både med tanke på å styrkje algoritmisk tankegang, og lære å programmere.

3 Metode

Utgangspunktet for denne studien var å finne noko ut om korleis programmering ved hjelp av undervisningsmodellen PRIMM kan brukast for å å få fram algoritmisk tankegang hos elevane. Vidare vil eg gjere greie for kva for metode som vart brukt og kvifor, samt drøfte styrkjer og svakheiter ved denne tilnærminga. I tillegg vil eg gi grunn for planlegging og gjennomføring av datainnsamling til studiet, før eg til slutt ser på etiske utfordringar.

3.1 Forskingsdesign

Sidan eg skal studere korleis algoritmisk tankegang kjem til syne i dei ulike fasane av PRIMM hos ei gruppe elevar på 8.trinn, er det naturleg å ta utgangspunkt i ei kvalitativ tilnærming til datainnsamling gjennom eit casestudie. CT er ein tankeprosess uavhengig av teknologi (Bocconi et al., 2016), og for å ta del i korleis elevane tenkjer er eg avhengig av å kome tett på dei. Eg må prøve å forstå korleis dei tenkjer, noko som ville vore svært vanskeleg å få til i ein kvantitativ studie.

3.1.1 Enkeltinstrumentelt casestudie

Eit casestudie er eit forskingsdesign der ein ynskjer å få ei grundigare forståing innanfor eit avgrensa system (Creswell, 2007, s. 73), og casestudiar vert ofte brukt når ein ynskjer å svare på spørsmål som *korleis* og *kvifor* (Sander, 2022). Eit case kan vere ein einskild person, eller ei gruppe som i denne studien, og tar gjerne utgangspunkt i observasjonar, intervju, lyd-/videoopptak og dokument (Sullivan, 2009). På denne måten kan ein innhente mykje informasjon ut frå ei lita gruppe. Studien gir ikkje alltid eit konkret svar på eit spørsmål, men kan gi indikasjonar ein kan ta i bruk for å prøve å svare på problemstillinga.

Når det kjem til intensjonar, er det tre variasjonar av casestudiar, der mitt studie fell inn under den enkeltinstrumentelle casestudien (Creswell, 2007, s.74). I denne typen studie vert det fokusert på ei problemstilling for så å velje ut eit avgrensa case for å illustrere problemet eller fenomenet. Ved å gjennomføre eit casestudie ynskjer ein dermed å få ei djupare forståing for det ein forskar på, og aktiviteten eller fenomenet er vektlagd framfor personane som deltar i studiet. Dette passar inn med hensikta bak studien min, då denne er å sjå korleis algoritmisk tankegang kjem til syne hos elevar medan dei jobbar etter undervisningsmodellen PRIMM.

3.1.2 Kvalitativ metode

Tradisjonelt skil ein mellom kvantitative og kvalitative datainnsamlingar. Kvantitative datainnsamlingar resulterer gjerne i diagram og tabellar, og når ein refererer dei er det ofte ved bruk av tal (Bogdan & Biklen, 1982). Denne metoden vert gjerne betrakta som deduktiv, der forskaren veit kva ein ser etter og held fast ved dette. Utgangspunktet er ofte tidlegare forsking. Ein kvalitativ forskingsmetode derimot vil i følgje Bogdan & Biklen (1982) eigna seg når ein prøver å forstå menneskeleg oppførsel og erfaring. Ein forskar med andre ord på menneske i ein naturleg kontekst, og det blir eit tett samarbeid mellom forskar og dei som blir forska på. Denne sosiale interaksjonen kan plasserast i ein sosiokulturell kontekst (Postholm, 2004). Ein av fordelane ved å velje kvalitativ metode for min del er at eg kan stille oppfølgingsspørsmål dersom eg er usikker på om eg forstår elevane rett. Slik kan mistydingar unngåast, og eg kan få ei betre forståing for korleis elevane tenkjer.

Sjølv om ein har eit tenkt utgangspunkt, kan gjerne fokusområdet vise seg, eller endre seg, i løpet av datainnsamlinga (Bogdan & Biklen, 1982). Dette vert referert til som induktiv metode (Postholm & Jacobsen, 2019) og datainnsamlinga føregår gjerne gjennom intervju, observasjon og analyse (Bogdan & Biklen, 1982). Ein kvalitativ studie vil alltid bære preg av subjektivitet. Så også i min studie. Erfaringar frå tidlegare, og kjennskap til elevane, vil setje preg på kva eg ser og korleis eg tolkar resultata. Postholm (2004) viser til bilete av ei and eller kanin (figur 3). Om ein ser anda eller kaninen først kjem an på kva for referansar ein har, og slike erfaringar vil og påverke analyse av forskingsresultata. Ho viser også til alle bileta Monet har av parliamentsbygninga i London. Ulike lystilhøve fører til at bileta av same bygning framstår med ulike uttrykk. På same måte kan det ein ser i eine stunda bli oppfatta annleis ei stund etter (Postholm, 2004). Det eg observerer i klasserommet og tolkar på ein måte der og då, kan tolkast annleis når eg seinare sit med analysearbeidet.

Figur 3 henta frå [Kvalitativ forskning på praksis. Fra opprinnelse til forskerfokus (idunn.no)]

Som hovudregel kan ein ikkje generalisere statistisk ved denne type undersøkingar. Likevel bør resultata ha ein overføringsverdi (Larsen, 2017).

Lærar som forskar på eiga undervisning eller eigne elevar vekslar ofte mellom deduksjon og induksjon, og forskingsmetodane vil gjerne utfylle kvarandre (Postholm & Jacobsen, 2019). Når eg studerer algoritmisk tankegang hos elevane medan dei jobbar med programmering, har eg rammeverk og forsking som utgangspunkt. Likevel vil måten elevane arbeider på og tankar og spørsmål som kjem fram i undervisningsøktene, vere avgjerande for kva eg tar tak i under analysedelen, og kva for rammeverk og forsking eg legg til grunn for arbeidet. På denne måten vekslar eg mellom deduktiv og induktiv metode.

3.2 Elevutval og organisering

Som informantar til studiet tok eg utgangspunkt i eine klassen eg underviser i matematikk. Alle fekk invitasjon til å vere med på studiet gjennom NSD-skjema (vedlegg 1), og dei fleste sa ja til å delta. Det var elevar som reserverte seg for bandopptak, men gav løyve til å bruke innleveringar og observasjonar eg gjorde i klasserommet. Nokre takka nei til å delta, eller berre gløymde å ta med underskrifta.

Vidare vart det laga åtte par og ei gruppe med tre elevar ut i frå samtykke og kven eg visste arbeida godt saman. Det vart ikkje teke omsyn til fagleg nivå, då programmering er nytt for dei fleste, og at det i min studie er viktigare at elevane får til ein god dialog, uavhengig av faglege ferdigheitar.

Elevane vart observert i to økter på til saman 2,5 timar, og sat parvis i klasserommet (ei gruppe på tre). Kvart par brukte ein PC, dette med tankar om at diskusjonen elevane imellom ville gå lettare då enn om dei sat med kvar sin PC. To par har fått ekstra fokus gjennom bandopptak. Desse vart valt ut frå kven eg antok var trygge nok til å halde samtalen gåande under opptak. Fokuselevane vart tekne ut av klassen i andre fag enn matematikk, eitt par av gongen. Eg var til stades heile tida, bortsett frå tre-fire minutt med eine paret, då eg vart henta ut av annan elev. Eine paret (elevane K og L) vart utfordra til å jobbe med oppgåve 1e), og andre paret (C og D) oppgåve 2e). Dette ut frå kor langt dei var komne i prosessen då eg tok dei ut. Begge bandopptaka vart transkribert rett i etterkant av øktene, og bilete eg tok undervegs i arbeidet sett inn på rett plass i samtalane deira. Utdrag frå dette vert omtala i analysedelen.

I sjølve forskingsfasen hadde eg ein annan lærar med meg i klasserommet. Elevane fekk beskjed om å ta kontakt med denne læraren om det var noko dei ville spørje om, og så vart det den andre læraren si rolle å ta kontakt med meg ved behov. På denne måten sikra eg at eg kunne ha mest mogleg fokus på observasjonsrolla, utan å bli forstyrra. Dette fungerte godt. To-lærar hadde ikkje erfaring med Scratch, og i følgje seg sjølv ikkje kompetanse innan programmering. Dette var ein fordel med tanke på at hen ikkje kunne hjelpe elevane, og slik påverke resultatet utan at eg fekk det

med meg. Ulempa var at det vart vanskar med felles gjennomgang i klassen. Å inneha ei observatør-rolle på same tid som ein har ein diskusjon i klasserommet er vanskeleg å få til.

3.3 Datainnsamling

Det er mange måtar å samle inn data til forsking. For denne studien vil data seie teikn på algoritmisk tankegang. Eg ser dermed etter oppgåveløysingar eller lyttar etter dialogar mellom elevar som viser teikn til dei ulike kjerneferdigheitene i CT, omtala i avsnitt 2.3. I dette avsnittet vil eg seie litt om korleis eg skal gjere det, og kvifor eg har valt å gå fram på denne måten.

3.3.1 Elevarbeid

Oppgåvene elevane jobbar med ligg på læringsplattforma *It´s learning* som filer dei kan skrive og lime bilete inn i. Dei får også utlevert oppgåvene i eit hefte dersom dei heller ynskjer å gjere det skriftlege arbeidet der, for så å sende inn bilete av løysingsforslaga sine på It´s learning. Elevane vart også oppfordra til å sende inn (alle) løysingsforslag dei hadde som vart feil. Dette for at eg skulle kunne sjå kva dei gjorde feil, og korleis dei gjekk fram for å rette opp i feilen. Fordelen med å ta inn elevarbeid på denne måten, var at eg fekk ei oversikt over heile klassen, ikkje berre dei eg rakk over i observasjonsøktene. Likevel var berre noko av det eg fekk inn eigna til analysedelen. Eg fekk ei oversikt over til dømes kor mange som nytta løkker, men elevane fann det tungvint å sende inn alt arbeidet, så dei fleste sende berre det endelege resultatet, og prosessen mangla.

3.3.2 Observasjon

Observasjon er ein mykje brukt metode for å samle inn data, og observasjon i klasserommet er kjenneteikna av systematikk og målretting. Skal dette bli vellykka må ein ha eit fokus bestemt av problemstillinga slik at ein veit kva ein skal sjå etter, og kvar ein skal sjå etter det (Postholm & Jacobsen, 2019). Ein må altså ha bestemt kva ein skal fokusere på, over kor lang tid det skal gå føre seg, og kva for observatørrolle ein sjølv skal ha. For denne studien er deltakande observasjon (Fangen, 2022) relevant sidan eg har valt å observere ein av klassane eg underviser i matematikk. Dette vart gjort med tanke på tid då det gjer meg moglegheit til å sjonglere med timeplanen for å sikre nok data. Utfordringa er å ikkje bli for involvert som deltakar, slik at observasjonen kjem i bakgrunnen. Sjølv om observasjon kan gi mykje informasjon, må eg også vere klar over feilslutningar. Som skrive i avsnitt 3.1.2, vil subjektivitet og tidlegare erfaringar kunne setje preg på kva eg observerer og korleis dette blir tolka. Saman med det at elevane sin åtferd kan endre seg når dei veit

dei blir observerte, og at eg ikkje får med meg alt som går føre seg i klasserommet, er dette noko eg må vere merksam på (Larsen, 2017).

Gruppene vart observerte i klasserommet medan dei arbeida med oppgåver. Observasjonen gjekk føre seg ved at eg sette meg ned saman med gruppene etter kvart som eg oppdaga god dialog om oppgåvene, og eg fekk såleis tid til å notere og ta bilete av skjermen medan dei jobba. Observasjonen kan dermed seiast å vere open, og ein mellomting mellom ein semistrukturert observasjon med utgangspunkt i kjerneferdigheitar i CT, og ustrukturert observasjon der eg noterte observasjonar av interesse (Larsen, 2017). Dess fleire ein observerer, dess meir kan ein argumentere for at erfaringa ein gjer seg kan overførast til andre (Postholm & Jacobsen, 2019), og ved å setje meg ned med ei og ei gruppe er det stor risiko for å gå glipp av data frå andre grupper. Likevel vel eg å gjere det på denne måten då kvaliteten på det eg får observert sannsynlegvis blir betre.

3.3.3 Fokusgruppeintervju og bandopptak

Kvalitative intervju kan vere meir eller mindre strukturerte (Larsen, 2017). I denne studien er det brukt ustrukturert fokusgruppeintervju av to par, der bandopptak vart gjort. Eg hadde ikkje ferdig formulerte spørsmål, men tok utgangspunkt i oppgåvene elevane jobba med i dei ulike fasane av PRIMM og stilte spørsmål som «Kva trur dykk skjer i denne koden?» og «Klarar dykk å finne ut kva i koden som gjer at ……skjer?». Gjennom denne type intervju kan ein stille oppfølgingsspørsmål, i motsetnad til skriftlege intervju. I tillegg kan ein få tak i informasjon som ikkje så lett kan observerast i oppgåvesvar og ved arbeid i klasserommet, og slik unngå mistolkingar. Det er likevel viktig å vere merksam på at den eller dei som vert intervjua kan vere påverka av situasjonen og meg som intervjuar (Larsen, 2017). Sjølv om elevane har fått beskjed om at svara dei gir ikkje vert vurdert med karakter, er dei likevel klar over at karakteren dei får i faget vert sett av meg.

I denne studien var det naudsynt med bilete av skjermen i tillegg til bandopptak, transkripsjon av opptaket og notatar. Dette for å kunne følgje diskusjonane elevane mellom når eg seinare skulle analysere resultata. Under observasjon i klasserommet vart dette utfordrande då eg som observatør både skulle notere det elevane sa og ta bilete. Under fokusgruppeintervju/observasjon, slapp eg notere det elevane sa, og fokuset vart dermed å ta bilete undervegs i prosessen. Bruk av video vart vurdert, men konsekvensen hadde truleg vore at fleire elevar hadde takka nei til å bli med.

Det vart også stilt spørsmål til elevar medan dei jobba i klasserommet. Også desse gav funn av interesse, men utan å bruke bandopptak er det fare for at ikkje alt blir korrekt notert.

3.4 Val av programmeringsspråk

3.4.1 Kva er eit programmeringsspråk?

Eit programmeringsspråk er eit konstruert språk brukt for å styre og kontrollere datamaskinar. Orda er eintydige, og syntaksen må følgast nøye (Statped, 2021). Det fins mange ulike typar programmeringsspråk, og programmering kan skje både ved hjelp av blokker og tekst.

3.4.2 Blokkprogrammering

LK20 gir ingen føringar for om ein skal bruke blokkprogrammering eller tekstprogrammering, men ser ein i læreverk kan det sjå ut for at blokkprogrammering skal nyttast på barneskulen (http://skolen.cdu.no; http://salaby.no) medan tekstprogrammering er det ein jobbar med på ungdomsskulen (Tofteberg, Tangen, Bråthe, Stedøy, & Alseth, 2021; Kongsnes & Wallace, 2020). Dette kjem også fram i intervju med Morten Munthe (doktorgrad i matematikkdidaktikk ved Norges miljø- og biovitenskapelige universitet (NMBU), lærar ved Oslo handelsgymnasium og jobbar samstundes ved utdanningsetaten i Oslo kommune), som uttalar at han synest synd på lærarar på ungdomsskulen som skal ta den vanskelege overgangen frå blokkprogrammering på barneskulen til tekstprogrammering på vidaregåande skule (TEKNA, 2021).

Det er tredje året me har elevar som kjem til ungdomsskulen etter at ny læreplan tredde i kraft. Erfaringa frå dei to føregåande kulla er at dei har jobba lite med programmering på barneskulen. Med bakgrunn i dette, og i spørsmål til klassen tidleg hausten -22, bestemte eg at klassen skulle jobbe med blokkprogrammering gjennom Scratch i studien min. At det er lett å lage ferdige kodar og leggje ut i Scratch Community er også ein styrke, då desse kan manipulerast av elevane utan risiko for at nokon øydelegg den opphavelege koden (Jiang et al.,2021). Dette er heilt i tråd med PRIMM-metodikken.

3.5 Undervisning

3.5.1 I forkant av undervisningsøktene

Kort tid etter at elevane starta på ungdomsskulen gjorde eg ei undersøking med tanke på om elevane hadde jobba med programmering på barneskulen. Elevane kjem frå seks ulike skular, og erfaringa frå tidlegare er at når nye ting skal innførast, skjer dette ulikt frå skule til skule. Programmering er nytt i læreplanen, og undersøkingar har vist at mange lærarar ikkje føler seg kompetente til å undervise dette enno (Berggren & Jom, 2019) noko som truleg også gjeld her i området. Spørsmåla (vedlegg 2) dreia seg om elevane hadde jobba med programmering tidlegare, og om ja, kva for program dei kjente til. Vidare fekk dei spørsmål om på kva for måte dei hadde

jobba med programmering. Det siste spørsmålet viste fire kodar, og elevane skulle avgjere kven av dei som teikna eit rektangel. Alle spørsmåla hadde alternative svar, og nokre av dei bilete, for å støtte hukommelsen.

Med utgangspunkt i elevane som takka ja til å vere med i prosjektet, var det 13/19 elevar som svara at dei hadde jobba litt med programmering. Ein av elevane hadde aldri programmert tidlegare og to var usikre. Tre av elevane svara at dei hadde jobba litt med koding, men ikkje programmering. Dette utan at eg hadde nemnt ordet koding. 18/19 elevar hadde kjennskap til Scratch, eller i alle fall hugsa å ha sett katten. Fire elevar frå ein skule hadde prøvd Python, og to elevar skreiv at dei hadde kjennskap til micro:bit. På spørsmål 3 var det tretten elevar som hadde jobba med programmering gjennom Kodetimen/Lær Kidsa Koding og ti elevar som svara at dei hadde jobba med programmering gjennom kurs på Kikora eller liknande. Ni elevar skreiv av det lærar skreiv på tavla, medan fire elevar hadde jobba på eiga hand med oppgåver i bok. På siste spørsmålet var det kun fire elevar som ringa rundt begge korrekte svaralternativa og to som hadde ringa rundt eitt av dei. Elles var det tre som hadde heilgardert og like mange som hadde skrive at dei ikkje visste. Ein av elevane hadde ringa rundt eitt rett og eitt feil alternativ. Dei resterande elevane hadde ikkje svara.

Informasjonen frå denne undersøkinga vart lagt vekt på då eg laga oppgåver til sjølve prosjektet, noko eg kjem tilbake til i avsnitt 3.6.

Veka før prosjektet starta snakka me om kva algoritmisk tenking er. Elevane fekk sjå filmklippet «Kva er algoritmisk tenkning?» (Learnlink u.d). Vidare laga dei geometriske figurar på papir med instruks for korleis teikne, for så å få ein medelev til å teikne kun ved hjelp av forklaringa. Me hadde også forklaring av algoritme, vilkår, løkke og variabel gjennom å «programmere» elevane (figur 4). Dette var ein ide lånt av Kristine Sevik, Universitetet i Agder, då ho vitja skulen vår hausten -22.

Figur 4. Dette er ein skjermdump av ei side i ein Power-Point presentasjon eg laga. Figurane var animert, og viste seg i rytmen elevane skulle trampe og klappe.

3.6 Oppgåver

Sidan eg tidleg i skuleåret hadde undersøkt om elevane hadde jobba med programmering på barneskulen (avsnitt 3.5.1), vart det kjapt konkludert med at blokkprogrammering ved hjelp av Scratch var det elevane skulle jobbe med. Sjølv om det berre var ein elev som meinte å vere sikker på at hen ikkje hadde programmert tidlegare, og denne eleven også var den einaste eleven som ikkje kjente att katten i Scratch, vurderte eg ut frå korleis dei tolka koden dei fekk sett i oppgåve 4 (vedlegg 2) at kompetansenivået ikkje var høgt då det kom til programmering.

Elevane fekk to oppgåver å jobbe med, inndelt frå a) til e) ut frå stega i PRIMM. Oppgåvene vert presentert undervegs i analysedelen, bortsett frå oppgåve 2c)[2] (vedlegg3). Utgangspunktet var at første oppgåva skulle ha fokus, men sidan nokre elevar hadde jobba litt med Scratch tidlegare laga eg ei litt meir utfordrande oppgåve 2. Dette sikra også at ingen gjekk tom for arbeid medan observasjonen gjekk føre seg.

