Universitetet i Stavanger

FAKULTET FOR UTDANNINGSVITENSKAP OG HUMANOIRA

MASTEROPPGAVE

Studieprogram: Masteroppgave i matematikk, grunnskolelærerutdanning 5-10	Vårsemesteret, 2023
Forfatter: Lars Midttun Vestbøstad	
Veileder: Sean Peter Kåss Martin	
Tittel på masteroppgaven: Programmering i matematikk: En casestudie om elevers problemløsningsprosesser i introduksjon til programmering i matematikkfaget Engelsk tittel: Programming in mathematics: A case study of students problem solving processes in the introduction to programming in mathematics	
Studiepoeng: 45	
Emneord: Programmering i matematikk, Python, matematisk tenkning, algoritmisk tenkning, casestudie, tekstbasert programmering, matematikk didaktikk	Sidetall: 79 sider + vedlegg/annet: 108 sider Stavanger, 02.06.2023 dato/år

FORORD

Denne masteroppgaven markerer slutten på fem år som grunnskolelærerstudent ved Universitetet i Stavanger. Det har vært fem krevende, men lærerike år der jeg har fått muligheten til å opparbeide meg mye god erfaring og mange gode venner. Prosessen med å skrive en masteroppgave har vært meget krevende, og det er derfor på sin plass å takke de som har bidratt.

Først vil jeg takke min kone Andrea for utallige timer med korrekturlesing og gode tilbakemeldinger, i tillegg til å være støttende, omsorgsfull og tålmodig. Takk for at du har holdt ut med meg denne våren, denne oppgaven hadde ikke blitt gjennomført uten deg. Deretter vil jeg takke min familie som har stått meg i ryggen, og heiet på meg i hele prosessen. Mine medstudenter Ottar, som også er min bror, og Stine fortjener også en stor takk for både mental støtte under masteroppgaven, men også utrolig mye god hjelp og samarbeid gjennom årene på studiet.

Videre vil jeg takke min veileder Sean Peter Kåss Martin for all den gode hjelpen han har gitt meg, selv om ting ikke alltid så like lyst ut. Til slutt er det viktig for meg å takke elevene som ønsket å delta i studien min, læreren som har hjulpet meg i prosessen og alle kollegaene mine for masse god støtte.

Tusen hjertelig takk!

Stavanger, 02. juni 2023

Lars Midttun Vestbøstad

SAMMENDRAG

LK20 introduserte programmering inn i flere fag i skolen i 2020. Matematikk står som det faget der programmeringen har størst relevans, og det er derfor interessant å undersøke nytteverdien til programmering i matematikk. Blokkprogrammering blir mye brukt som introduksjon til programmering, gjerne i form av Scratch. Studien ønsker derfor å undersøke hvordan introduksjon til tekstbasert programmering påvirker elevenes evne til problemløsning i matematikk. Jeg har undersøkt hvilke problemløsningsprosesser som kan observeres i elevenes introduksjon til tekstbasert programmering, og prøvd å besvare forskningsspørsmålet:

Hvilke problemløsningsprosesser kommer til uttrykk i elevers introduksjon til tekstbasert programmering i matematikk på ungdomstrinnet?

Denne studien er en kvalitativ casestudie der jeg har studert ni ungdomsskoleelever i 10. klasse i arbeid med matematiske tekstbaserte programmeringsoppgaver i programmeringsspråket Python. Datamaterialet er innhentet gjennom lyd- og skjermopptak av tre elevgrupper over to uker. Informantenes samtaler og programmering har vært grunnlaget for studiens resultater. Oppgavene elevene har fått var designet slik at problemene elevene fikk underveis, hovedsakelig var knyttet til kodeforståelse.

Det teoretiske rammeverket er i hovedsak knyttet til matematisk og algoritmisk tenkning, ettersom begge er knyttet til problemløsning og i forlengelse, programmering. Gjennom resultatene har elevene vist tegn til både matematiske og algoritmiske tankerekker, og tyder derfor på at programmeringen bidrar positivt til at elevene får øve på og forbedre sine problemløsningsferdigheter.