Temaet for oppgåvene var geometri. Sidan fokus var algoritmisk tankegang, og elevane mangla eller hadde lite kompetanse innan programmering, vart det viktig at dei fekk jobbe med matematikk dei hadde kjennskap til. Utfordinga mi vart å lage gode oppgåver, sidan min eigen kompetanse er mangelfull innan programmering. Progresjonen i oppgåve 1 var å gå frå å programmere eit kvadrat, via løkker og til slutt andre geometriske figurar. I oppgåve 2 skulle elevane ta med seg det dei hadde lært frå oppgåve 1, og gå vidare med rotasjon og endring av fargar gjennom å lage mønster. I ettertid ser eg at oppgåve 1 fungerte bra. Elevane vart utfordra, men oppgåvene var ikkje så vanskelege at dei gav opp. Oppgåve 1e) der elevane skulle lage ein kode som teikna ein trekant, og deretter femkant og sekskant, kunne vore vidareutvikla til å jobbe med variablar, og kanskje det kunne vore fokus på oppgåve 2 i staden for mønster og rotasjon. Oppgåve 2 vart endra mange gongar undervegs, der eg først hadde fokus på at oppgåve 2 skulle ende ut i eit mønster av geometriske figurar der både rotasjon og endring av farge var krav. Så tok eg bort ordet rotasjon og kravet om endring av farge. Eg endra ikkje oppgåve 2c) tilsvarande, noko som kanskje påverka resultatet av elevarbeid i 2e). Dette hadde truleg vorte oppdaga dersom eg hadde utført eit pilotprosjekt i annan klasse, noko eg av ulike årsaker ikkje fekk gjort.

[2] Det vert ikkje gjort analyse av denne oppgåva, då elevane hadde svara på spørsmål, utan å ta noko med seg vidare i arbeidet sitt.

3.7 Etikk

Som forskar på eigen klasse er det viktig at relasjonane eg har til dei som deltek i studien vert oppretthaldne og vidareutvikla. Å ta hensyn til etiske retningslinjer er dermed svært viktig. Desse retningslinjene er utarbeidd av Den nasjonale forskningsetiske komité for samfunnsvitenskap og humaniora (NESH), der mellom anna personvern, informert samtykke og anonymitet vert omtala (NESH, 2021).

Mitt prosjekt var meldepliktig sidan eg skulle ta lydopptak. Det vart dermed meldt inn til NSD og godkjent på bakgrunn av personvernlovgivinga (vedlegg 1.)

Informert samtykke (vedlegg 4.) vart sendt til heimane for underskrift frå føresette og elevane sjølve i forkant av prosjektet. Eit slikt samtykke skal vera *frivillig*, *informert* og *utvetydig* (NESH, 2021). Dette vil seie at det skal komme tydeleg fram kva det inneber å delta, og at det er frivillig og dermed ikkje får konsekvensar for den einskilde om ein ikkje er med eller trekkjer seg undervegs. Dessutan skal kvar og ein aktivt gi uttrykk for at dei ynskjer å bli med i prosjektet.

Anonymitet handlar om at dei som deltek ikkje skal kunne kjennast att for dei som les oppgåva. Dette vart teke i vare ved at alt elevarbeid er utan namn. Dialogar og utsegn er vist til med bruk av bokstavar i alfabetisk rekkjefølgje, og alle elevar vert omtala som *hen*.

3.8 Kvalitet på studien

Å utføre ein studie vil ikkje seie det same som å finne ei eksakt sanning. Det eg gjer er å prøve å beskrive korleis desse elevane tenkjer på dette tidspunktet i opplæringsløpet. Kor god kvaliteten på arbeidet er avheng dermed av refleksjonen eg har over kor gyldige og pålitelege funna og resultata er (Postholm & Jacobsen, 2019).

Validitet, eller kor gyldige funn og resultat er, kan delast inn i *indre* og *ytre validitet* (Postholm & Jacobsen, 2019). Det vil seie om resultata er gyldige innanfor og utanfor konteksten dei vart utført i. Indre validitet handlar mellom anna om det er konsistens mellom funna eg gjer og teoretiske rammeverk eg brukar, eller konsistens mellom intervju og observasjon (Larsen, 2017). Har eg dekning for at analysen eg gjer av det eg samlar inn er rett i forhold til elevane eg observerer? Validiteten i studien min vert auka ved hjelp av triangulering, gjennom å ta inn både skriftleg arbeid/arbeid på It´s learning, observere og intervjue. Ved å knytte funn eg gjer opp mot omgrep innanfor algoritmisk tankegang (Bocconi et al, 2018; Grover & Pea, 2017; Shute et al, 2017; Brennan & Resnick, 2012; Csizmadia et al., 2015) kan eg også auke grad av validitet. På den andre sida er det

ikkje ein eintydig definisjon av kva CT eller algoritmisk tankegang er, noko eg synest har vore utfordrande i dette arbeidet.

Den ytre validiteten handlar om kor vidt funn og resultat kan generaliserast til andre kontekstar eller grupper ein ikkje har utforska (Postholm & Jacobsen, 2019). Sidan dette er ei lita gruppe elevar kan ikkje resultatet utan vidare generaliserast til å gjelde alle elevar på 8.trinn. Resultatet av prosjektet kan likevel vere av interesse for andre då funn eg gjer er samsvarande med tidlegare studiar (Fagerlund et al, 2020; Grover & Pea, 2013; Shute et al, 2017; Lee et al., 2011, Lytle et al., 2019).

Reliabilitet, eller kor mykje ein kan lite på resultatet, er avhengig av om arbeidet med datainnsamling, registrering, reinskriving, analyse og framstilling av funn er nøyaktig (Postholm & Jacobsen, 2019). I kvalitative undersøkingar kan dette vere vanskeleg, då ein under observasjon gjer mange ulike tolkingar, og at ulike forskarar kan ha ulik tolking av same situasjon (Larsen, 2017; Postholm 2004). På same måte kan det eg observerer i klasserommet oppfattast annleis når eg seinare skal analysere observasjonen. Observasjon og intervju kan også påverke den som blir intervjua eller observert, og dermed vere av betyding for kva som kjem fram. Sidan eg gjer undersøking i eigen klasse er dette særskilt viktig å tenkje over. Eg har kunnskap om elevane på førehand, og må dermed passe på at denne kunnskapen ikkje påverkar tolkinga eg gjer av resultata. Ekstra viktig er dette i det skriftlege arbeidet. For å sikre at mi forståing av arbeid levert vart så objektiv som mogleg, tok eg arbeidet med til tilbake til dei aktuelle elevane og spurte korleis dei hadde tenkt. Likevel kan eg gjere feil vurderingar, og som ved observasjon og intervju må eg gjere det beste for å unngå mistydingar mellom kva informantane meiner og eg som forskar oppfattar. I forkant av undervisninga fekk elevane beskjed om at eg ikkje var ute etter rette svar. Derimot sa eg at det å ikkje få det til var heilt normalt når ein jobba med programmering, og dessutan av stor interesse for kva eg såg etter. I tillegg fekk dei beskjed om at ingenting av det dei presterte ville ha (negativ) betyding for karakteren i faget. Dette for å prøve å trygge dei, slik at dei kunne svare ærleg utan å prøve å gjere meg som lærar «tilfreds». Eg var litt usikker på om bandopptak ville påverke dialogen omkring oppgåvene, men det verka som om bandopptakaren var gløymt etter kort tid.

4 Resultat og analyse

I dette kapittelet vil eg leggje fram funn frå studien. Eg tar ikkje med alle elevsvar, men funn som kan relaterast til CT. Sidan eg ser etter algoritmisk tankegang hos elevar når dei jobbar med programmering etter undervisningsmodellen PRIMM, er det naturleg å sjå etter dei ulike CT-ferdigheiter (Bocconi et al., 2018) innafor kvar av dei ulike fasane av modellen; *predict, run, investigate, modify* og *make*. I tillegg til **abstraksjon**, **behandling av algoritmar**, **automatisering**, **dekomposisjon**, **testing og feilsøking** og **generalisering** ser eg etter **samarbeid** (Grover & Pea, 2017; Brennan & Resnick, 2012; Shute et al., 2017) og **uthalding** (Weintrop et al., 2015).

Som ei hjelp for meg sjølv medan eg strukturerte funna, laga eg ein tabell der eg noterte inn stikkord etter kvart som eg fann spor av dei ulike kjerneferdigheitene. Då dette er elevar i startfasen av programmering, definerer eg dei ulike omgrepa vidt. Funna vart først strukturert etter innleverte oppgåver på it´s learning og i heftet dei fekk utlevert, så gjennom observasjon og diskusjon i klasserommet og til slutt gjennom observasjon og ustrukturert intervju med bandopptak. Skjema legg eg ved som ei oppsummering av analysedelen.

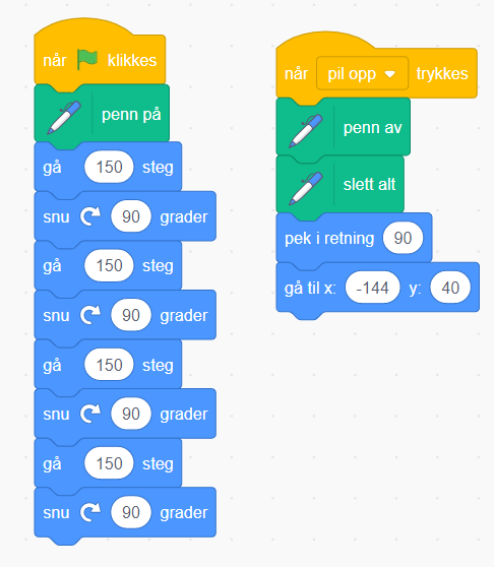
Når eg refererer til elevar brukar eg bokstavar. Eg startar frå A i alfabetet, og same bokstav står for same elev gjennom heile analysen. For å vise at det har vore dialog som ikkje er tatt med, brukar eg tre stjerner ***.

Eg hadde to oppgåver til elevane, der kvar av dei var delt inn frå a) til e). Oppgåvene vert i analysedelen presentert i oppstart av kvar fase.

4.1 Predict og Run – Forutsei kva som kjem til å skje i koden, og sjekk om hypotesen er rett

Oppgåve 1

a)Kva skjer i koden?



Her ser du to kodar som programmerar katten til å utføra eit oppdrag. Den første koden består av 10 blokker, og den andre av 5 blokker.

Skriv ned steg for steg kva dei ulike blokkene gjer. Hugs å få med kva retning ansiktet til katten peikar på kvar linje. Utgangspunktet til katten er slik den står over.

Kode 1

Blokk ein: _____

Blokk to:_____

Blokk tre:_____

Blokk fire:_____

Blokk fem:_____

Blokk seks:_____

Blokk sju: _____

Blokk åtte:_____

Blokk ni:_____

Blokk ti:_____

Oppdraget til katten er:_____

Kode 2:

Blokk 1:_____

Blokk 2:_____

Blokk 3:_____

Blokk 4:_____

Blokk 5:_____

Oppdraget til katten
er:_____

_____

_____

_____

I oppgåve 1a) hadde alle par løysing som inkluderte *firkant* i forslaget sitt, men berre to av para hadde konkludert med at dette måtte vera eit kvadrat. Det kan vere elevane såg det var eit kvadrat, men er upresise når dei noterar. På den andre sida kan det og vere at dei berre observerer 90^o og at dei skriv firkant ut frå det.

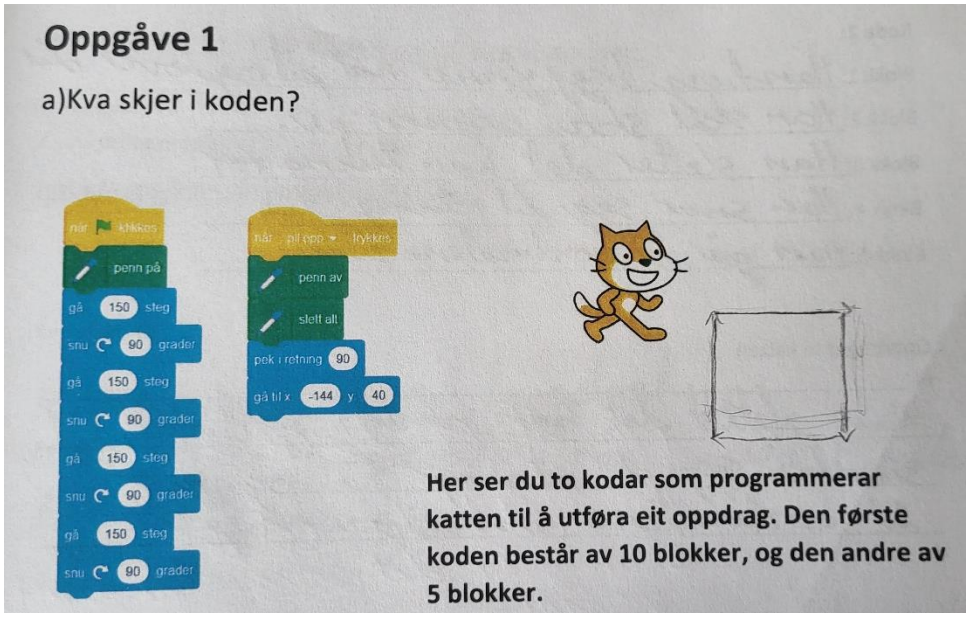
Under observasjon i klassen stilte eg dette spørsmålet til elevane A og B:

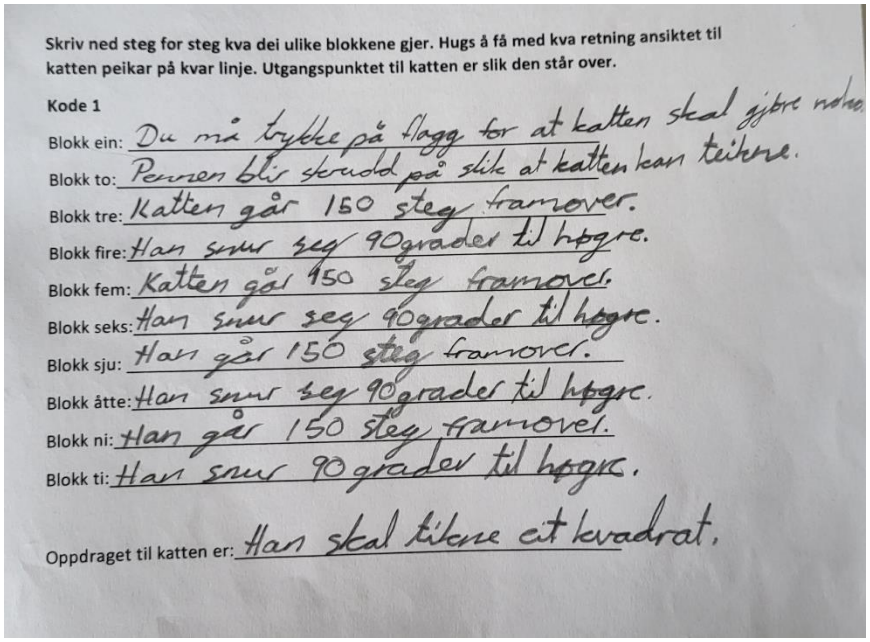
> Lærar: Kva for type firkant er det?
>
> A: Er det viktig då? Det er jo ein firkant.
>
> ***
>
> Lærar: Korleis veit de det er ein firkant då?
>
> A: Fordi det står 90 grader
>
> Lærar: Er det noko anna i koden som gjer at me veit kva type firkant det er?
>
> B: Katten går like mange steg heile tida, så då er det sikkert eit kvadrat.

Elevane kan namn og kjenneteikn på ulike geometriske figurar, men er likevel ikkje presise når dei skal behandle algoritmane, noko Grover & Pea (2017) presiserer er viktig. Eit program krev ein presis algoritme for å fungere som tenkt, og elevane treng øving i dette, noko eg meiner dei kan få gjennom å tolke ferdige kodar i predict-fasen (**behandling av algoritmar**). I følgje Wing (2014)

inneber abstraksjon mellom anna å kjenne att mønster, og her kan det sjå ut som om elevane har **abstrahert** 90 grader og sett bort frå resten av informasjonen. Likevel klarar dei å finne andre kjenneteikn på kvadrat i koden då dei får spørsmål om dette. Same spørsmål som over vart og stilt til andre par som ikkje hadde skrive at katten teikna eit kvadrat, og resultatet av dialogane samsvara med elev A og B.

Dei to neste bileta er løysinga eitt par leverte i heftet.





Paret har konkludert med at katten teiknar eit kvadrat. Sjølv om dei har formulert kva som skjer på dei ulike linjene i koden, har dei teikna eit kvadrat med piler for gangretning for å illustrere kva som skjer. Dei forklara etterpå at dei trudde katten ville teikne eit kvadrat, men for å vere sikker, så teikna dei linje for linje i koden. I oppgåva er koden allereie knekka ned linje for linje (**dekomposisjon**), men elevane forsøkte å **teste** hypotesen sin ved å teikne dei ulike stega. Teikninga fungerte dermed som **mediering** (Kozulin, 2003). Dette paret viser også tydeleg at dei forstår kva som skjer i dei ulike blokkene ved at dei forklarar med eigne ord, og ikkje berre skriv det som står på

kvar blokk. Medan sju andre elevpar i sine oppgåveløysingar berre hadde skrive «penn på» til forklaring av blokk to, forklarar desse at «pennen blir skrudd på slik at katten kan teikne». Dei er presise når dei behandlar algoritmane (Grover & Pea, 2017).

Eitt par tok kontakt med meg fordi dei ikkje heilt forstod blokka «penn på».

> C: Kva betyr penn på?
> Lærar: Kva trur du det betyr?
> C: Det kan bety mykje .
> Lærar: Som kva?
> C: Det kan jo bety at pennen skal på, men det seier jo den første linja.
> Lærar: Eg forstår ikkje heilt kva du meiner. Kan du forklare?

Dette paret startar allereie under Predict-delen å diskutere om blokker er naudsynte eller ei, noko som kan knytast til **abstraksjon**. Løysing av komplekse problem blir lettare om ein reduserer detaljer som er naudsynte, og fokuserer på det som betyr noko (Bocconi et al., 2016). Er det naudsynt å ta med blokka «penn på»?
Eg trur eg forstår kva elev C meiner når hen seier at den første linja (Når flagg klikkes) seier at pennen skal på, men spør for å sikre at eg tenkjer rett. Vidare startar ein diskusjon elevane mellom ut frå spørsmålet eg stilte.

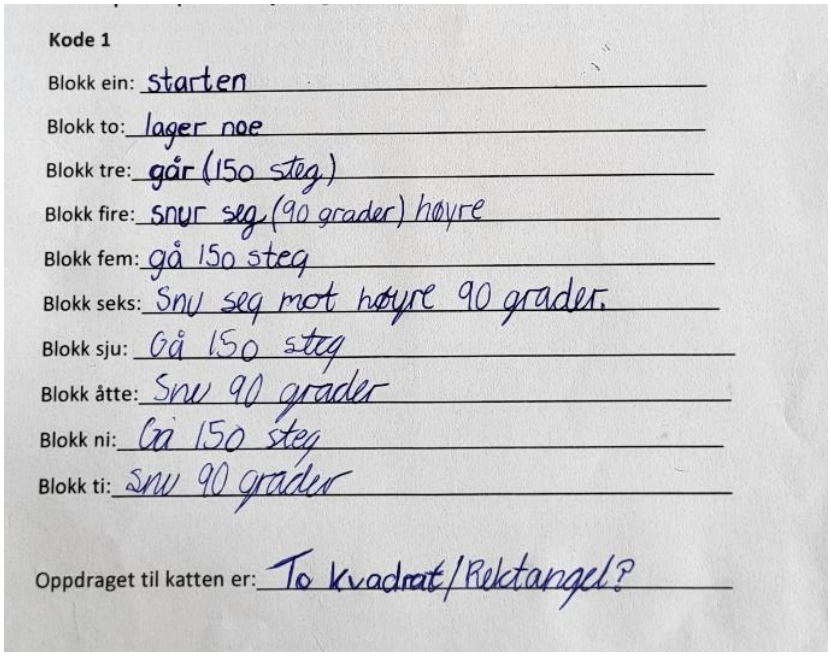
> C: Den med "Når flagg trykkes" seier at du skal byrje, og koden startar jo å gå utan "penn på".
> D: Men det kan jo vere at katten teiknar noko når det står "penn på".
> C: Ja, og så berre gjer den bevegelsane når blokka ikkje er der. Eg trur du har rett.