INNHOLDSFORTEGNELSE

OVERSIKT OVER FIGURER OG TABELLER

1 INNLEDNING

Vi befinner oss midt i en teknologisk æra i konstant endring, der vi som deltakere i samfunnet må kjempe for å holde tritt med den digitale verden. Disse endringene er med på å forme både samfunnet rundt oss og hver enkelt person, uavhengig om de selv ønsker det eller ikke. Dette er deler av grunnene til at flere og flere land har sett seg nødt til å fokusere mer på kunnskap og ferdigheter rettet mot digital teknologi. Ettersom skolen gir barn og unge muligheter til å forberede seg på hvordan hverdagen og fremtiden kommer til å være, bør teknologi og den digitale verden være en relevant del av skoleløpet. Elever skal ha krav på å få utvikle kunnskap og ferdigheter som er relevante for å kunne delta aktivt i samfunnet, samt få mulighet til å kjenne på gleden av å utforske og skape (Opplæringslova, 1998).

1.1 BAKGRUNN FOR STUDIEN

Ifølge Balanskat og Engelhardt (2014) har flere land innført programmering i skolen, et engasjement som har vært spesielt sterkt i Europa. De viser til at alle landene som deltok i deres undersøkelse allerede hadde implementert programmering i skolen, bortsett fra Norge. Først i 2016 ble det innført prøvevalgfaget «programmering» (Meld. St. nr. 28 (2015-2016)), før det i 2020 ble innført ny læreplan, «Læreplan for Kunnskapsløftet 2020» (Kunnskapsdepartementet, 2019) (heretter forkortet til LK20). Gjennom LK20 ble programmering endelig innført i skolen, i tillegg til det som på engelsk kalles *Computational Thinking*, som vi på norsk har valgt å kalle algoritmisk tenkning. Derfor, fra 2020, har alle grunnskolene i Norge måtte innføre programmering gradvis i fagene der kompetansemålene tilsier det. 2022 var det første året der alle som går i norsk grunnskole får undervisning i programmering, ettersom de som startet på skolen i 2003 var det siste kullet som fulgte gammel læreplan ut av videregående skole.

Algoritmisk tenkning legges frem av Utdanningsdirektoratet som en problemløsningsmetode, der man systematisk angriper et problem (Kunnskapsdepartementet, 2019). Det er mange forskjellige definisjoner på hva som ligger i begrepet algoritmisk tenkning, men i bunn og grunn går det ut på å lage løsninger som både mennesker og datamaskin kan løse, og drar dermed mye inspirasjon fra informatikken (Bocconi, Chioccariello, Earp, & Group, 2018). Det må derimot ikke forveksles med det engelske begrepet *algorithmic thinking* som er en underkategori av CT, og må da være en underkategori av det norske begrepet algoritmisk tenkning (Gjøvik & Torkildsen, 2019).

1.2 BAKGRUNN FOR FORSKNINGSSPØRSMÅL

Programmering er knyttet tett opp til algoritmisk tenkning, og i LK20 finner vi kompetansemålet «utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering» (Kunnskapsdepartementet, 2019). Ettersom programmering ligger tett opp til algoritmisk tenkning og algoritmisk tenkning er en problemløsningsmetode, kan dette tyde på at problemløsning også ligger tett opp til programmering. Programmering er fortsatt ganske nytt i skolen og introduseres ofte gjennom blokkprogrammering. Tekstbasert programmering legger opp til en større dybdeforståelse av hvordan programmet og algoritmene fungerer og vurderes av Haraldsrud, Sveinsson og Løvold (2020) som den mest effektive måten å lære og undervise programmering. Jeg valgte først å undersøke hvilke problemløsningsstrategier elevene brukte i programmeringen, men så det som mer interessant underveis i studien å heller undersøke selve prosessene. Dette ledet fram til forskningsspørsmålet mitt:

Hvilke problemløsningsprosesser kommer til uttrykk i elevers introduksjon til tekstbasert programmering i matematikk på ungdomstrinnet?

For å besvare dette spørsmålet vil jeg presentere et teoretisk rammeverk som består av problemløsning og matematisk tenkning, før jeg beskriver teori knyttet til algoritmisk tenkning og programmering. Studien baseres på lyd- og skjermopptak av 9 elever på 10. trinn som arbeidet i grupper med programmeringsoppgaver i et tekstbasert programmeringsspråk de ikke hadde tidligere erfaring med.

2 TEORI

I dette kapittelet presenteres de teoretiske rammene som jeg mener er essensielle for å kunne besvare studiens forskningsspørsmål. Jeg vil redegjøre for generell problemløsningsteori og Pólyas (2004) problemløsning. Deretter presenterer jeg matematisk tenkning og dens inndelte elementer. Ettersom algoritmisk tenkning er tett knyttet opp til programmering beskriver jeg i kapittel 2.2. en oversikt over algoritmisk tenknings teoretiske inndeling, før jeg belyser programmering i skolen i dag og presenterer studiens valgte programmeringsspråk.