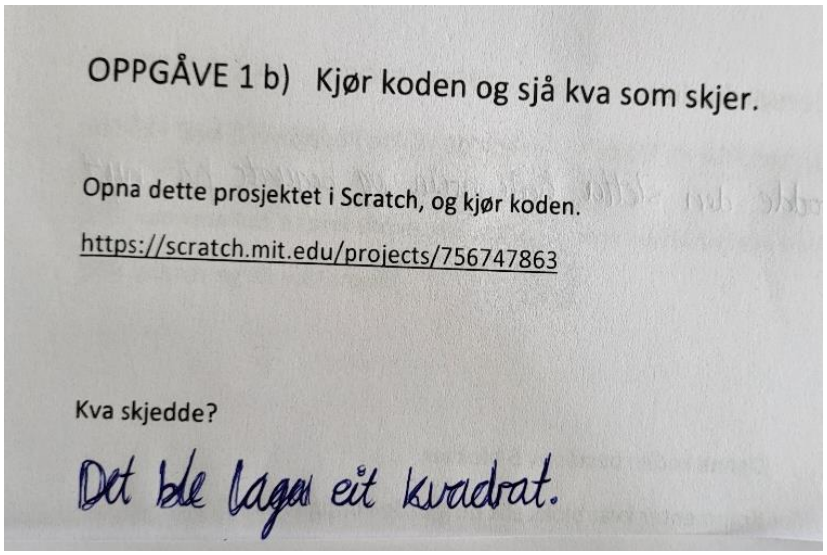
Dei finn svar på utfordringa ved å diskutere seg i mellom (Brennan & Resnick, 2012), i ein aktivitet som ligg innanfor næraste utviklingssona (Skott et al., 2019). **Logic and logical thinking** (Grover & Pea, 2017; Csizmadia et al., 2015) gjer at elevane tenkjer seg fram til kva brikka kan tyde, ved å bruke erfaring frå dagleglivet til å tyde koden.

> Lærar: De kan jo testa det ut etterpå. Kva med bevegelsane då? Korleis bevegar katten seg?
> D: Den lagar ein firkant.
> Lærar: Ein firkant? Korleis ser den firkanten ut då?
> D: Det er ….vent litt.....det blir eit kvadrat, fordi alle sidene er like lange og så er vinklane 90 grader.

På same måte nyttar dei også tidlegare erfaring og læring til å konkludere med at dette er eit kvadrat.

Bileta under viser korleis eitt par hadde laga ein hypotese over at koden førte til at katten teikna to kvadrat eller rektangel, og at resultatet vart eit kvadrat då dei kjørte koden. Etter å ha sett kva dei konkluderte med at oppdraget til katten var, spurte eg dei om dette.





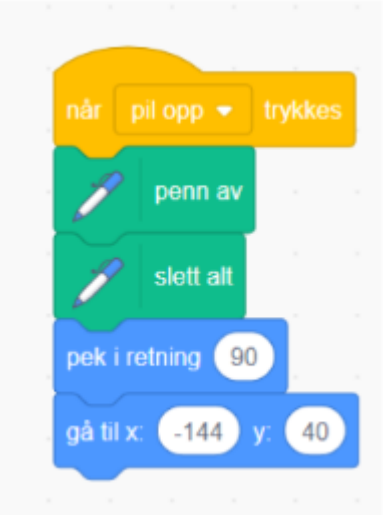
Lærar: Eg ser at forslaget dykkar var at katten skulle teikne to kvadrat eller rektangel, men at dette ikkje stemte med resultatet. Veit de kvifor?

E: Ja, for me tenkte liksom sånn at først så gjekk katten, og så snudde den seg og gjekk vidare. Så gjorde den det same ein gong til, og då vart det eit kvadrat……me hugsa først ikkje heilt om det heitte kvadrat eller rektangel, men det er eit kvadrat.

F: Og så gjorde den det same ein gong til, og då hadde me to kvadrat. Altså to gongar på det eine kvadratet og to gongar på det andre kvadratet, og derfor vart fire gongar to kvadrat, om du forstår kva eg meinar…

Samtalen over er etter at elevane har tolka og kjørt koden, og sett at dei hadde tolka feil. Dei kunne utan opphald svare meg då eg spurte kvifor dei hadde skrive at det vart to kvadrat eller rektangel. Det første av di dei ikkje hugsa om det heitte kvadrat eller rektangel, og det andre av di dei tenkte at *snu 90 grader* meinte at katten snudde og gjekk vidare ved hjelp av same blokk. Då *gå 150 steg*, og *snu 90 grader* vart gjenteke 4 gongar, tolka dei det som to kvadrat/rektangel. Sidan elevane kunne svara meg på kva dei hadde gjort feil med ein gong spørsmålet mitt vart stilt, tolkar eg det som om dei har tenkt gjennom kva som var feil med hypotesen dei hadde før eg spurte. Dei hadde altså **evaluert og funne feilen**, noko som er eitt av CT-konsepta hos Bocconi et al.(2018) og kjem inn under CT-practices hos Grover & Pea (2017) og Brennan & Resnick (2012).

Også slettekoden skulle forklarast i oppgåve 1a). Eg observerer medan elevane diskuterer. Dette er same elevane som over diskuterte om *penn på* hadde ein funksjon i koden.



C: Må den klossen vera med? (Peikar på *pek i retning 90*) Og kvifor skal den vera der (*gå til x -144, y 40*)?

D: Kva er det som skal peike i retning 90? Og kva tyder det eigentleg? Eg trur kanskje den *gå til* gjer at katten flyttar på seg. X og Y det jo slikt rutenett, men her er det ikkje ruter….men katten skal sikkert flytte på seg når den har sletta kvadratet.

C: Men eg forstår ikkje den (peikar nok ein gong på *pek i retning 90*). Det er sikkert noko med katten, for alt blir jo sletta, og då står berre katten att. Men eg forstår ikkje kvifor det står 90.

Etter litt att og fram utan å konkludere, opnar dei programmet (oppgåve 1b).

Dei trykkjer *grønt flagg* (teiknar kvadratet) og *pil opp* (slettar).

C: ……men den teiknar jo kvadratet og slettar det att. Forstår ikkje dei klossane no heller.

Eg viser *sjå inni*, og ved å endre på verdiar finn dei ut av kva begge blokkene dei var usikre på utfører. For at elevane etter kvart skal kunna **behandle algoritmar**, er det viktig at dei kan lese og forstå kva dei ulike blokkene utfører. Dette elevparet kunne ha sagt seg nøgd med at katten teikna eit kvadrat og sletta det att, men dei ønska likevel å finne ut av kva for funksjon kvar av brikkene hadde. Dei viser **uthalding** (Weintrop et al., 2015). Elevane stilte ikkje spørsmål til kvifor desse blokkene var med i slettekoden slik dei gjorde med *penn på* tidlegare i oppgåva.

Oppgåve 2a

Denne koden består av 11 blokker. Kva er oppdraget til katten?

Blokk 1:

Blokk 2:

Blokk 3:

Blokk 4:

Blokk 5:

Blokk 6:

Blokk 7:

Blokk 8:

Blokk 9:

Blokk 10:

Blokk 11:

Eg trur katten får i oppdrag å teikna_____

I denne koden.

Oppgåve 2a) var det tre par som hadde jobba med. Elevane har jobba med løkker i oppgåve 1, men her er løkke inni løkka introdusert. I tillegg er det rotasjon og endring av pennefarge.

1. «Me trur at katten får i oppdrag å teikne 24 kvadrat i ulike fargar».
2. «Me trur at katten teiknar mange kvadrat etter kvarandre. 24 faktisk. Og dei har forskjellige fargar».
3. «Me trur at dette blir ein regnboge av 24 firkantar».

Dette er tre forklaringar henta frå heftet og it´s learning. Alle tre par såg at dette vart 24 kvadrat, og at dei endra farge. Alle hadde skrive «snu 15 grader» som forklaring på blokk 8, men berre eine paret hadde forsøkt seg på kva det gjer med mønsteret. Eg var litt usikker på kva dei meinte med ein regnboge av firkantar, om det var fargen eller forma dei refererte til. Eg spurte difor i etterkant for å sikre at eg tolka rett. Det vart bekrefta at dei tenkte både form og farge, «me såg berre ikkje at det vart ein heil sirkel».

Elevane **generaliserer** det dei har jobba med i oppgåve 1, der dei vart introdusert for løkker. Slik kunne dei bruke dette for å tyde koden. Dette går under **patterns and pattern recognition** hos Grover & Pea (2017).

Etter å ha kjørt koden, skulle elevane i oppgåve 2c) skrive kva det var som gjorde at dei hadde feil i hypotesen sin (om dei hadde feil). Dette går under **testing og feilsøking** eller **evaluering**. To av para hadde skrive at det var blokka *snu 15 grader* som gjorde at dei ikkje oppdaga at kvadrata var forma som ein sirkel, eller «ein donut» som eine paret skreiv. Det tredje paret hadde oppdaga at 15 grader roterte kvadrata, men oppdaga ikkje, som skrive over, at 15 grader 24 gongar gjer at det vert ein sirkel. Som i oppgåve 1a) hadde kanskje det å teikne det koden ville utføre vore til hjelp i tolkinga. Kanskje dei ikkje hadde oppdaga sirkelen, men i alle fall *snu 15 grader* hadde fått meir fokus i tolkinga.

4.2 Investigate – undersøk koden

Oppgåve 1 c) Undersøk koden

Dersom det skjedde noko anna når du kjørte koden enn du tenkte i oppgåve 1, kvar i koden tenkte du feil?

Denne koden består av 5 blokker.

Kommenter kvar blokk slik du gjorde i oppgåve 1:

Samanlikna desse kodane.

Kva er likt?

Kva er ulikt?

Vil kodane gi ulikt resultat når dei blir køyrde? Gi grunn for svaret ditt.

Å undersøkje koden på oppgåve 1c) meistra dei fleste godt. Konklusjonen som gjekk att under observasjon og i arbeid som vart levert inn, var at begge kodane førte til at eit kvadrat vart teikna, men at den «eine koden var lengre enn den andre». Ei gruppe hadde forklara med at «kodane utfører det same, men *gjenta-blokka* gjer at denne koden blir kortare og meir oversiktleg», **ein brukar løkker for å gjere instruksjonane kortfatta** (Brennan & Resnick, 2012).

Eg stoppa opp ved eitt par, då eg observerer at dei har skrive «at koden lagar 4 firkantar».

G: Denne koden lagar fire firkantar.

Lærar: Korleis veit du det?

G: Jo for det er samme lengde katten skal gå, og så er det 90 grader, og så står det gjenta fire gongar. Altså fire firkantar.

Eg spør om eleven kan forklare firkanten.

G: Den går 150 steg, og så snur den 90 grader, og så går den vidare slik....(teiknar eit kvadrat med fingeren).

Lærar: OK, så den skal gå 150 steg, og så snu 90 grader, og så gjenta dette fire gongar?

G: Ja, det var jo det eg sa. Og så blir det fire firkantar.

Lærar: OK. Skriv det, og så testar de koden før eg kjem tilbake.

Eg håpar at dei finn ut av feilen sjølv, og gjer dei tid til å opne den ferdige koden før eg går tilbake.

Lærar: Korleis gjekk det med koden?

H: Det vart feil. Det vart to firkantar ved sida av kvarandre. To kvadrat.

Lærar: To kvadrat ved sida av kvarandre. Kan eg få sjå?

G: Me har berre sjekka, og så sletta me.

Lærar: Det hadde vore kjekt å sjå løysinga. Kan de prøve ein gong til slik at eg får sjå?

Denne gongen blir eg verande for å sjå kva dei gjer.

G: Ja. Slik.....men no vart det ikkje likt.....hmmm

Lærar: Korleis blei det no då?

G: Den lagar mange firkantar over kvarandre. Kvadrat altså. Det er sikkert fire. Eg hadde sikkert rett første gongen.

Eg ser at koden er rett, men at dei trykkjer flagg fleire gongar.

H: Der ser du, den laga enda ein firkant over.

Lærar: Ein eller fleire?

H: Det er fleire, men den lagar berre ein om gongen.

Lærar: Så kva utfører koden?

G: Fire firkantar....ja, eg veit, kvadrat.

Elevane får i oppgåve å lage ein slettekode, og etter å ha testa slettekoden og så starta koden på nytt:

G: Den lagar berre eitt kvadrat.

***

H: Åhh det står jo gjenta fire gongar. Det er jo ikkje fire kvadrat. Den skal jo gå og skifte retning fire gangar for at det skal bli ein firkant…..eit kvadrat.

Elevane **evaluerer** resultatet sitt (Grover & Pea, 2017, Brennan & Resnick, 2012) og fann at hypotesen deira om at det vart teikna «fire firkantar» ikkje stemte. Det vart «to kvadrat ved sida av kvarandre». Denne forklaringa godtok elevane, og dei sletta arbeidet sitt. Om dei hadde prøvd på nytt (**debugging)** utan lærar til stades, er usikkert. I dette tilfellet fungerte ikkje elevane som støtte for kvarandre, men dei fann saman ut av problemet med støtte frå lærar (Wood et al., 1976).
Eg tolkar det også som om desse elevane ikkje har heilt forståing for **automatisering** . I første omgang seier elevane at programmet gjer at katten teiknar «fire firkantar», i neste runde er det «der ser du, den laga enda ein firkant».
Automatisering skal spare oss for arbeid ved at datamaskinen skal utføre eit sett med oppgåver samanlikna med om menneske skulle ha gjort same operasjon (Lee et al., 2011). Dersom ein ynskjer

eit arbeid utført, og må setje programmet i gang for kvar operasjon, er ikkje dette effektivt, og dermed kan ein like gjerne gjere det utan bruk av programmering. Dessutan skal same kode utføre same operasjon kvar gong han blir kjørt, og ikkje «fire firkantar» eine gongen, og «ein firkant til» andre gongen.
Elevane verkar heller ikkje å ha taket på kva ei løkke er. Dei ser ut til å forstå at det er noko som skal gjentakast fire gongar, då elev G seier at «koden lagar 4 firkantar», men ser ikkje ut til å forstå kva som skal gjentakast. Dei klarar ikkje å tolke koden **(behandling av algoritmar**).

Oppgåve 2c) vel eg å ikkje kommentere, då elevane som jobba med den har funne ut kva som skjer, utan å ta det med seg vidare i arbeidet sitt. Oppgåva ligg som vedlegg (vedlegg 3).

4.3 Modify – endre koden

Oppgåve 1 d) Endre koden.

Du har no jobba med ein kode som teiknar eit kvadrat. Endre denne slik at den teiknar eit rektangel.

Det kan vere lurt å først skrive ned setningar som forklarar linje for linje kva du skal gjere.

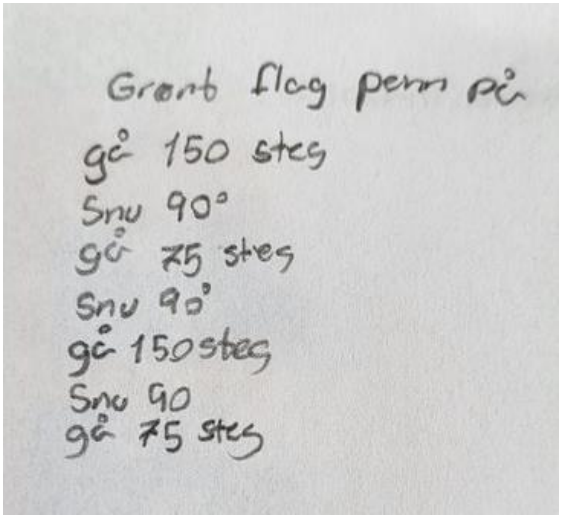
Slik teiknar eg eit rektangel:

Lag koden i Scratch.

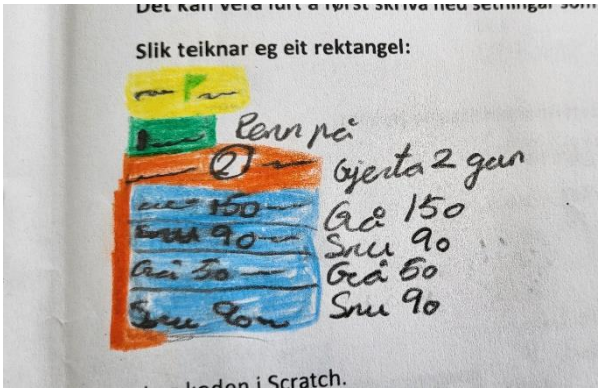
Bruk utklippsverktøy, og lim inn bilete av koden du enda opp med under.

Du kan med fordel også ta med kodar du lagar som ikkje gjer ønska resultat.

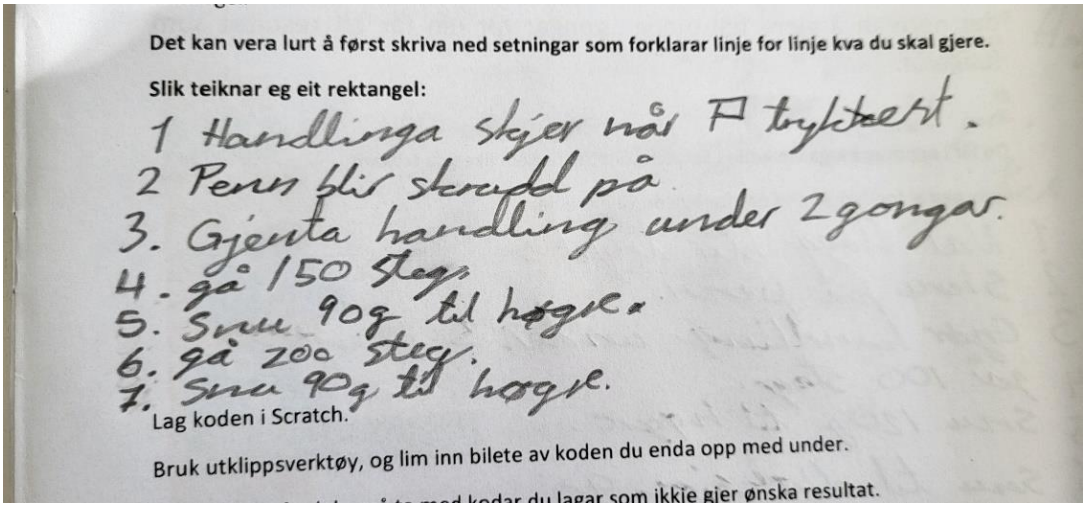
Resultata eg presenterer frå oppgåve 1d) er alle henta frå innlevert skriftleg arbeid (hefte og its`learning). Først tar eg med tre forklaringar henta frå heftet.



Bilete 1



Bilete 2



Bilete 3

Bileta over viser tre forklaringar på korleis elevpar vil gå fram for å lage eit program som teiknar eit rektangel. Med forklaring var tankane mine at dei skulle skrive tekst slik dei ville forklare det til ein medelev utan tanke på programmeringsspråk. Elevane har tolka det som om dei skulle skrive i programmeringsspråket.

Dekomposisjon og **behandling av algoritmar** er sentralt her. Elevane må først bryte ned problemet ved å ta tak i eigenskapane til eit rektangel, for så å lage ein algoritme steg for steg. Alle tre løysingsforslaga over har truleg brukt koden i oppgåve 1c) som utgangspunkt. Dette omtalar Brennan & Resnick (2012) som **reusing and remixing** medan Csizmadia et al. (2015) har det under

generalisering. To av para (bilete 2 og 3) har også teke i bruk løkker framfor å ta med alle stega (black-boxing), noko som går under **abstraksjon.** Elevane med løysingsforslaga som vert vist i bilete 1 og 2, har ikkje presisert retning for kva veg dei skal snu, men dette har det heller ikkje vore trekt fram i oppgåvene dei har løyst til no.

Dei to påfølgande forklaringane er henta frå It´s learning.

«Skift to av gå 150 til 300. Då får du eit rektangel». Eg tolkar det slik at også her er koden i oppgåve 1c) brukt som utgangspunkt, då denne hadde ei løkke der ein skulle gjenta fire gangar «gå 150 steg, snu 90 grader til høgre». Forklaringa desse elevane har er truleg presis nok for elevane sjølve, men ikkje ein algoritme for korleis teikne eit rektangel. Dei viser likevel forståing for at same kode kan nyttast med få justeringar (Brennan & Resnick, 2012).

«Gå 75 steg, snu deg 90 grader, gå 150 steg, snu deg 90 grader, gå 75 steg, snu deg 90 grader, gå 150 steg». Dette paret forklarer stegvis kva som skal skje, **behandling av algoritmar**, men heller ikkje dette paret har skrive noko om retning. Dei har heller ikkje brukt løkke, sjølv om dette vart gjort i oppgåve 1c). Dette kan tolkast som at dei, som elevane frå bilete 1, ikkje heilt har forstått korleis løkker kan nyttast. Når ein teiknar eit kvadrat, er det same handling som skal gjentakast fire gongar. Sidan eit rektangel har to og to like lange sider, må dei tenkje litt annleis. Dei har dermed ikkje klara å **generalisere** handlinga frå kvadrat til rektangel.

Etter gjennomgang av alle hefter og innleveringar på it´s learning fann eg at fire par hadde levert kode, men utelatt skriftleg forklaring. Som eg kjem tilbake til seinare, kan dette vere av di det er vanskeleg å skrive forklaring, men det kan og vere at Scratch gjer det enkelt å prøve og feile, og at dei dermed ikkje finn skriftleg forklaring naudsynt. To av para hadde ikkje nytta seg av løkke for å effektivisere koden (Brennan & Resnick, 2012), noko som kan tyde på mangelfull **abstraksjon og generalisering**. Alle hadde fått til å lage ein kode som fekk katten til å teikne eit rektangel.

Oppgåve 2d) Endre koden

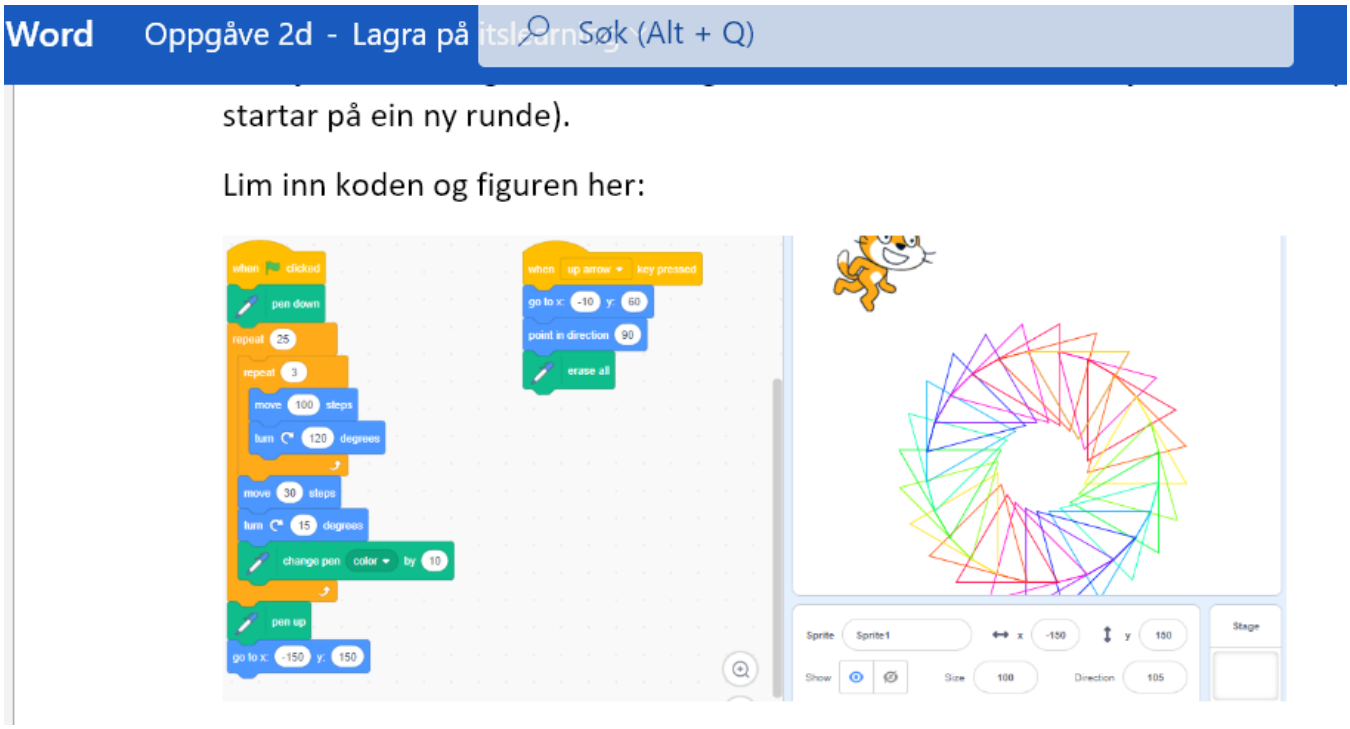
Bruk koden i frå oppgåve 2b) som utgangspunkt. Dette er rotasjon av eit kvadrat.

https://scratch.mit.edu/projects/743858627

Endre koden slik at det blir rotasjon av ein trekant (eller valfri mangekant, bortsett frå kvadrat). Sidelengde vel du sjølv. Endre også antal rotasjonar frå 24 til eit valfritt antal rotasjonar. NB! Figuren skal henge saman. Det skal heller ikkje vere overlappingar (at den startar på ein ny runde).

Lim inn koden og figuren her:

Biletet under viser innlevert arbeid frå eitt par.



Dette paret tar i bruk erfaringar frå tidlegare oppgåver, og veit kvar i koden dei skal gjere endringar for å teikne ein likesida trekant i staden for eit kvadrat. Dette viser dei ved å lage ei løkke der ein skal repetere tre gongar «gå 100 steg, snu 120 grader til høgre». Dei har også gjort endringar på steglengde. Antal rotasjonar er endra frå 24 gongar i oppgåve 2a) og 2b) til 25 gongar hos dette paret, men elevane har ikkje sett forholdet mellom antal rotasjonar og antal grader katten skal snu, og brukar 15 grader slik som utgangspunktet var.

Etter å ha sett gjennom innleveringa, spør eg om dei ser at katten held fram etter at den er ferdig med ein runde.

***

C: Gjer den det?

Lærar: Prøv kva som skjer om de tar bort siste blokka, den *gå til*…og følg nøye med på kvar katten startar og endar.

C: Det ser ut som den gjekk litt lenger, men eg er ikkje sikker.

L:ærar: Sjå om katten bevegar seg når de trykkjer *pil opp* (slettekode).

D: Ja, den gjekk litt tilbake. Den har starta på ein ny runde.

Lærar: Klarar de å finne ut kva i koden som gjer at dette skjer?

Dei studerer koden (**abstraksjon og behandling av algoritmar**), og konkluderer med at det må ha noko med at dei må endre *turn 15* når dei endrar antal rotasjonar, men dei klarar ikkje å finne samanhengen. Her er det truleg manglande matematisk kunnskap som hindrar dei å sjå dette. Ei betre oppgåve 2c) frå mi side, og gjerne fleire rundar med fokus på forholdet mellom rotasjonar og grader, kunne kanskje vore til hjelp her.

4.4 Make – lage eigen kode

Oppgåve 1e)

Lage eigen kode. Hugs at innan programmering og problemløysing er det normalt å gjere feil mange gongar før ein får eit resultat som fungerer.

Du får i oppdrag å lage ein kode i Scratch som teiknar ein likesida trekant.

Skriv ned ei forklaring med ord for kva du vil gjere her:

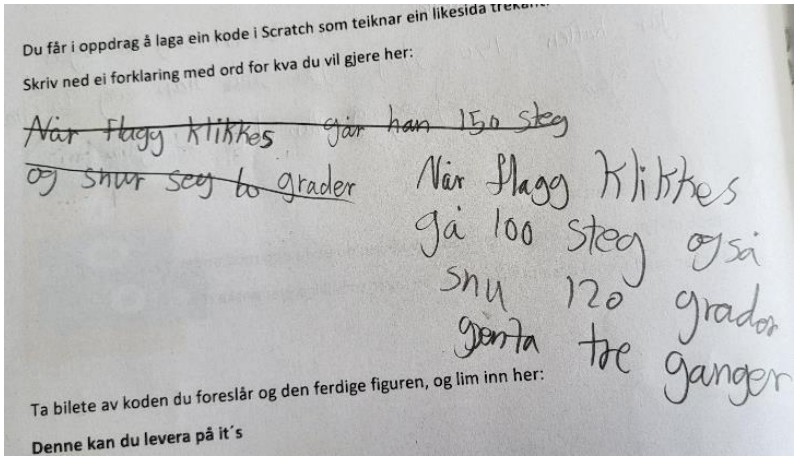
Ta bilete av koden du foreslår og den ferdige figuren, og lim inn her:

Ekstra utfordring:

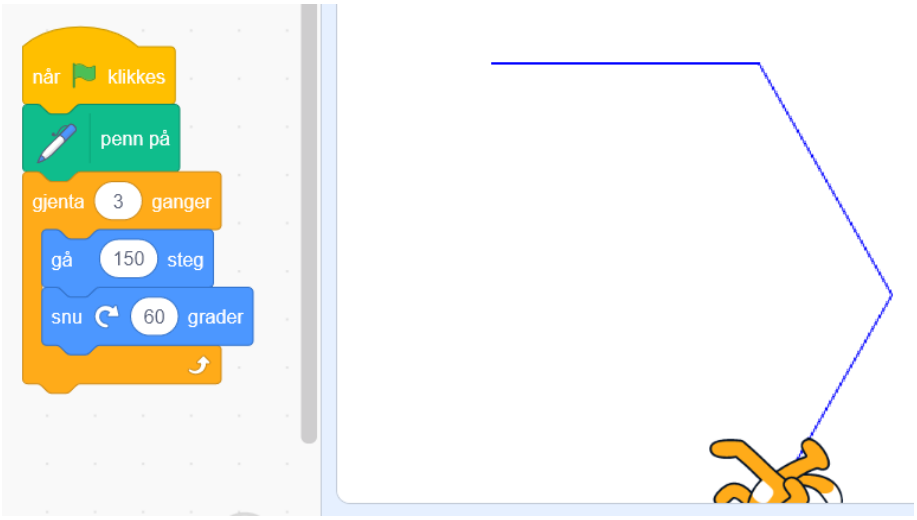
Lag ein kode som teiknar ein regulær (likesida) femkant, sekskant…….

Lim inn resultat under. (Evtentuelt kan de llevere på It´s learning).

Bilete 1 og 2 under representerer korleis dei fleste elevane gjekk fram for å løyse oppgåva. Dei endrar skriftleg forklaring etter at dei har funne ut av koden, skriv forklaringa etter å først ha laga koden, eller utelet forklaringa heilt. Eg vil først ta for meg generelt korleis dei fleste løyste oppgåva, for så å gå djupare inn i eitt par der det vart brukt bandopptak.



Bilete 1



Bilete 2

(henta frå It´s learning, etter observasjonen eg hadde i klasserommet).

Eg peikar på tekst med overstreking (bilete 1), og spør kvifor dei har streka over teksten.

I: Me gjorde det fordi resultatet vart slik (peikar på skjermen, bilete 2)

Lærar: Kvifor vart det slik då?

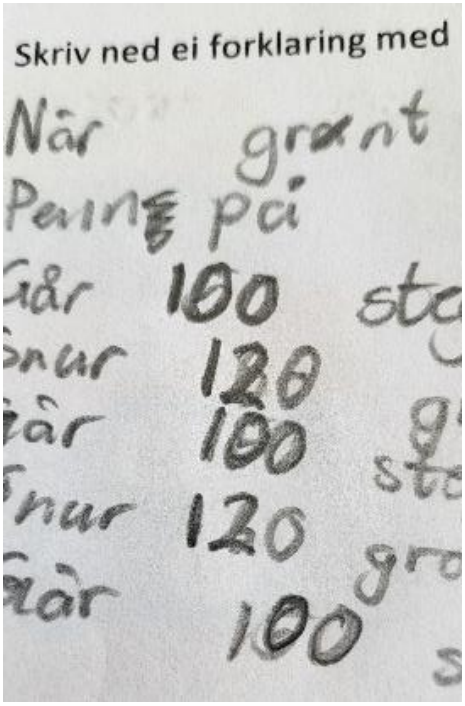
I: Eg veit ikkje, men me prøvde mange gongar…….først laga me litt mindre vinkel (viser i koden sin med snu 45 grader) då 60 grader (bilete 2) gjorde at vinkelen vart så stor.

J: Og då vart vinklane enda større, så me forstod at me måtte ha vinklar større en 60 grader. Me prøvde fleire gongar, og med 120 grader trefte me.

Også steglengda har dei endra (bilete 1), så eg spør om dette har noko å seie for om resultatet vert ein likesida trekant eller ikkje.

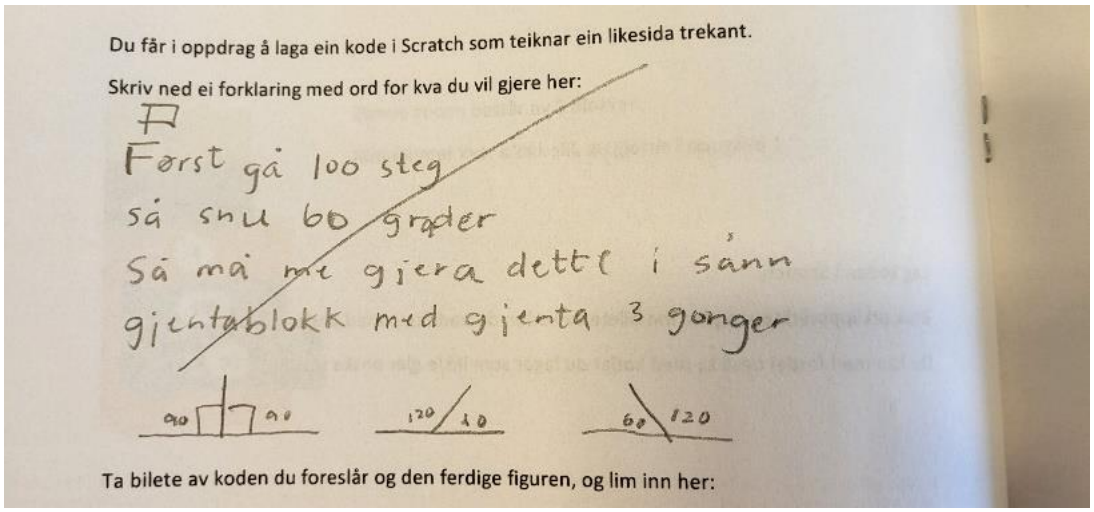
> I: Nei. Det var berre fordi katten forsvann ut av biletet medan me testa, og då prøvde me med færre steg.

Elevane har her laga ein kode. Dei vel å bruke løkke, **generalisering/abstraksjon**. Dei har **laga ein algoritme** for korleis dei vil gå fram, men truleg er dette gjort etter at dei var nøgd med resultatet. Eg tolkar det slik av di dei først skriv «Når flagg trykkes går han 150 steg og snur seg 60 gradar». Løkka manglar, så testinga har truleg starta her og så har dei notert endeleg forklaring etter at koden var ferdig. Elevane **prøver, feilar og evaluerer.**



Også i biletet over kjem det fram eit anna par har endra på algoritmen (truleg) etter **prøving og feiling**. *Snur 60 grader* har blitt skrive over til 120, og steglengda er justert.

Løysinga i biletet under såg eg i eit hefte som vart levert inn. Sidan elevane hadde teikna og skrive namn tett opp til løysinga si, har eg skrive nøyaktig av det som stod skrive.



Eg spurte i etterkant kva dei hadde gjort, og fekk til svar at dei hadde først tenkt dei kunne nytte same kode som i tidlegare oppgåver, men sidan ein likesida trekant hadde vinklar på 60 grader, måtte dei endre frå 90 grader til 60 grader. Dessutan måtte det vere «gjenta 3 gongar» i staden for fire. Då dei såg at gradene var feil, prøvde dei nokre gongar til dei fann at det var 120 grader. Teikningane deira hadde dei laga då dei såg at 60 grader + 120 grader vart 180 grader, og 90 grader + 90 grader vart 180 grader. Dei tenkte at dei skulle gjere det på same måte då dei skulle lage femkant, men så kom dei ikkje lenger før me avslutta økta. Dette er nok eit døme på **generalisering.** Elevane nyttar også teikning som **mediering**.

Vidare tar eg for meg korleis eitt par løyste oppgåve 1e). Det vart gjort bandopptak av samtalen mellom elevane medan eg observerte og stilte spørsmål innimellom. Dette er transkripsjon av delar av diskusjonen.

> K: Skal me ha ei *gjenta – blokk* eller? Meina den me brukte på den andre oppgåva.
> L: ehhhh Me prøver utan *gjenta-blokk* først. Det er lettare det. Visst me tenkjer litt logisk…. om den går «sånn, sånn og sånn» (peikar på skjermen). Visst den går 50 steg først, og så snur han til hø..
> K: Den kan snu til venstre.

Samtalen tyder på at elevane har gått i gang utan å lage ein stegvis plan for korleis gå fram (**behandling av algoritmar**). Elev K verkar som om hen har fått forståing for løkker, og kan bruke det

som eit språk på tankestadiet. Elev L, som på dette tidspunktet sit med tastaturet, har truleg ikkje same forståinga då hen meiner det er lettare å forsøke utan.

L: Nei, vent den snur sånn til høgre 60 grader. Me prøver på det. Me seier den går 30 steg. Nei 50.
K: No må du bestemme deg her.
L: Me seier 50. Og så snu mot venstre.
K: Ja, 60 grader.
L: Ja, me prøver det. Ojjj, det vart feil. Den vinkelen er jo alt for stor.
K: Nei….me må berre ta litt mindre grader. 50 kanskje. Sjå (peikar) den vert jo enda større.

Elevane prøver fleire gongar, og kjem fram til at det må vera 120 grader.

K: Me må ha fleire steg, slik at me ser noko anna enn katten. (Endrar til 100 steg).
L: Gå 100 steg, snu så til venstre. Eller det treng jo ikkje vera til venstre då.
K: Jo, det må det. Eller forresten. Det treng ikkje vera til venstre, men me gjorde det.

Elevane har kommentert fleire gongar om katten skal snu til venstre eller høgre, men sjekkar ikkje om det har noko å seie for resultatet kva dei vel, så eg spør. «Den lagar vel same trekanten uansett», var responsen.

K: Me kan jo sjekka. Hen startar å endre koden.
Lærar: Prøv å sjå for dykk eller teikne kva som skjer om den snur til høgre eller til venstre.

Eine eleven teiknar med fingeren, og kjem fram til at «spissen står opp dersom me snur til venstre, og ned om me snur til høgre». I denne oppgåva hadde det ikkje noko å seie, men elevane fekk øving i å **tolke algoritmen** med fingeren som **mediering**.

****
K: Du har ikkje sagt at dette skal gjentakast fire gongar. Eller tre gongar er det.

Elev K nemner for andre gang gjentaking (løkke), utan at det vart gjort noko med. Programmet vart køyrt.

K: Klarte me det? Der har me den, me må berre ta bilete. Me er så gode! Me er så gode! Too easy

L: Det vart eit meisterverk. Sjølv om den ikkje er så stor, så er det bra. Det viktigaste er at me har laga ein trekant. Vent litt….ja, den var likesida. Send den inn du, så skriv me den greia. («Den greia» er forklaringa for korleis dei skulle gå fram.)

Desse elevane har, på same måte som dei tidlegare nemnte, løyst oppgåva gjennom **prøving og feiling**. Den stegvise forklaringa (**algoritmen**) er ikkje laga i forkant, men elevane kommenterer at dei skal skrive «den greia» etter at løysinga er sendt inn på it´s learning . Elev K nemner to gongar i samtalen *løkke*, der hen første gong refererer til «den me brukte på andre oppgåva». Vedkommande startar dialogen med denne setninga, og nyttar dermed **abstraksjon** der hen får fokus bort frå repetisjon og over på det som er utfordringa med denne figuren, altså kor mange grader det skal vera. Det er også ei **generalisering** ved at hen ser at metoden ein brukar for å lage kvadrat og rektangel også kan nyttast til å lage likesida trekantar. Sjølv om hen også nemner dette før oppgåveløysinga vert sendt inn, vert dette ikkje endra. Litt usikker på kvifor dette ikkje vert gjort, men det kan ha årsak i at hen gav seg fordi hen antok at den andre eleven hadde meir kontroll. Dette er eit døme på at **schaffolding** (Wood, Bruner, & Ross, 1976) ikkje alltid fungerer som tenkt.

Måten dei løyste problemet på vart ikkje kommentert av meg på dette tidspunktet, av di eg måtte forlata paret nokre minutt. Då eg kom inn att var dei nettopp starta på ekstra utfordring med å lage kodar for regulær sekskant og femkant.