2.1 PROBLEMLØSNING

Ifølge Mason (2016) er et problem et problem når personen som står overfor problemet opplever det som problematisk og angriper problemet for å få det til å gi mening. Hvilke utfordringer eller oppgaver som er problemer er dermed individualisert. Problemløsningsbegrepet har vært mye omstridt og vanskelig å bestemme. Men som en generell tanke er problemløsning en samlebetegnelse på prosessene en person gjennomgår for å finne en løsning på et problem (Schoenfeld, 2016). Halmos (1980) introduserer tanken om at kjernen av matematikken er problemer, vi må derfor lære bort hvordan vi problemløser.

> … , that the mathematician's main reason for existence is to solve problems, and that, therefore, what mathematics really consists of is problems and solutions. (Halmos, 1980, s. 519)

> … I do believe that problems are the heart of mathematics, and I hope that as teachers, in the classroom, in seminars, and in the books and articles we write, we will emphasize them more and more, and that we will train our students to be better problem posers and problem solvers than we are. (Halmos, 1980, s. 524)

Skolen ønsker å kunne utdanne gode problemløsere, men med en spredt forståelse av begrepet problemløsning vil målet være vanskeligere å nå (Schoenfeld, 2016). Storparten av forskningen rundt problemløsning det siste halve århundret baseres i hovedsak på forskningen gjort av George Pólya. I hans tekst *How to solve it* fra 1945 introduserer han begrepet «moderne heuristikk» for å forklare problemløsningens kunst (Polya, 2004). Han så på problemløsning som en aktivitet, og mente at det var essensielt med deltakelse, å få en hands-on-opplevelse av

matematikken. Du må, ifølge Pólya, delta i oppdagelsene, hvor mye skjer gjennom gjettverk. Han endrer blikket fra at matematikken er en rigid, formell disiplin, til en som ligner mer på de fysiske vitenskapene som er avhengige av gjetninger, innsikt og oppdagelse:

> Mathematical facts are first guessed and then proved, and almost every passage in this book endeavors to show that such is the normal procedure. If the learning of mathematics has anything to do with the discovery of mathematics, the student must be given some opportunity to do problems in which he first guesses and then proves some mathematical fact on an appropriate level. (Pólya, 1954, sitert i Schoenfeld, 2016, s.6)

Schoenfeld (2016) henviser til Pólya og hans tanker om at hvordan elever opplever matematikk må stå i stil til hvordan vi gjør matematikk. Tanken om å se verden med matematiske øyne og lære elevene å tenke matematisk står sterkt hos Schoenfeld. Elevers matematiske utdannelse i skolen er i stor grad definert av kulturelle og sosiale omstendigheter. Hvordan elever og lærere ser på hva matematikk er, er med på å endre det matematiske miljøet i klasserommet. Vanligvis ser vi på matematikk som et fag som er nokså svart og hvitt; å kunne komme frem til og svare det korrekte svaret. Dette er en kulturell forventning som er godt representert i klasserommene i dag (Schoenfeld, 2016). Lampert (1990) skiller mellom å *gjøre* matematikk, altså å følge reglene du har lært, og å *kunne* matematikk, både å huske hva du har lært og å kunne bruke det. Den matematiske sannheten som dannes i klasserommene er definert når læreren godkjenner og avkrefter svar, og ved elevers iakttagelse av dette gjennom flere år i skolen (Lampert, 1990). Pólya presentere tanken om at problemløsningen krever mye mer gjetning og matematisk innsikt, heller enn formell, deduktiv disiplin. Ved dette trekker han matematikken lengre i retningen mot utforskingen vi finner i fysikken og den naturfaglige verden (Schoenfeld, 2016).

2.1.1 Pólyas problemløsning

Pólya (2004) presenterer hvordan vi tenker i møte med et problem og hvilke strategier som dukker opp i dette møtet. Han vektlegger hvor viktig det er for elever å få opparbeidet erfaring gjennom å jobbe med eget arbeid. Lærerens rolle vil være å veilede elevene gjennom denne prosessen uten å gi for tydelig svar, da eleven skal sitte igjen med følelsen av å ha løst problemet selv (Polya, 2004). Han presiserer at veiledningen fra læreren er meget viktig, og bør bestå av generelle hint som etter hvert blir mer og mer spesifikke om nødvendig. Hvordan

problemløsningsprosessen utarter seg for hver elev, avhenger av hvordan læreren og andre medelever påvirker eleven (Polya, 2004).