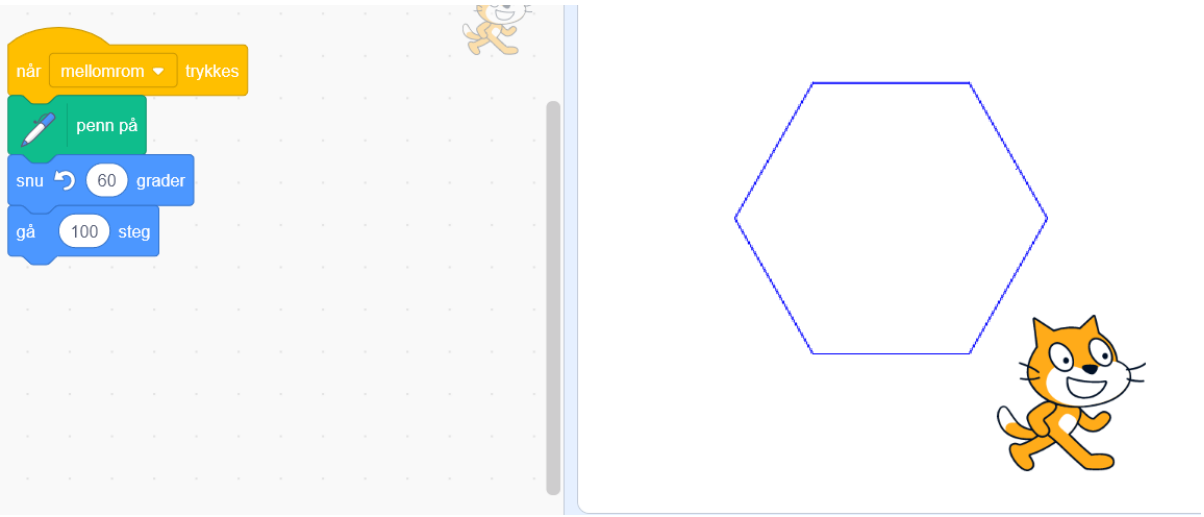
Eg lyttar medan paret diskuterer korleis lage ein kode som får katten til å teikne ein sekskant.

L: No kan me ikkje snu 120 grader

K: Nei, det er litt for mykje

L: Eg prøver med 60 grader, for då me skulle lage trekanten prøvde me jo med 60 grader, men me fekk jo feil. Likna ikkje den me fekk då på sånn den skal vere i ein sekskant? Eg trur kanskje me skal skrive 120 grader når me vil ha 60 grader…..eller?

K: 120 grader og 60 grader det er jo 180 grader….det er sikkert noko med det. Og så må den gå 100 steg til. Det ser ut til å funke. Me får ein sekskant trur eg. Me er gode!



Elevane har **generalisert** ved å ta i bruk erfaring frå førre oppgåve for å løyse utfordringa med å lage ein sekskant.

Også i arbeidet med å lage slettekode viser elev K teikn på forståing.

L: Nei. No må me lage slettekode. Me må ta vekk alt. Visk vekk alt når du trykke på «den» (peikar på flagg).

K: Me må ha ei sånn «gå til koordinat». Du må bruke «penn av» og altså.

L: Nei, me treng ikkje ta pennen vekk, for når katten går dit så forsvinn det. Me har jo skrive «slett alt».

K: Neeei. Du må ta «penn av» før, elles teiknar den mens den går. Det kan eg vedde på altså.

L: Eg kan vedde på at me ikkje treng.

K: Ser du, kva er det der? Det er ei linje. Pennen er jo nede når katten flytter på seg.

L: Ja, vel då. Du hadde rett. Eg fiksar.

Elevane diskuterer seg i mellom medan dei **prøver og feilar**. Elev L tvilar framleis på elev K, men ved å testa i Scratch, ser hen at elev K hadde rett. Denne gongen fungerte programmeringsspråket som **mediering** då utsagnet til elev K vart testa ut.

Elevparet vil starta på femkant, men då stoppar eg dei.

> Lærar: Gå tilbake til koden for å lage ein sekskant. Kva skjer når ein trykkjer «mellomrom» ?
> L: Katten teiknar ein sekskant.
> Lærar: Er du sikker på det?
> L: Ja, sjå då.

Eg ser at eleven først trykkjer ein gang på «mellomrom», før hen gjer det same fem gangar til. Dermed tar eg opp ein diskusjon me hadde i klassen før programmeringsprosjektet starta. Dette handla om korleis ein robotgrasklyppar fungerer.

> ***
> Lærar: Er det slik at den går 50 meter i ein retning, snur 60 grader, og så må nokon i huset trykkje for at den skal klyppe vidare?
> L: Nei, den er jo programmert på førehand.
> Lærar: Hadde koden de har laga då fungert til å programmere klypparen?
> L: Nei, det er sant
> Lærar: Kva må gjerast då?
> K: Me må sikkert gjere noko slik at me slepp trykkje «mellomrom» heile tida. Eg trur me må bruke….nei, eg veit me må bruke «gjenta-blokk». Eg sa jo det då me laga koden for trekant (ser på elev L).

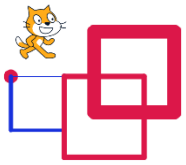
Elevane har **dekomponert** problemet til at ein sekskant består av seks sider, og at kvar av vinklane er 120 grader. Dei har likevel ikkje klart å byggje delane opp att slik at katten teiknar ein sekskant, noko nybyrjarar ofte får problem med (Grover & Pea, 2017). Dette dømet viser også at elevane ikkje heilt har taket på **automatisering**, då dei ikkje ser at eit program skal vere effektivt og spare oss for arbeid (Grover & Pea, 2017). Då elevane går vidare til å jobbe med regulær femkant, har dei med løkke frå start. Også denne gang er det **prøving og feiling** med vinklar, med utgangspunkt i å setje grader til ein stad mellom 60 og 120 ut frå kodar for trekant og sekskant.

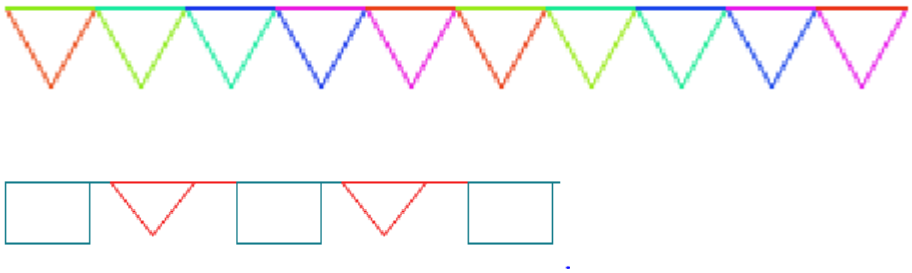
Sjølv om få elevar har jobba med oppgåve 2, vel eg å ta med korleis eitt par løyste det å lage kode på eiga hand i oppgåve 2e). Som elevparet over, vart også desse tekne ut på eige rom, og det vart gjort bandopptak. Denne gangen var eg til stades heile tida.

Oppgåve 2e) Programmer eit mønster

Programmer eit mønster som består av:

- Valfri geometrisk figur/figurar
- Løkker





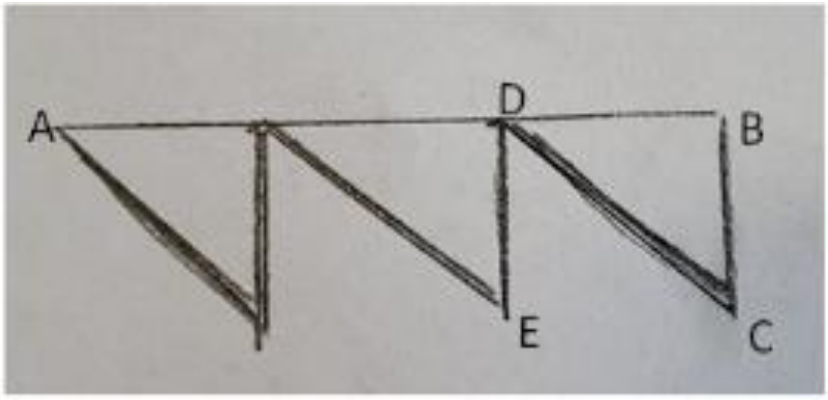
Planlegg mønsteret på papir, og skriv ned trinn for trinn (med ord) korleis du vil gå fram for å lage koden.

Ta bilete av mønster og framgangsmåten og lim inn her, eller lever inn på papir.

Lag koden i Scratch, og lim den og mønsteret ditt inn her:

Dersom du blir ferdig, kan du lage eit nytt mønster der du også brukar ulike fargar.

Gjennom 2a), 2c) og 2d) har elevane jobba med løkke i løkke, endring av farge og rotasjon.



Skisse

Biletet over viser mønsteret dette paret ville lage kode til. Eg har sett inn bokstavar i figuren for at det skal bli lettare å forstå kvar i mønsteret ein er i dialogen nedanfor.

Forklaring til korleis dei vil gjere det: «Først lagar me ein trekant med 90 grader oppe til høgre og ein spiss vinkel oppe til venstre og ein spiss ned. Så lagar me ei løkke slik at me får tre like trekantar». Elevane har dermed brote ned mønsteret i mindre delar for å løyse problemet (**abstraksjon og dekomponering**). Sjølv om forklaringa ikkje er presis, så har dei ein plan.
Biletet over vert heretter omtala som *skisse*.
Elevane startar med å opne prosjektet som ligg i oppgåve 2b:
https://scratch.mit.edu/projects/743858627

Deretter slettar dei koden, men held på slettekoden, før dei startar på oppgåva med å lage mønster.

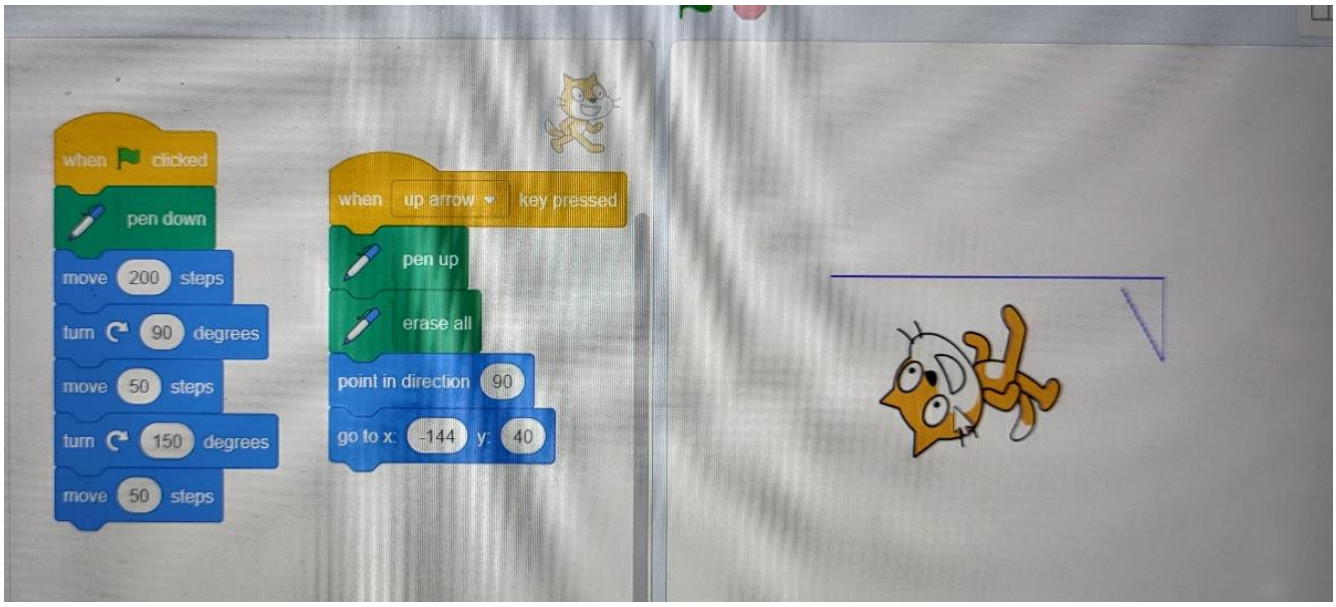
> C: Me må ha ei slik blokk som seier at når me trykkjer på flagget så startar det.
> D: Og så må me ha ein slik penn……ein som seier at katten skal teikne strekar. Er det denne? Peikar på *pen up*. Eller er det den andre? (refererer til *pen down*).
> C: Eg trur det er den andre.
> D: Og så skal den gå….var det den brikka her? Ja. Me kan ta at han går 200 steg. Han går veldig langt. Me tar litt mindre steg. Og så….kva var det den skulle etter det? …..og så 90 grader.

Sjølv om elevane har laga ei enkel forklaring på korleis dei skal lage trekanten, legg dei det bort, og arbeider etter **prøve- og feilemetoden**.

> C: Snu 90 grader mot høgre. Sånn.
> D: Den skal gå sånn. Peikar nedover.
> C: Men me kan jo gå heile vegen bort først. (Refererer til linjestykket merka AB i skissa).
> D: Visst me går 200 steg, og så snur 90 grader.

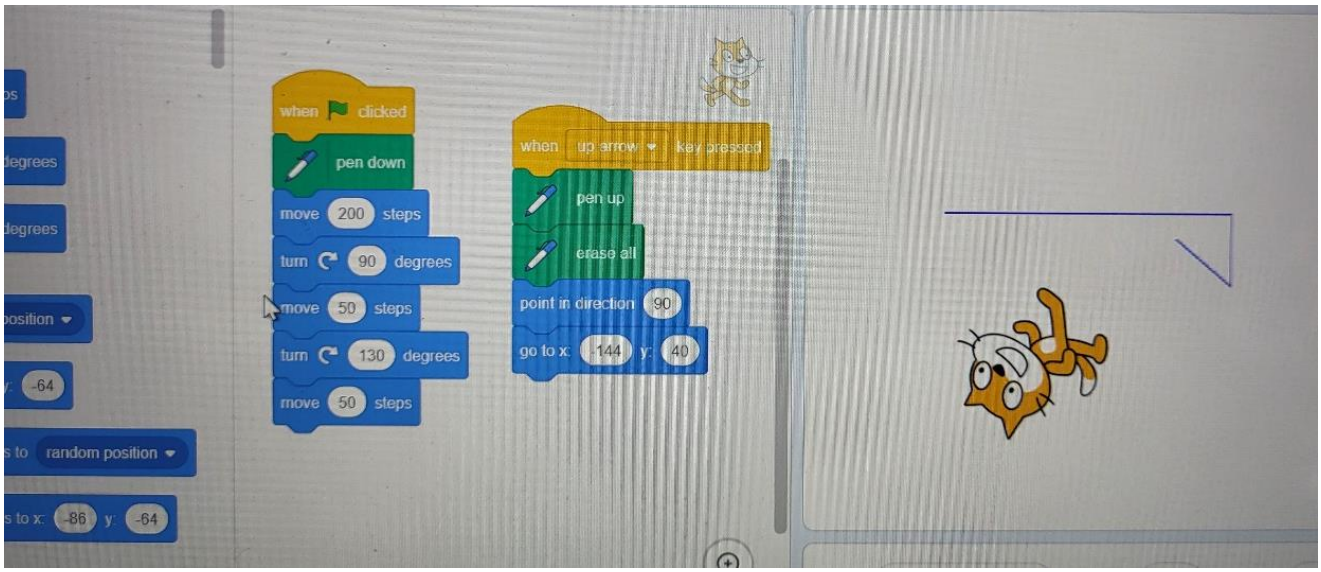
Elevane startar med å peike på første trekanten i mønsteret, og det verkar som om dei vil ta utgangspunkt i første trekanten i koden sin, men så kjem elev C kjem med ideen om å lage heile linjestykket AB, og elev D verkar å vere samd. Hen gir i alle fall ikkje uttrykk for å vere usamd. Så langt har dei nytta erfaring frå tidlegare oppgåver.

> C: og så kan me prøva med å gå 50 steg nedover (frå B til C i skissa) og så må den snu seg vidare den vegen. (Peikar mot D i skissa).
> Berre mykje meir enn 90 grader.
> D: Kanskje 150 grader?
> C: Trur det blir 150 grader, ja. Og så må den gå….. Ja. Sånn. Den kan gå 50 steg til. Og så prøver me om det fungerer. Om den har laga ein av trekantane.
> D: Flytt katten så me ser. Ha,ha. Trur ikkje det vart heilt rett.



Elevane har forsøkt å **dekomponere** problemet. Dei skriv litt kode, før dei **testar** det dei har gjort. Dei ser sjølv at dette ikkje vart rett, at trekanten var for liten til at linjestykket AB skulle gi plass til eksakt tre like trekantar.

C: Det vart feil. Kva skal me endre? Trekanten vart liten…..om me tar litt større vinkel først? (Endrar frå 150 til 130 grader). Sånn, og så kan me sjekke om det var rett.



C: Det ser jo ikkje så gale ut, berre at katten ikkje gjekk så langt som den skulle. Kanskje me kan ha fleire steg?

D: Visst me berre har litt meir steg, så trur eg det der blir bra.

Elevane har lært gjennom føregåande oppgåver at om skal katten teikne ein større vinkel, så må gradene dei set i koden vere mindre. Dette tolkar eg ut frå at dei gjorde ei endring utan å diskutere eller prøve seg fram. Vidare testar dei med ulikt antal steg. Justerer til dei er nøgde, og endar opp med 75 steg som ved augekast på skjermen ser bra ut. Dei seier seg dermed nøgde med første trekanten, og held fram med koden.

D: Men me må jo ha tre slike….

C: Kan du ikkje berre bruka «gjenta-blokka» då?

D: Jo. Viss me først har dei blokkene…..då er katten der (peikar på D i skissa), så kan me bruke «gjenta» derifrå etterpå…..då skal katten ende opp der (peikar på A i skissa).

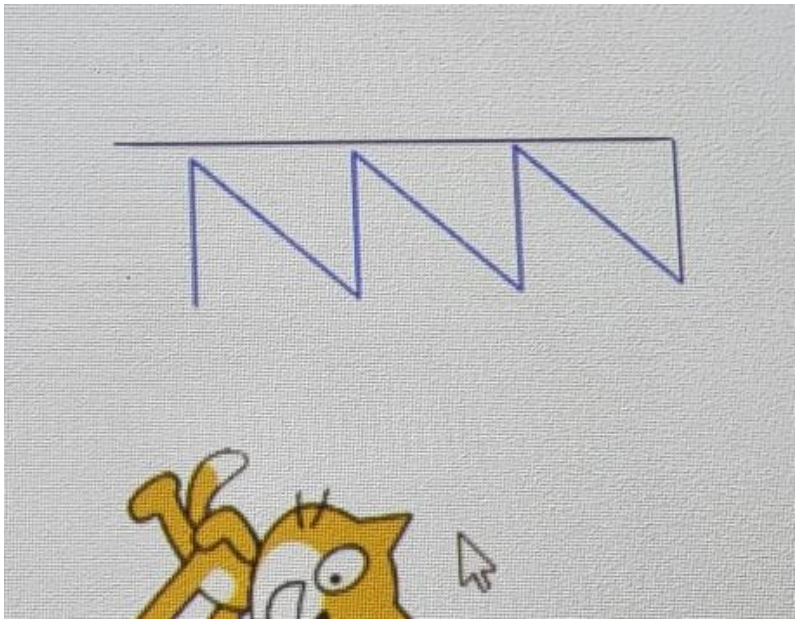
C: Men korleis skal me få katten til å snu retning og gå nedover att? (Mot E i skissa).

Elevane **prøver og feilar** mange gangar med ulike vinklar. Dei diskuterer med kvarandre om katten skal snu seg mot høgre eller venstre.