For å løse et problem er det nødvendig å gjentatte ganger endre oppfatningen av problemstillingen. Oppfatningen endres underveis i problemløsningen, ettersom vi blir mer kjent med problemet (Polya, 2004). Pólya deler inn problemløsning i fire forskjellige steg som jeg ønsker å presentere i mer detalj:

- Forstå problemet
- Legg en plan
- Utfør planen
- Se tilbake/reflekter

Alle de fire forskjellige stegene er avhengige av hverandre, bør gjennomføres i den gitte rekkefølgen, men fungerer som en syklus (Polya, 2004). Prosessen begynner med å forstå problemet du har foran deg. Hvis elevene skal kunne løse problemet, må de først forstå hva slags problem de står overfor. I tillegg til dette er det nødvendig at eleven selv har et ønske om å finne en løsning på problemet. Dette medfører at det er meget krevende å lage gode problem som elevene ikke kun forstår, men i tillegg ønsker å løse (Polya, 2004).

Ifølge Mason og Davis (1991) angriper mange elever problemet før de har lest hele oppgaveteksten og satt seg godt inn i problemet. De mener at det er nødvendig at læreren er tydelig og hjelper elevene med å forstå hvorfor de skal ønske å løse problemet. Læreren må ikke anta at de klarer dette på egenhånd, uten bistand (Mason & Davis, 1991). Videre i prosessen poengterer Pólya (2004) at det nå vil være viktig å legge en plan for gjennomføringen av løsningen til problemet. For å kunne legge en plan må eleven først jobbe med ideer som kan føre til en plan. Pólya (2004) vektlegger at eleven er helt avhengig av kunnskap om temaet problemet ligger under, for å være i stand til å utvikle en plan. Hvis du ikke har nok forkunnskaper til å løse problemet du står overfor, er det sannsynlig at du stopper opp på dette punktet, selv om du kanskje forstår problemet i steg én. Lærerens oppgave i planleggingssteget vil være å bistå eleven i å finne frem til gode ideer uten å gi eleven løsningen. For å bli en bedre planlegger i

møte med problemer er det viktig å ha flere tidligere erfaringer med lignende problem som du kan hente informasjon og ideer fra (Polya, 2004).

I steg tre av Pólyas (2004) problemløsningstrinn er målet å gjennomføre planen eleven la i steg to. Dette steget krever mye mindre av eleven enn de foregående stegene. Eleven bør derimot ha god tålmodighet til å stegvis gjennomføre planen. Lærerens rolle vil her være å passe på at eleven holder seg innenfor planens rammer. Pólya (2004) fremhever at dette vil være enklere dersom eleven selv har utformet planen. Videre vil det være viktig for læreren å passe på at eleven utfører alle delene av planen, og at eleven beviser hvorfor utførelsen av planen er riktig. Selv om dette steget avslutter problemet er det viktig å ikke hoppe over det siste trinnet, men å ta seg tid til å se tilbake og reflektere over problemløsningen (Polya, 2004).

Pólya (2004) poengterer tydelig at ved å se over løsningen, planleggingen og veien til mål er det mye elevene kan lære. Elevene kan arbeide lenge med et problem før det ikke er noe de kan lære av det. Selv om de sier seg ferdig med problemet er det veldig viktig at de forstår og legger til minne hvordan kom frem til en løsning. Dette er essensielt hvis de skal bli flinkere problemløsere i fremtiden. Store deler av problemløserens arsenal er tidligere erfaringer og kunnskap dannet gjennom problemløsningsoppgaver. Pólya (2004) understreker at hvis elevene umiddelbart legger arbeidet bort etter de har kommet frem til et svar, er det mye av dette de kan miste. I tillegg til dette er det viktig å ta seg tid til å se over at løsningen deres faktisk er korrekt. Det er mye feil som ikke lett oppdages i en slik prosess, spesielt hvis det er et større og sammensatt problem (Polya, 2004).