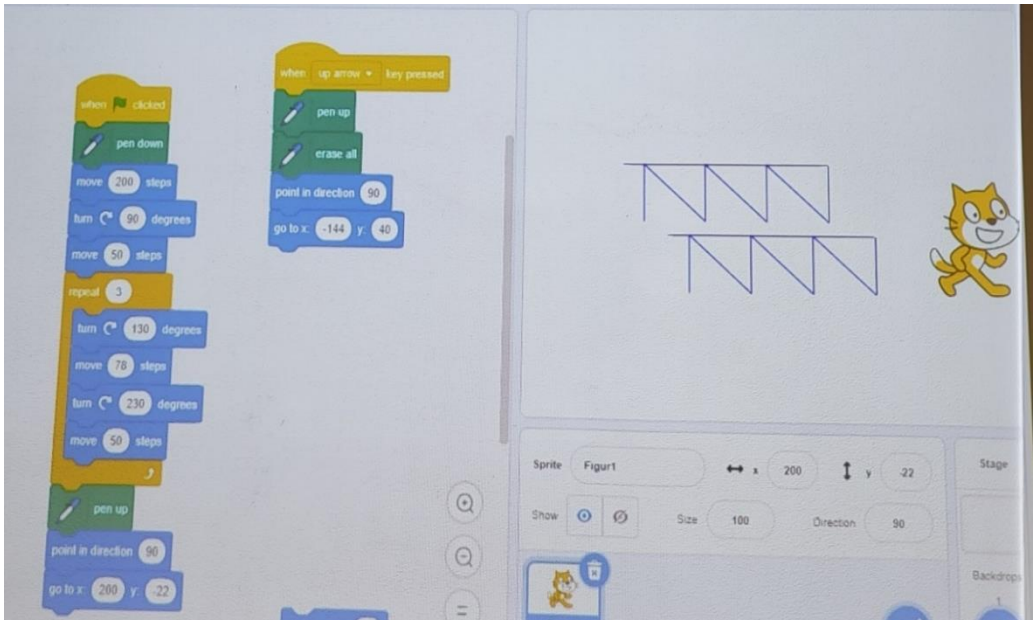
D: Kjør koden slik at me ser korleis katten står. Då er det lettare å sjå korleis den skal snu seg.

C: Ja, me må liksom få fjeset til å snu seg.

Elevane kjører koden, og brukar støtte frå programmet for å sjå korleis det må bli (**mediering**). Etter ein del **prøving og feiling** testar dei koden, og konkluderer med at 75 steg ikkje var nok. Katten nådde ikkje heilt ned til linjestykket (AB på skissa), og glipa vart større og større for kvar trekant som vart laga.

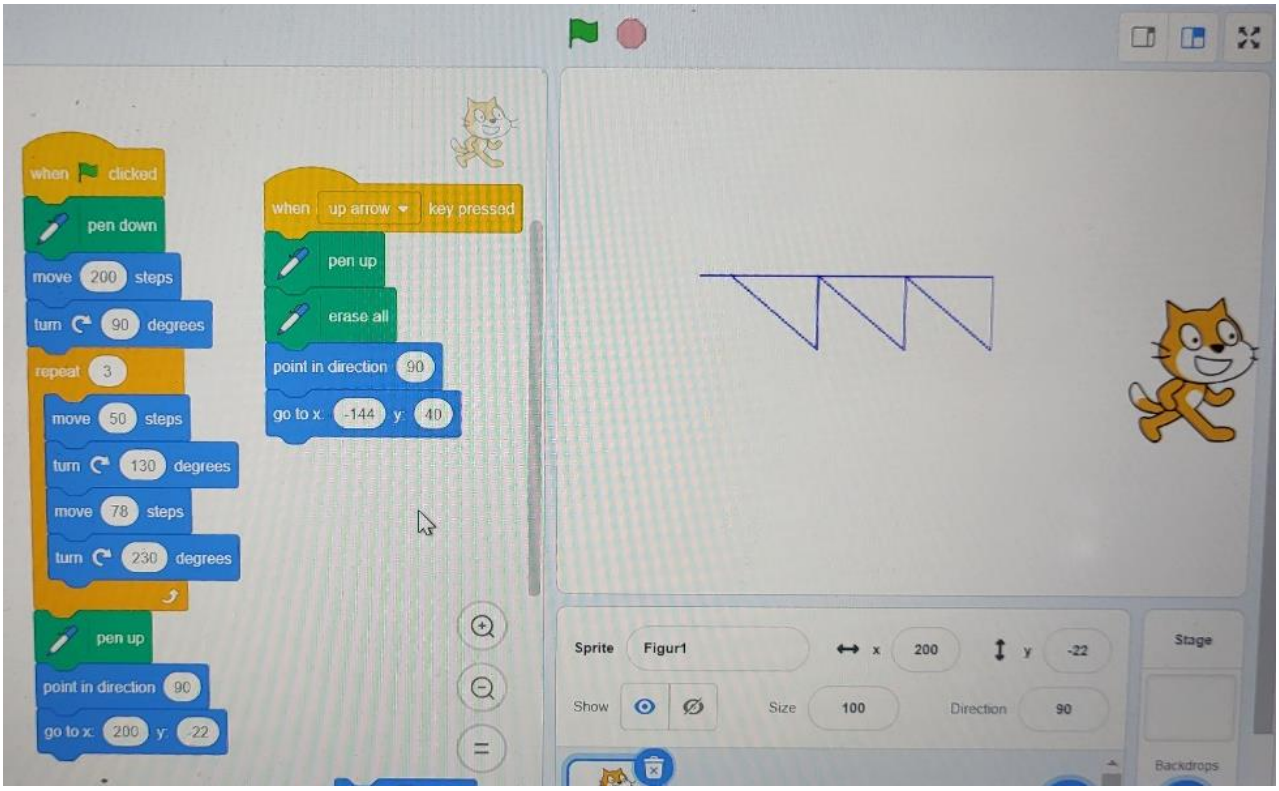


Biletet over viser resultatet med 75 steg.



Biletet viser resultatet etter at dei har justert til 78 steg.

Elevane er nøgde med mønsteret, bortsett frå at katten startar å teikne ein fjerde trekant.
Dei jobbar dermed litt vidare, og endrar på løkka slik at fjerde trekant ikkje vert byrja på, som vist i biletet under.



Som lærar har eg observert elevane heile tida medan dei jobba. Dialogen mellom elevane var bra, og eg avbraut dei ikkje i prosessen.
Då dei verkar å vera nøgde spør eg om dei har laga det så presist dei får til.

> C: Ja. Ser at den lengste linja (AB på skissa) er litt for lang, men det kan sikkert fiksast om me endrar 200 steg til for eksempel 175 eller noko slikt.

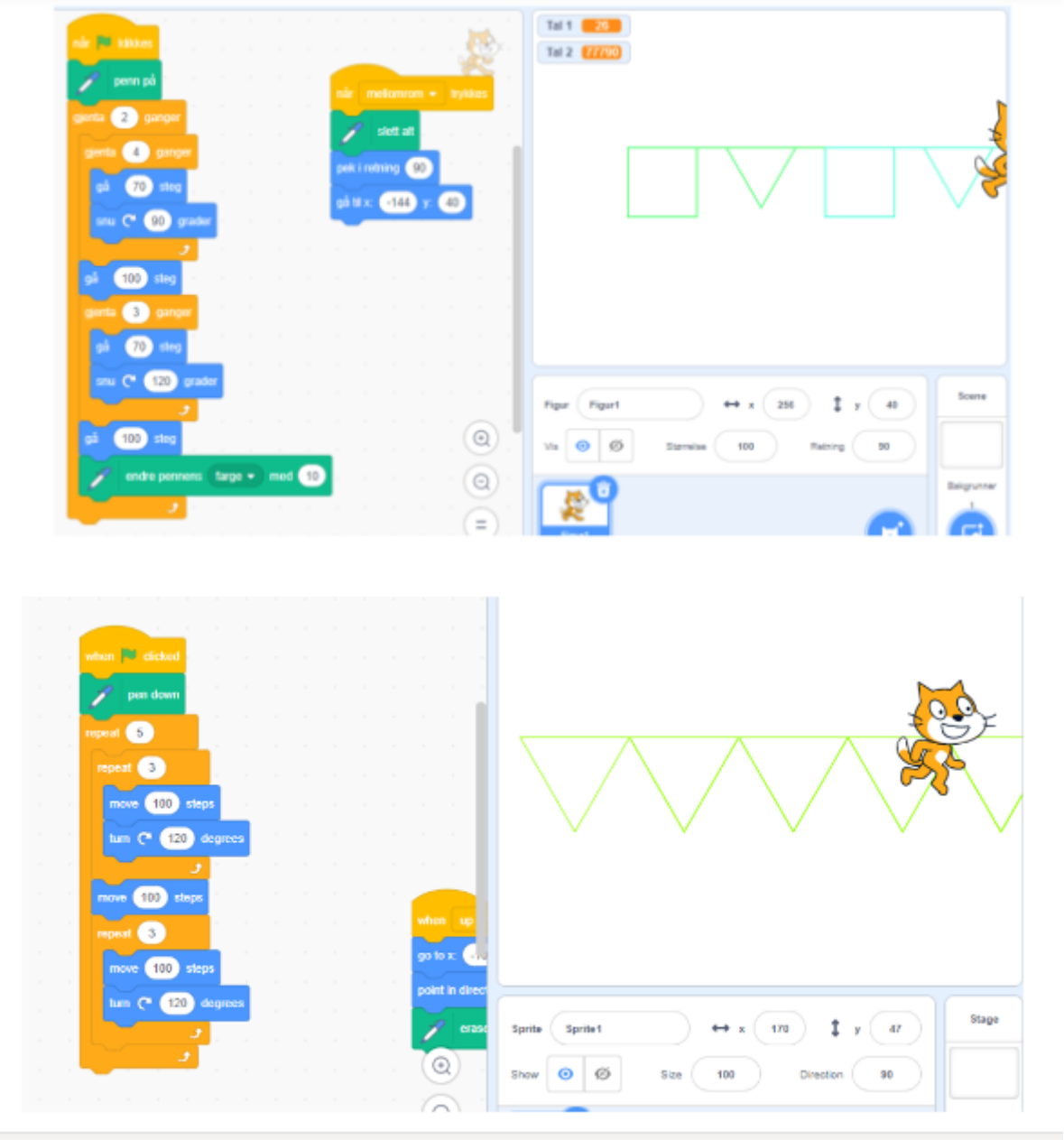
Eleven klarar å finne att kvar hen kan endre koden for at løysinga skal bli betre. I dette tilfellet mangla elevane matematisk kunnskap for korleis dei skulle få mønsteret rett, men det vart ikkje fokus frå mi side. I staden spurte eg om korleis denne koden kunne brukast dersom eg til dømes ønska å lage eit mønster bestående av fem trekantar.

> D: Det vil ikkje fungere med denne koden…….Eg tenkte jo eigentleg først at me kunne lage ein trekant, og så laga ei løkke for å repetere. Då kunne me ha laga så mange trekantar me ville bortover.
> Lærar: Ja, om eg tolkar den skriftlege forklaringa rett så er det planen.

> C: Trur me gløymde å sjå på den. Og så verka det så lurt å starte med den linja som var felles for alle trekantane. Eg tenkte litt på det undervegs, men eg ville ikkje gi opp, eg ville få det til.

Å jobbe med prosjekt er ein adaptiv prosess der ein jobbar, prøver ut og utviklar vidare (Brennan & Resnick, 2012). Bocconi et al (2018) og Weintrop et al (2015) framhevar også evna til å **handtere problem** og **uthalding**. Elevane C og D var svært uthaldande i prosessen. Elev C seier hen tenkte på å endre strategi undervegs, men at hen ville få det til. Dette kan også tyde på at eleven har oppdaga at det er fleire kodar som kan føre til same resultat, noko som seinare kan utviklast til diskusjon omkring kva som meinast med ein effektiv kode. Tolkinga mi er også at desse elevane har god kontroll på kva for brikker som utfører dei ulike stega, då dei gjekk direkte på blokka som måtte endrast når noko var feil.

Tar til slutt med to løysingar som vart levert på it´s learning. Desse elevane hadde ikkje levert skisse til mønster eller forklaring på korleis dei ville gå fram. Elevpara har gått ut frå likesida geometriske figurar, noko som gjer at dei kan ta i bruk kodar dei har jobba med tidlegare (**abstraksjon** og **generalisering**). Sjølv om oppgåve 2 d) inneheldt rotasjon, hadde ingen forsøkt seg på dette.



Som skrive i starten av kapittel 5 noterte eg stikkord om resultat frå elevarbeid, observasjon og transkripsjon av fokusgruppeintervju/bandopptak. Eg prøvde å sjå etter teikn på **abstraksjon, behandling av algoritmar, automatisering, dekomposisjon, generalisering, testing og feilsøking, samarbeid** og **evna til å handtere opne problem/uthalding** innanfor kvar av fasane i PRIMM-modellen. Etter at eg hadde gått gjennom alt elevarbeidet, var dette tabellen eg sat att med.

CT kjerneferdigheitar	Predict/ forutsei	Run/kjør	Investigate/ undersøk	Modify/ endre kode	Make/ laga kode
Abstraksjon	Elevar diskuterer om brikker er naudsynte eller ikkje. Elevane kjenner att delar av koden frå ting dei har lært i faget tidlegare.	Dersom hypotesen ikkje stemte.	Kva gjer *gjenta-blokka*? Kva skjer om ein endrar verdiar og/eller rekkjefølgje på blokker?	Elevane ser at dei kan bruke løkke. Elevane ser at dei kan bruke erfaringar frå tidlegare oppgåver.	Elevane nyttar abstraksjon når dei lagar eige program.
Behandling av algoritmar	Lese/forstå algoritmar.	Dersom hypotesen ikkje stemte.	Samanlikning av algoritmar. Endre rekkjefølgje på (plassering av) blokker.	Kva skal endrast, og kva skal behaldast?	Lage skriftleg forklaring. Lage kode.
Automatisering	Flagg klikkes/ pil opp trykkes. Øving på å sjå at ei handling fører til at ein kode vert kjørt.		Løkker	Løkker, når flagg/pil opp vert trykt.	Tar i bruk *når....klikkes* Tar med løkker
Dekomposisjon	Øve opp til ved å tolke kodar. Teikne skisse over kva som skjer.	Dersom hypotesen ikkje stemte.	Øve opp ved tolking av kodar. Øve opp ved å endre plassering av blokker.	Bryt ned for å sjå kva som skal endrast.	Bryt ned for å lage eigen kode

Generalisering	Kjenne att mønster frå matematikk dei har lært tidlegare.		Kjenne att mønster frå tidlegare oppgåver.	Kjenne att mønster frå tidlegare oppgåver.	Kjenne att mønster frå tidlegare oppgåver.
Testing og feilsøking Evaluering	Dersom første hypotese ikkje stemte.	Dersom hypotesen ikkje stemte.	Elevane testa ved å skrive lik kode i Scratch. Endra på verdiar i 2c.	Etterkvart som dei endra koden.	Heile vegen medan dei laga koden.
Samarbeid	Elevane samarbeider gjennom å diskutere kva koden utfører. Lærar støttar.	Dersom dei hadde feil. Elevane hentar oppgåve på Scratch community.	Elevane samarbeider medan dei utforskar. Lærar støttar. Elevane hentar oppgåve på Scratch community.	Elevane samarbeider medan dei endrar koden. Lærar støttar.	Elevane samarbeider medan dei lagar ny kode. Lærar støttar.
Evna til å handtere opne problem/uthalding	Elevane tok seg tid med oppgåva.	Elevane brukte tid på å finne kva som var feil.	Vart ikkje utfordra på uthalding. Oppgåve 2c fungerte ikkje mtp at elevane noterte kva som skjedde, men klarte ikkje å ta det med vidare.	Alle elevane jobba med oppgåva til den var løyst.	Alle elevane jobba med oppgåve 1e til den var løyst. Dei som fekk tid starta på ekstraoppgåvene før dei starta på oppgåve 2a. Elevar som jobba med 2e var svært uthaldande.

5 Drøfting

Korleis kjem algoritmisk tankegang til uttrykk hos elevane i dei ulike fasane av PRIMM?

I denne delen vil eg drøfte observasjon og elevsvar opp mot forskingsspørsmålet mitt, i tillegg til å knyte dette opp mot teori og forsking presentert i kapittel 2.

Elevane som er med i prosjektet har lite forkunnskap om programmering generelt og Scratch spesielt. I tillegg er omgrepet *algoritmisk tankegang* nytt for dei. Kan ein likevel vente å sjå spor av algoritmisk tenking i opplæringsfasen, og bruke programmering for å styrkje desse? Er undervisning etter PRIMM-modellen ein eigna metode for dette?

Som skrive i innleiingskapittelet var PRIMM ein undervisningsmodell eg oppdaga då eg i fjor var med på å skrive gruppeoppgåve om korleis det stod til med programmeringskunnskap og –undervisning hos lærarar ulike stadar i landet. Utgangspunktet var altså å bytte ut ein metodikk som eg erfarte ikkje fungerte, å la elevane jobbe med kurs på eiga hand, med ein metode som hadde vist seg å fungere for elevar og lærarar tidlegare (Sentance et al., 2019). Tankane var at modellen ville fungere med omsyn på å lære å programmere. I tillegg hadde eg ei hypotese om at PRIMM kunne vere til hjelp i utviklinga av algoritmisk tankegang slik som UMC tidlegare har vist å vere det (Lytle, et al., 2019), og at dette ville kome til syne i dei to siste fasane, altså *modify* og *make*.

Under analysedelen såg eg derimot at også dei tre andre delane, *predict*, *run* og *investigate,* eigna seg til dette arbeidet. I følgje Sentance et al. (2019) treng ein ikkje gjennomføre undervisningsopplegg der alle fasane er med. Ein kan like gjerne bruke ei heil økt berre på til dømes ein, to eller tre fasar, og også ta fleire slike rundar, før ein går vidare til andre fasar. Gjennom *predict*, *run* og *investigate* fann eg at elevane kan få god øving i **behandling av algoritmar** ved å lese og forstå ulike algoritmar dei vert presenterte for. Noko som er viktig med tanke på at dei etter kvart sjølv skal lære å lage effektive og presise program. Tolking av kodar gjer også at dei får sjå korleis problem kan **dekomponerast**, for så å byggjast opp att. Her var det elevar som **testa** hypotesen sin ved å teikne det dei trudde kom til å skje. Eg tenkjer i ettertid er at oppgåveformuleringa burde ha vore at dei skulle teikne forslaget sitt i staden for, eller tillegg til, skriftleg forklaring. **Generalisering** og **abstraksjon** kom til uttrykk i første fase ved at elevane kjente att kunnskap dei hadde frå geometri på tidlegare klassetrinn og brukte dette til å tolke koden. Vidare tok dei med seg mønster dei lærte frå denne delen til seinare fasar. Etter å ha undersøkt kodane i oppgåve 1c gjekk alle, bortsett frå to par, over til automatisk å bruke løkke i staden for å ta med alle steg. Dei starta også alle kodar med «når mellomrom/flagg/pil opp trykkes» (**automatisering)**.

I desse fasane, som i seinare fasar, var også **samarbeid** tydeleg då elevane diskuterte seg fram til kva koden utførte. Elevar som etter å ha kjørt koden såg at dei hadde feil hypotese, fekk øving i **testing og feilsøking** og **uthalding**. Her må ein likevel vere observant på at det kan vere elevar som ser at dei har feil hypotese, men som likevel ikkje er uthaldande nok til å sjekke kvar feilen ligg.

Run-fasen verka som å vere ein enten-eller fase med tanke på algoritmisk tankegang. Dersom elevane tolka sitt forslag som identisk med det katten utførte, såg det ut som om dei gjekk direkte vidare. Det vil seie at dei fleste ikkje tenkte over skilnaden mellom *firkant* i eige svar, og *kvadrat* som katten teikna. Dersom dei derimot hadde feil hypotese, eller som i oppgåve 2a) ikkje klarte å tolke heile koden, vart konsept frå algoritmisk tankegang teke i bruk.

Arbeidet med *predict* og *run* viste også at gjennomgang i heile klassen er naudsynt. Det var elevar som hadde tolka koden på sitt vis, testa og sagt seg nøgd (elevane G og H, K og L). Denne gangen oppdaga eg det ved å gå rundt å observere, men det kunne like gjerne ha skjedd at dei gjekk vidare med andre oppgåver medan eg observerte andre elevar.

Dei ulike konsepta i CT (avsnitt 2.3) vart alle jobba med i løpet av matematikkøktene og dei ulike fasane av PRIMM. **Testing og feilsøking** var det konseptet som kom tydlegast fram. Dette viste seg når elevane sjekka og retta opp i feil dersom hypotesen i *predict-delen* ikkje stemte, og kom enda betre til syne under *make-delen*. Elevane K og L sjekkar blokk for blokk om dei er på rett spor i oppgåve 1e), og brukar **logikk** og **generalisering** for å komme fram til kor stor vinkelen skal vere i ein femkant og sekskant. «Me sjekkar» gjekk også att hos elevane C og D. Dei testa undervegs til dei kom fram til eit løysingsforslag for korleis lage ein kode for mønsteret dei hadde teikna i oppgåve 2e). Medan elevane jobba med feilsøking, fekk dei øving i **abstraksjon**, **dekomposisjon** og **behandling av algoritmar.** Også elevane eg observerte i klasserommet verka å ha kontroll på kva for blokker som skulle endrast i denne prosessen. Dei gjekk ikkje tilfeldig inn og endra til dømes verdien i ei blokk, men snakka om å endre vinkelen, og gjekk rett på «snu»-blokka. Det same viste seg når det var steglengde som skulle justerast. Det var med andre ord ikkje tilfeldig kva for blokker eller verdiar dei endra, men gjennomtenkte forslag slik eg opplevde det.

Scratch gjer elevane moglegheit til å sjekke etter kvart steg i koden og få respons utan opphald, og dette kan vere medverkande til at dei **heldt så lenge ut** i arbeidet sitt. Og uthalding er noko Weintrop et al. (2015) inkluderer i omtalen av ferdigheitar som inngår i CT. Erfaring frå arbeid med problemløysingsoppgåver med penn og papir når det gjeld elevane i prosjektet, og tidlegare elevar eg har hatt, er at dei viser lite uthalding. Eg diskuterer innimellom med kollegaer at forholdet mellom farten på internett og uthalding hos elevar er omvendt proporsjonal. Det å bryte ned problem krev resonnering og er ein kognitivt krevjande operasjon, og mi erfaring er at størsteparten

av elevane gir opp om dei ikkje ser løysinga etter kort tid. Dette skjedde ikkje medan dei jobba med oppgåvene knytt til dette prosjektet. Alle elevane som deltok jobba godt gjennom økter med varigheit på 1,5 time utan å spørje etter ein liten pause, noko dei gjer til vanleg. Dei haldt seg også til oppgåvene dei skulle utan å gå vidare før dei var nøgde med løysingsforslaget sitt. Ikkje før slutten av siste økt var det elevar som testa andre funksjonar i Scratch enn det oppgåvene la opp til. Denne observasjonen var for meg overraskande. Før eg starta prosjektet tenkte eg mykje att og fram på om elevane skulle få ut alle oppgåvene på same tid, eller om dei skulle porsjonerast ut ei og ei. Eg landa på det første med tanke på at eg skulle få observere utan å bli forstyrra av elevar som var klare for ny oppgåve, men antok at det ville vere elevar som hoppa over steg i PRIMM av di dei støtte på vanskar. Dette skjedde ikkje. Truleg er elevane si uthalding eit resultat av at Scratch er intuitivt å bruke, og at programmet gjer det lett å prøve og feile, men eg ser det og som mogleg at den stegvise progresjonen i PRIMM-rammeverket støttar opp om uthalding hos elevane.

Oppgåve 1d) i *modify-delen* og oppgåve 1e) og 2e) i *make-delen* oppfordra elevane til å lage ein skriftleg algoritme (**behandling av algoritmar**) for korleis dei skulle lage dei ulike kodane. Dette viste seg å vere vanskeleg, noko det også gjorde i før-fasen til programmeringsøktene (avsnitt 3.5.1). Fleire elevpar laga forklaringa etter at dei hadde laga koden. Nokre starta på forklaringa utan å gjere den ferdig, men elevparet K og L seier tydeleg «Send inn du (om løysingsforslaget), så skriv me den greia (forklaringa) etterpå.» I ettertid tenkjer eg at lite fokus på skriftleg forklaring kan vere fordi Scratch er så intuitivt og lett å bruke ved prøving og feiling at dei ikkje har trong for å lage ei slik skildring, men at dei gjer det i etterkant fordi oppgåva seier at dei skal gjere det. Det kan tenkjast at dette vert meir naudsynt når elevane går i gang med vanskelegare oppgåver, eller går over til tekstprogrammering. I så fall er det viktig at dei får øve på oppgåver som krev dette, til dømes ved analog programmering, slik at dei får god trening på behandling av algoritmar.

Jo Boaler (2016) har forska på korleis tradisjonell matematikkundervisning kan påverke motivasjonen for faget. Ho fann at samfunnet er prega av at matematikk er noko du enten kan eller ikkje kan, ei haldning ein ikkje ser i andre fag. Å gjere matematikk blir å følgje reglar vist av lærar, og å forstå matematikk vert å hugse ein regel å bruke denne når lærar spør. Beviset på at du har lært er når ein får rett svar (Skott, Skott, Jess, & Hansen, 2019). Dette er ikkje ei forståing av faget som styrkjer algoritmisk tankegang, der prosessen er sentral. Slik eg ser det, vil kurs innan programmering der elevane jobbar på eiga hand, eller tavleundervisning der elevane skriv av kodar lærar viser, vere å lære programmering i tradisjonell ånd. Gjennom undervisning etter PRIMM-rammeverket blir prosessen derimot viktigare. Elevane blir aktive i læringssituasjonen, og rammene for å utvikle og styrkje algoritmisk tankegang etter mitt syn betre.

Som skrive i avsnitt 2.5, er PRIMM-rammeverket influert av Vygotskij og den sosiokulturelle læringsteorien (Sentance et al., 2019). **Samarbeid** er også noko Bocconi et al. (2018), Brennan & Resnick (2012) og Grover & Pea (2017) legg vekt på, og knyter CT, eller algoritmisk tankegang , tett opp til dette synet på læring. Det er av verdi å designe noko saman med andre, og ein får til meir om ein samhandlar. Forrige skuleår jobba klassen eg då hadde med programmering gjennom kurs. Elevane sat med kvar sin PC, og sjølv om dei hadde lov å diskutere, vart det minimalt av dette. Kurset la ikkje opp til den type arbeid. Resultatet vart som skrive i innleiinga at elevane hadde manglande forståing for programmering i ettertid. I denne studien såg eg derimot mykje samarbeid, og godt samarbeid. Det var ikkje slik at ein elev jobba, medan den andre såg på. I staden utfylte dei kvarandre, og diskuterte seg fram til løysingsforslag. Til dømes diskuterer elevane C og D seg fram til kva «penn på» tyder. Den eine eleven kjem med forslag den andre eleven meiner kan vere rett, men som hen gjerne ikkje hadde blitt merksam på om hen sat åleine. Hypotesen vart testa ut, og dei fekk ei felles forståing av kva denne blokka eller delen av koden utførte. Ein del av programmeringsspråket vart gjennom kommunikasjon og mediering til indre tale for begge elevane (Imsen, 1984; Sentance et al., 2019). Dette er også i tråd med korleis Morten Munthe anbefaler å jobbe innanfor programmering (TEKNA, 2021). I planleggingsfasen hadde eg ei vurdering av om elevane skulle sitje med kvar sin PC eller ikkje. Eg enda på at dei skulle dele. Dette verka å vere som ei rett vurdering i dette tilfellet. Etter kvart som elevane får auka kompetanse innan programmering, og kanskje skal jobbe saman om større prosjekt, vil derimot å sitje med kvar sin PC truleg vere eit meir naturleg val.

Sjølv om samhandling kan føre til læring og felles forståing, finn ein også døme på at denne felles forståinga elevane får kan vere feil. Elevane E og F hadde funne at «gjenta-blokka» førte til at det vart teikna fire kvadrat. Dei verka å vere samstemte om dette, og hadde nok ikkje oppdaga feilen om dei ikkje fekk rettleiing frå lærar. Dei godtok også at testing av denne koden gjorde at det i følgje seg sjølv vart teikna to kvadrat ved sida av kvarandre, utan å sjekke dette vidare. At elev I kunne ha vore ei støtte for elev J i oppgåva der dei skulle lage ein kode for ein regulær mangekant, fungerte heller ikkje, då utsegna om at «skal me ha ei «gjenta-blokk» eller?» ikkje vart teke omsyn til. Dette viser at sjølv om elevane jobbar godt saman i par og grupper, er det viktig med felles gjennomgang i klassen av løysingsforslag, og tankar omkring kvifor dei har valt å løyse oppgåva slik dei har gjort. På denne måten får elevane ta del i ulike måtar å løyse problem på, og kan opparbeide kunnskap dei kan ta med vidare i arbeidet med å gjere programmeringsspråket til eit indre språk. Det er etter mitt syn også av verdi å sjå etter ulike forslag for løysingar for å diskutere kva som er den mest effektive løysinga på ulike problem.

Sidan elevane er heilt i startfasen med programmering, var ikkje oppgåvene designa med tanke på samhandling ved å ta i bruk delar av program laga av andre (bortsett frå det dei fekk av lærar), eller ved å dele problemet opp i mindre delar og jobbe med kvar sin del. Oppgåvene var laga med tanke på at dei skulle vera innafor elevane si *Zone of Proximal Development* (Skott et al, 2019). Oppgåve 1 fungerte her. Elevane hadde bakgrunnskunnskap i geometri som støtta dei då dei skulle tolke kodar og var til hjelp då dei laga eigne kodar. Ikkje alle elevane kom i gang med oppgåve 2. Denne oppgåva vart litt for utfordrande med tanke på rotasjon. Bakgrunnskunnskapen mangla og dei fekk dermed lite støtte frå tidlegare erfaring (Skott et al, 2019). Resultatet her hadde truleg vorte annleis dersom me hadde stoppa opp ved *predict, run* og *investigate***,** for klasseromsdiskusjonar med lærar som støtte. Ein kunne også gjerne tatt fleire rundar med desse tre, eller berre *investigate* før ein gjekk vidare med *modify* og *make***.** På denne måten kunne ein ha sikra at elevane forstod kva som låg bak alle delane av koden dei undersøkte (Sentance et al., 2019).

6 Konklusjon og avsluttande del

I denne studien har eg sett på korleis algoritmisk tankegang kan komme til uttrykk i dei ulike fasane av undervisningsmodellen PRIMM. Eg har knytt dette opp mot teori og tidlegare forsking som er aktuell for studien, som sosiokulturell læringsteori, algoritmisk tankegang og programmering ved hjelp av Scratch. PRIMM er ein ny undervisningsmodell (Sentance et al., 2019), som gir elevane auka læringsutbytte når det kjem til opplæring innan programmering. Elevane får og ei rikare forståing for kjernen i programmering (Sentance et al., 2019). Sidan PRIMM er ei vidareutvikling av UMC, som viser å engasjere elevar i CT (Sentance et al., 2019), er det nærliggjande å tru at også PRIMM gjer dette. Eg finn derimot ikkje forsking som viser korleis algoritmisk tankegang kjem til uttrykk i dei ulike fasane av modellen.

Ut frå analyse og etterfølgjande drøfting, meiner eg å ha funne at algoritmisk tankegang kan komme til uttrykk i alle fasar av PRIMM, og at progresjonen i modellen kan føre til at dei ulike konsepta vert styrkja etter kvart som elevane arbeider seg gjennom dei ulike fasane. Elevane tolkar først ferdiglaga kode, før dei testar og evaluerer hypotesen sin. Vidare får dei i oppgåve å endre på ein kode slik at resultatet av å køyre koden vert litt annleis enn utgangspunktet var. Til slutt skal alt elevane har jobba seg gjennom munne ut i at dei lagar sin eigen kode frå botnen av. I dei første fasane av PRIMM får elevane dermed trening i dei ulike konsepta som inngår i algoritmisk tankegang gjennom å diskutere med læringspartnaren sin. I dei siste fasane får elevane vise ferdigheitane og styrkje desse ved å endre ferdiglaga kode og lage sin eigen kode.

Før eg gjekk i gang med studien hadde eg eit håp om at undervisningsmodellen PRIMM for opplæring av programmering skulle fungere som eit betre alternativ enn kurs og avskriving frå tavle. I tillegg hadde eg eit håp om at eg skulle finne ut at kjerneelement og kompetansemål kunne jobbast med frå ein starta opplæring i programmering. Dette slik at programmering ikkje berre blir nok ein ting ein skal undervise i utan fleire timar til rådvelde. Etter å ha gjennomført prosjektet har eg fått ei meir positiv haldning til programmering i matematikkfaget. Sidan kjerneelementet problemløysing handlar om at elevane skal utvikle løysingsmetodar på problem dei ikkje kjenner frå før (Utdanningsdirektoratet, 2020) vert algoritmisk tankegang viktig i prosessen for å utvikle desse strategiane og framgangsmåtane når ein jobbar med oppgåver i klasserommet. Då dette også er viktig når ein jobbar med programmering (Grover & Pea, 2017), kan det å bruke programmering til å utvikle algoritmisk tankegang vere ein god metode, noko eg meiner min studie bekrefta. I tillegg fann eg at ein ikkje treng vere erfarne programmerarar før ein går i gang med kjerneelement og andre læreplanmål knytta opp mot programmering og algoritmisk tankegang. Ein kan starte med

opplæring på same tida som ein jobbar med andre mål, og PRIMM verkar å vera ein god metode for dette, i alle fall fungerte det for mine elevar.

I avsnitt 2.2 skreiv eg at forsking etter forsøket med programmering i skulen på 1980-talet viste at kompetansen elevane fekk gjennom programmering i lita grad let seg overføre til andre område og problemstillingar (Dolonen et al., 2019). Det kunne vore av interesse å sjå om dette er annleis no når programmeringsspråka er betre, utstyr meir tilgjengeleg og fokus på at lærarar må inneha programmeringskompetanse er auka. Vil elevar etter ei tid med programmering utvikla ferdigheiter som betre set dei i stand til å til dømes løyse problem med penn og papir? Vil dei klare å bryte ned problemet slik at dei kjem fram til ei løysing? Vil dei vere like uthaldande? Klarer dei å overføre kompetansen dei opparbeider seg ved hjelp av programmering til andre område?

Vidare vil det vere interessant å sjå om PRIMM også fungerer når elevane skal til med tekstprogrammering. Vil bruk av PRIMM gi same resultat i Python som i Scratch? Eller er det bruk av Scratch åleine som gjer at eg fekk dei resultata eg fekk, at desse hadde vore like uansett kva for metodikk ein nytta i undervisninga.

Meir forsking må nok til før ein kan svare på desse spørsmåla, og inntil det kjem forsking som motseier mine funn, kjem eg til å undervise etter PRIMM-modellen.

7 Referansar

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

Boaler, J. (2016). *MATHEMATICAL MINDSETS: Unleashing Student´s POTENTIAL Through Creative Math, Inspiring Messages and INNOVATIVE TEACHIN.* JOSSEY BASS A Wiley Brand.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). *The Nordic approach to introducing Computational Thinking and programming in compulsory education.* Report prepared for the Nordic@BETT2018 Steering Group. Hentet fra https://doi.org/10.17471/54007

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education : implications for policy and practice.* EUR 28295 EN. Hentet fra https://data.europa.eu/doi/10.2791/792158

Bocconi, S., Chioccariello, A., Kampylis, P., Dagienė, V., Wastiau, P., Engelhardt, K., . . . Stupurienė, G. (2022). *Reviewing computational thinking in compulsory education : state of play and practices from computing education.* Publications Office of the European Union. Hentet fra https://op.europa.eu/en/publication-detail/-/publication/bbf875ec-a5a2-11ec-83e1-01aa75ed71a1/language-en

Bogdan, R. C., & Biklen, S. K. (1982). *Qualitative Research for Education: an Introduction to Theory and Methods* (3. utg.). Boston: Allyn & Bacon. Hentet fra http://math.buffalostate.edu/dwilson/MED595/Qualitative_intro.pdf

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vol.1*. Hentet fra http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Bråting, K., & Kilhamn, C. (2021). Exploring the intersection of algebraic and computational thinking. *Mathematical Thinking and Learning, 25*(2), ss. 170-185. doi:https://doi.org/10.1080/10986065.2020.1779012

Bueie, H. (2019). *Programmering for MATEMATIKKLÆRERE.* Universitetsforlaget.

Cappelen, D. (u.d.). *Skolen*. Hentet fra https://skolen.cdu.no/

Creswell, J. W. (2007). *Qualitative inquiry and research design: Choosing among five traditions* (2. utg.). Sage publications, Inc. Hentet fra https://revistapsicologia.org/public/formato/cuali2.pdf

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

Det kongelige kirke- utdannings- og forskningsdepartement. (1996). *Læreplanverket for den 10-årige grunnskolen.* Nasjonalt læremiddelsenter. Hentet fra https://www.nb.no/items/f4ce6bf9eadeb389172d939275c038bb?page=0

Dolonen, J. A., Kluge, A., Litherland, K., & Mørch, A. (2019, 11 8). *Litteraturgjennomgang av programmering i skolen.* UiO: Institutt for pedagogikk. Det utdanningsvitenskapelige fakultet. Hentet fra https://www.duo.uio.no/bitstream/handle/10852/76290/Litteraturgjennomgang%2bav%2bprogrammering%2bi%2bskolen%2b-%2bfinal.pdf?sequence=2&isAllowed=y

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2020). *Computational thinking in programming with Scratch in.* Computer Applications in Engineering Education, 2021;29:12–28. doi:https://onlinelibrary.wiley.com/doi/10.1002/cae.22255

Fangen, K. (2022). Kvalitativ metode. De nasjonale forskningsetiske komiteene. Hentet fra https://www.forskningsetikk.no/ressurser/fbib/metoder/kvalitativ-metode/

Gjøvik, Ø., & Torkildsen, H. A. (2019). Algoritmisk tenkning - tidsskrift for matematikkundervisning. *Tangenten, 30*(3). Hentet fra http://tangenten.no/wp-content/uploads/2021/12/Tangenten-3-2019-Gjovik-Torkildsen.pdf

Gretter, S., & Yadav, A. (2016). Computational thinking and Media & Information Literacy: An Integrated Approach to Teaching Twenty-First Century Skills. *TechTrends, 60*(5), ss. 510-516. Hentet fra https://doi.org/10.1007/s11528-016-0098-4

Grover, S., & Pea, R. (2013). Computational thinking in K-12 A Review of the State of the Field. *Educational Researcher, 42*(1), ss. 38-43. doi:10.3102/0013189X12463051

Grover, S., & Pea, R. (2017). Computational Thinking: A Competency Whose Time Has Come. I *Computer Science Education.* doi:10.5040/9781350057142.ch-003

Gyldendal. (u.d.). *Salaby*. Hentet fra https://www.salaby.no/

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

Haagensen, C., Krogsæther, V., & Tennfjord, B. L. (2022). “Hvordan er tilstanden i arbeidet med implementering av programmering i matematikkfaget ". Oppgåve i MA932, UIA, Kristiansand, Agder Fylkeskommune.

Imsen, G. (1984). *Elevens verden: Innføring i psykologi.* Aschehoug: Tano. Hentet fra https://urn.nb.no/URN:NBN:no-nb_digibok_2007071104071

Jiang, B., Zhao, W., Gu, X., & Yin, C. (2021). Understanding the relationship between computational thinking and computational participation: a case study from Scratch online community. *Education Tech Research Dev, 69*, ss. 2399-2421. Hentet fra https://doi.org/10.1007/s11423-021-10021-8

Kaufmann, O. T., & Stenseth, B. (2021). Programming in mathematics education. *International Journal of Mathematical Education in Science and Technology, 52*(7), ss. 1029-1048. Hentet fra https://doi.org/10.1080/0020739X.2020.1736349

Kongsnes, A. L., & Wallace, A. (2020). *Matemagisk 8.* Aschehoug Undervisning.

Kozulin, A. (2003). Psychological Tools and Mediated Learning. I *Vygotsky's Educational Theory in Cultural Context* (ss. 15-38). Cambridge University Press.

Larsen, A. K. (2017). *En enklere metode. Veiledning i samfunnsvitenskapelig forskningsmetode* (2. utg.). Fagbokforlaget.

Learnlink . (u.d.). Hentet fra Hva er algoritmisk tenkning?: https://www.youtube.com/watch?v=fy-MYqsq8ro

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., . . . Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, Volume 2*(Issue 1), ss. 32-37. Hentet fra https://doi.org/10.1145/1929887.1929902

Lytle, N., Cateté, V., Boulden, D., Dong, Y., Houchins, J., Milliken, A., . . . Barnes, T. (2019). Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for STEM Classes. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ss. 305-401. Hentet fra https://doi.org/10.1145/3304221.3319786

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

Norge Kirke-og undervisningsdepartementet. (1987). *Mønsterplan for grunnskulen: M87.* Kirke- og undervisningsdepartementet. Aschehoug. Hentet fra http://urn.nb.no/URN:NBN:no-nb_digibok_2007080200101

Norstein, A., & Haara, F. (2021). *Matematikkundervisning i en digital verden.* Cappelen Damm AS.

NOU. (2013 : 2). *Hindre for digital verdiskaping.* Norges offentlige utredninger. Digitutvalget. Hentet fra https://www.regjeringen.no/contentassets/e2f0d5676e144305967f21011b715c16/no/pdfs/nou201320130002000dddpdfs.pdf

NOU. (2020). *Fremtidige kompetansebehov.* Regjeringen. Hentet fra https://www.regjeringen.no/contentassets/053481d65fb845be9a2b1674c35d6d14/no/pdfs/nou202020200002000dddpdfs.pdf

NRK- innlandet. (2021, Desember). Skal lære elevene koding, men forstår det ikke selv. Hentet fra https://www.nrk.no/innlandet/laerere-trenger-hjelp-til-a-knekke-koden-pa-koding-1.15781343

Papert, S. (2020). *MINDSTORMS children, computers, and powerful ideas.* Basic Books.

Postholm, M. B. (2004). Kvalitativ forskning på praksis. Fra opprinnelse til forskerfokus. *Norsk pedagogisk tidsskrift, Vol.88*(1), ss. 3-18. Hentet fra https://www.idunn.no/doi/10.18261/ISSN1504-2987-2004-01-02

Postholm, M. B., & Jacobsen, D. (2019). *Læreren med forskerblikk. Innføring i vitenskapelig metode for lærerstudenter* (1.utgave, 10.opplag. utg.). Cappelen Damm.

Postholm, M. B., & Moen, T. (2018). *Forskings-og utviklingsarbeid i skolen. En metodebok for lærere, studenter og forskere* (2. utg.). Universitetsforlaget.

PRIMM. (u.d.). Hentet fra https://primmportal.com/

Resnick, M., Silverman, B., Maloney, J., Monroy-Hernández, A., Rusk, N., Kafai, Y., . . . Maloney, J. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), ss. 60-67. Hentet fra https://www.researchgate.net/publication/265435403_Scratch_Programming_for_Everyone

Røsseland, M. (2005a). Hva er matematisk kompetanse -del 1. *Tangenten*, ss. 12-18. Hentet fra http://tangenten.no/wp-content/uploads/2021/12/t-2005-1.pdf

Sander, K. (2022). Casestudie. Hentet fra https://estudie.no/casestudie/

Sanne, A., Berge, O., Bungum, B., Jørgensen, E., Kluge, A., Kristensen, T., . . . Voll, L. (2016). *Teknologi og programmering for alle - en faggjennomgang med forslag til endringer i grunnopplæringen.* Utdanningsdirektoratet. Hentet fra https://www.Utdanningsdirektoratet.no/globalassets/filer/tall-og-forskning/forskningsrapporter/teknologi-og-programmering-for-alle.pdf

Sentance, S., Waite, J., & Kallia, M. (2019). Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education, 29: 2-3*, ss. 136-176. Hentet fra https://doi.org/10.1080/08993408.2019.1608781

Sevik, K., & m.fl. (2016). Programmering i skolen. Hentet fra https://www.Utdanningsdirektoratet.no/globalassets/filer/programmering_i_skolen.pdf

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, ss. 142-158. doi:https://doi.org/10.1016/j.edurev.2017.09.003

Skott, J., Skott, C. K., Jess, K., & Hansen, H. (2019). *Matematikk for lærerstuderende DELTA 2.0, Fagdidaktikk 1.-10.klasse* (2. utg.). Samfundslitteratur.

Statped. (2021). Programmering. Hentet fra https://www.statped.no/laringsressurser/teknologitema/programmering-for-barn-med-saerskilte-behov/programmering/programmeringssprak/

Sullivan, L. E. (2009). The SAGE Glossary of the Social and Behavioral Sciences. *SAGE Publications, Inc*, s. 64. Hentet fra https://web.s.ebscohost.com/ehost/ebookviewer/ebook/ZTAwMHh3d19fNDc0Njg1X19BTg2?sid=43a3ed62-63e6-445c-8840-84b7f5a8ac97@redis&vid=1&format=EB&rid=1

TEKNA. (2021). Programmeringsdidaktikk. Hentet fra https://www.tekna.no/fag-og-nettverk/realfag-og-utdanning/realfagsbloggen/programmeringsdidaktikk/

Tofteberg, G. N., Tangen, J., Bråthe, L. T., Stedøy, I., & Alseth, B. (2021). Maximum 9. (2). Gyldendal .

Universitetet i Stavanger. (2022). Utvikling av algoritmisk tenking gjennom programmering med Scratch i grunnskolen. Hentet fra https://www.uis.no/nb/kunnskapssenter-for-utdanning/ressurser/utvikling-av-algoritmisk-tenking-gjennom-programmering-med

Utdanningsdirektoratet. (2006). Læreplan i matematikk fellesfag. Hentet fra Læreplan i matematikk fellesfag: https://www.Utdanningsdirektoratet.no/kl06/MAT1-04/Hele/Grunnleggende_ferdigheter

Utdanningsdirektoratet. (2019). Algoritmisk tenkning. Hentet fra https://www.Utdanningsdirektoratet.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-

kompetanse/algoritmisk-tenkning/#:~:text=Algoritmisk%20tenkning%20er%20en%20probleml%C3%B8sningsmetode.%20Algoritmisk%20tenkning%20inneb%C3%A6rer,informatiker%27%20n%C3%A5r%20vi%20skal%20

Utdanningsdirektoratet. (2020). Læreplan i matematikk 1.–10. trinn (MAT01-05). Hentet fra https://www.Utdanningsdirektoratet.no/lk20/mat01-05

Utdanningsdirektoratet. (2021). *Hvorfor har vi fått nye læreplaner?* Hentet fra https://www.Utdanningsdirektoratet.no/laring-og-trivsel/lareplanverket/stotte/hvorfor-nye-lareplaner/

Utdanningsdirektoratet. (2022). Hentet fra Digitalisering i grunnopplæringen Utvikling av algoritmisk tenkning i grunnskolen: https://www.Utdanningsdirektoratet.no/tall-og-forskning/finn-forskning/rapporter/utvikling-av-algoritmisk-tenking-gjennom-programmering-i-grunnskolen/

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2015). Defining Computational Thinking for Mathematics and Science. *25*, ss. 127-147. Hentet fra https://ccl.northwestern.edu/2015/Weintrop%20et%20al.%20-%202015%20-%20Defining%20Computational%20Thinking%20for%20Mathematics%20an.pdf

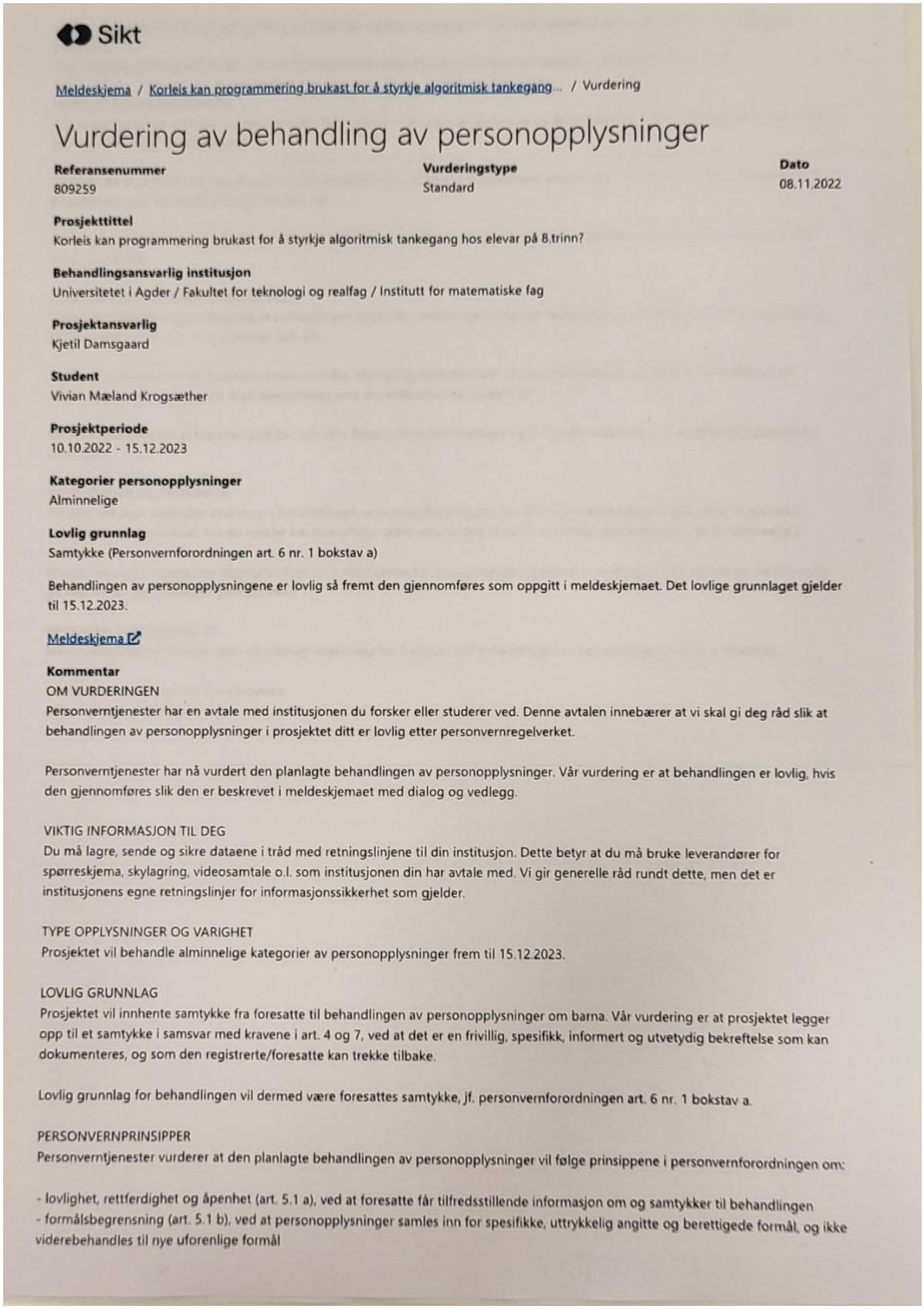
Wing, J. (2014). Computational Thinking Benefits Society. I *Social Issues in Computing.* Hentet fra http://socialissues.cs.toronto.edu/index.html%3Fp=279.html

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM, 49*(3), ss. 33-35. Hentet fra https://doi.org/10.1145/1118178.1118215

Wood, D., Bruner, J., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry, 17*(2), ss. 89-100.

8 Vedlegg

8.1 Vedlegg 1, NSD-skjema



Sikt

Meldeskjema / Korleis kan programmering brukast for å styrkje algoritmisk tankegang... / Vurdering

Vurdering av behandling av personopplysninger

Referansenummer
809259

Vurderingstype
Standard

Dato
08.11.2022

Prosjekttittel
Korleis kan programmering brukast for å styrkje algoritmisk tankegang hos elevar på 8.trinn?

Behandlingsansvarlig institusjon
Universitetet i Agder / Fakultet for teknologi og realfag / Institutt for matematiske fag

Prosjektansvarlig
Kjetil Damsgaard

Student
Vivian Mæland Krogsæther

Prosjektperiode
10.10.2022 - 15.12.2023

Kategorier personopplysninger
Alminnelige

Lovlig grunnlag
Samtykke (Personvernforordningen art. 6 nr. 1 bokstav a)

Behandlingen av personopplysningene er lovlig så fremt den gjennomføres som oppgitt i meldeskjemaet. Det lovlige grunnlaget gjelder til 15.12.2023.

Meldeskjema

Kommentar
OM VURDERINGEN
Personverntjenester har en avtale med institusjonen du forsker eller studerer ved. Denne avtalen innebærer at vi skal gi deg råd slik at behandlingen av personopplysninger i prosjektet ditt er lovlig etter personvernregelverket.

Personverntjenester har nå vurdert den planlagte behandlingen av personopplysninger. Vår vurdering er at behandlingen er lovlig, hvis den gjennomføres slik den er beskrevet i meldeskjemaet med dialog og vedlegg.

VIKTIG INFORMASJON TIL DEG
Du må lagre, sende og sikre dataene i tråd med retningslinjene til din institusjon. Dette betyr at du må bruke leverandører for spørreskjema, skylagring, videosamtale o.l. som institusjonen din har avtale med. Vi gir generelle råd rundt dette, men det er institusjonens egne retningslinjer for informasjonssikkerhet som gjelder.

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige kategorier av personopplysninger frem til 15.12.2023.

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

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

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

- lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at foresatte får tilfredsstillende informasjon om og samtykker til behandlingen
- formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke viderebehandles til nye uforenlige formål

- dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet
- lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

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

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

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

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

Ved bruk av databehandler (spørreskjemaleverandør, skylagring, videosamtale o.l.) må behandlingen oppfylle kravene til bruk av databehandler, jf. art 28 og 29. Bruk leverandører som din institusjon har avtale med.

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

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til oss ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde:
https://www.nsd.no/personverntjenester/fylle-ut-meldeskjema-for-personopplysninger/melde-endringer-i-meldeskjema. Du må vente på svar fra oss før endringen gjennomføres.

OPPFØLGING AV PROSJEKTET
Personverntjenester vil følge opp ved planlagt avslutning for å avklare om behandlingen av personopplysningene er avsluttet.

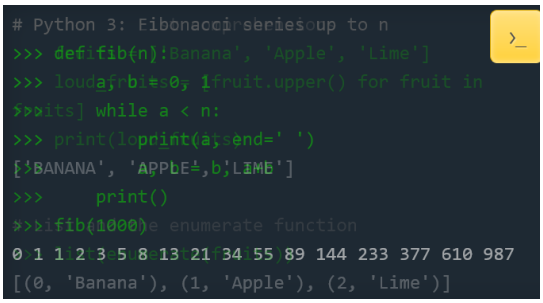
Kontaktperson hos oss: Markus Celiussen
Lykke til med prosjektet!

8.2 Vedlegg 2, Spørsmål før prosjektet

Programmering i matematikk

1. Har du jobba med programmering?_____
2. Dersom ja, kva for program kjenner du til?

Scratch_____



Python_____

Andre? Forklar_____

3. korleis jobba de med programmering? Set eit kryss ved det som stemmer for deg.

   Kodetimen/Lær kidsa koding

   «Kurs» på Kikora (eller liknande) der du fekk beskjed steg for steg

   Skreiv av det lærar gjorde på tavla

   Oppgåver i læreboka der eg måtte laga koden sjølv

4. Kven av desse kodane teiknar eit rektangel? Det KAN vera fleire.

a)

b)

c)

d)

8.3 Vedlegg 3, Oppgåve 2c)

Oppgåve 2c) Undersøk

Dersom resultatet i 2b) ikkje viste det same som det du tenkte i 2a), klarar du å finne ut kvar du tenkte feil?

Skriv her:

Hadde du delar av koden rett?

Skriv her:

Trykk på «Se inni».

Sjå på blokka «endre pennens farge med 10». Kva skjer om ein tar denne ut av løkka som skal gjentakast 24 gongar, og plasserar den rett under løkka? Kjør gjerne koden fleire gongar.

Dette skjer om blokka står inne i løkka:

Dette skjer om blokka står under løkka:

Dette skjer om blokka står under løkka og koden vert kjørt fleire gongar:

Endra steglengde i den 5.blokka til eit tal mellom 3 og 50. Korleis påverkar det koden?

Forklar:

8.4 Vedlegg 4, Samtykke

Vil du delta i forskingsprosjektet

Korleis kan programmering brukast for å styrkje algoritmisk tankegang hos elevar på 8.trinn?

Føremål

Dette året skal eg skriva master i matematikkdidaktikk, og i den samanheng ynskjer eg å undersøkja korleis programmering kan brukast for å styrkja den algoritmiske tankegangen hos elevane.

I den nye læreplanen, LK20, vart programmering innført som ein del av m.a matematikkfaget. Ein grunn til det er at kunnskap om programmering er viktig for å læra, arbeida og leva i morgondagens samfunn . Ein annan grunn er å utvikla gode verktøy for problemløysing. Ei algoritme er ei oppskrift for korleis løysa eit problem, og algoritmisk tankegang handlar om å utvikla metodar for å løysa problem, metodar som kan overførast til andre fag/område. (Norsk senter for IKT i utdanningen, 2016).

Kven er ansvarleg for forskingsprosjektet?

Universitetet i Agder er ansvarleg for prosjektet.

Kvifor får du spørsmål om å delta?

Sidan du er elev i ein av klassane eg underviser i matematikk og programmering er noko me skal jobba med (kompetansemål på 8. trinn), håpar eg at du har lyst å vera med i dette prosjektet. Eg trur at det blir kjekke timar på skulen, og set pris på om du vil hjelpa meg med oppgåva mi.

Kva inneber det for deg å delta?

Det inneber ikkje meir arbeid for deg å delta enn det du elles skal jobba med i matematikktimane. Det vert oppgåver du skal løysa saman med ein eller to andre på papir/its-learning og PC (programmering i Scratch). Me kjem til å bruka matematikktimane i ei til to veker.

Dersom du vel å delta, kan du i tillegg få spørsmål om eg kan få ta lydopptak medan du og partnaren din jobbar med/diskuterar oppgåvene.

Det er frivillig å delta

Det er frivillig å delta i prosjektet, altså å vera ein del av utvalet i undersøkinga, og det får ingen

negative konsekvensar for deg om du ikkje har lyst å bli med. Arbeidet du gjer på skulen blir det same, uansett deltaking eller ei. Dersom du vel å delta, kan du når som helst trekkje samtykket tilbake utan å gje nokon grunn. Det vil seie at om det er gjort lydopptak blir dette sletta. Oppgåver du har svara på i timane vert heller ikkje brukt i prosjektet om du trekkjer deg.

All informasjon er konfidensiell og blir i oppgåva anonymisert slik at ikkje noko skal kunna rettast tilbake til deg.

Ditt personvern – korleis vi oppbevarer og bruker opplysingane dine

Opplysningar om deg vil berre bli brukt til dei føremåla fortalt om i dette skrivet. Dei vert behandla konfidensielt og i samsvar med personregelverket. Innleveringar, lydopptak og oppgåve vert lagra adskilt og kan ikkje koplast. Personopplysningar vert ikkje tilgjengelege for andre enn meg som lærar.

Kva skjer med opplysingane dine når vi avsluttar forskingsprosjektet?

Lydopptak blir sletta når prosjektet er avslutta, seinast 15.12.23. Oppgåver på It´s learning vil bli liggjande så lenge du går på ungdomsskulen, med mindre du ynskjer at desse skal slettast. Oppgåver på papir vert makulert seinast 15.12.23

Kva gjev oss rett til å behandle personopplysingar om deg?

Vi behandlar opplysingar om deg basert på samtykket ditt.

På oppdrag frå Universitetet i Agder har NSD – Norsk senter for forskningsdata AS vurdert at behandlinga av personopplysingar i dette prosjektet er i samsvar med personvernregelverket.

Dine rettar

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

- innsyn i kva opplysingar me behandlar om deg, og å få utlevert ein kopi av opplysingane
- å få retta opplysingar om deg som er feil eller misvisande
- å få sletta personopplysingar om deg
- å sende klage til Datatilsynet om behandlinga av personopplysingane dine

Dersom du har spørsmål til studien, eller om du ønskjer å vite meir eller utøve rettane dine, ta kontakt med:

- Vivian Mæland Krogsæther (vivian.krogseter@bomlo.kommune.no)
- Universitetet i Agder ved Kjetil Damsgaard (kjetil.damsgaard@uia.no)

Dersom du har spørsmål knytt til Personverntjenester si vurdering av prosjektet kan du ta kontakt med:

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

Venleg helsing

Kjetil Damsgaard Vivian M Krogsæther

Rettleiar *Student*

Du treng berre levera tilbake denne sida:

Samtykkeerklæring

Eg har motteke og forstått informasjon om prosjektet *Korleis kan programmering brukast for å styrkje*

algoritmisk tankegang hos elevar på 8.trinn? og har fått høve til å stille spørsmål.

Eg samtykker til:

- ☐ å delta i lydopptak med spørsmål omkring oppgåvene det vert jobba med
- ☐ å delta i at oppgåver levert på It´s learning skal kunna brukast anonymisert i prosjektet
- ☐ å delta i at oppgåver levert på ark kan brukast anonymisert i prosjektet

Eg samtykker til at opplysingane mine kan behandlast fram til prosjektet er avslutta.

Dato og underskrift elev

Dato og underskrift føresett