Pólyas (2004) heuristikk for problemløsning presenterer flere forskjellige typer strategier du kan bruke i møte med et problem du i utgangspunktet ikke har en metode for å løse. Dette er meget krevende for læreren å lære bort til elevene. Schoenfeld (2016) beskriver hvordan det krever god matematisk didaktisk kunnskap å kunne veilede og vite når du skal bryte inn. I tillegg krever det god matematisk dybdeforståelse å kunne forstå og forutsi hvordan elevenes tanker kan ha innvirkning på problemet. Læreren kan da ikke alltid vite svaret, som krever erfaring, selvinnsikt og selvtillit (Burkhardt, 1988, sitert i Schoenfeld, 2016). Ekte problemløsning er minst like krevende for læreren som for elevene, men belønningen er mye større når det blir ordentlig gjennomført (Schoenfeld, 2016).

Pólya var en pioner innen matematisk problemløsning og det er derfor ikke mulig å drøfte problemløsning uten å trekke ham frem. I tiden etter utgivelsen av *How to solve it* har imidlertid flere studier vist at det er vanskelig å gi empiriske bevis for at det å lære elever heuristiske strategier gir en økt problemløsningsevne (Schoenfeld, 2016). Dette skal jeg presentere nærmere i neste kapittel (2.1.2).

2.1.2 Matematisk tenkning

Når vi skal diskutere problemløsning er det essensielt å trekke inn matematisk tenkning. Matematisk tenkning er et sammensatt tema og defineres på ulike måter fra ulike ståsteder. Det er derimot en viss konsensus blant forskere at matematisk tenkning består av følgende fem elementer (Schoenfeld, 2016):

- Kunnskapsbasen
- Problemløsningsstrategier/heuristikk
- Monitorering og kontroll
- Oppfatning og følelser
- Praksis

Alle mennesker innehar en kunnskap om hvordan de oppfatter at matematikken fungerer. Schoenfeld (2016) beskriver at hvordan en person møter matematiske problem, gjør at vi kan få et glimt inn i denne personens kunnskapsbase. Dette er de matematiske forhåndskunnskapene og disse er viktige å ha kunnskap om hvis du skal tenke deg frem til hvordan personen kommer til å angripe problemet. Hvilke deler av problemet treffer den faglige kunnskapen tydeligst, og hvilke valg vil problemløseren da ta? I en analyse av en persons matematiske problemløsning, mener Schoenfeld (2016) det vil være essensielt å få et innblikk i kunnskapsbasen til problemløseren. Kunnskapsbasen består av kunnskapen de innehar og hvordan de utfører og implementerer denne informasjonen. I denne analysen vil det være relevant å undersøke hvilke verktøy problemløseren hadde for å se hvorfor valgene underveis i prosessen ble bestemt (Schoenfeld, 2016). Valgte de bort tenkemåter og strategier med vilje, manglet de kunnskap for å oppdage denne tankerekken eller ble det oversett. Kunnskapsbasen omfatter i tillegg hvilken informasjon man klarer å hente frem fra langtidsminnet til å kunne brukes i problemer. Schoenfeld (2016) mener at det vil være interessant å påpeke at kunnskapsbasen til en person ikke trenger å være

korrekt. Konsekvensen av dette vil være at problemløser tror den har løst et problem på riktig faglig bakgrunn, men gir et svar som er feil. Fra personens faglige ståsted, ser imidlertid svaret riktig ut (Schoenfeld, 2016).

I Pólyas (2004) verk finner vi bruken av begrepet heuristikk. Heuristikk forklares som strategier til å løse matematiske problem. Dette beskriver Schoenfeld (2014) som en av de fire essensielle delene til å løse matematiske problem. Schoenfeld (2016) forklarer heuristikk som generelle strategier eller fremgangsmåter som skal bidra til å forstå problemet eller gi fremgang i problemløsningsprosessen. Han viser til kritikk knyttet til effekten av å lære elever Pólyas problemløsningsstrategier i årene etter lanseringen av *How to solve it*. Spesielt trekker han frem at det å trene seg i spesifikke problemløsningsstrategier ikke nødvendigvis vil gjøre deg til en mer erfaren problemløser. I prosessen av å løse et problem vil problemløseren alltid ta i bruk en form for problemløsningsstrategier. Dette krever derimot ikke at strategiene er øvd på fra før (Schoenfeld, 2016). Begle (1979) gir oss et godt bilde av denne utfordringen: «In fact, there are enough indications that problem solving strategies are both problem- and student-specific often enough to suggest that finding one (or few) strategies which should be taught to all (or most) students are far too simplistic» (1979, s. 146).

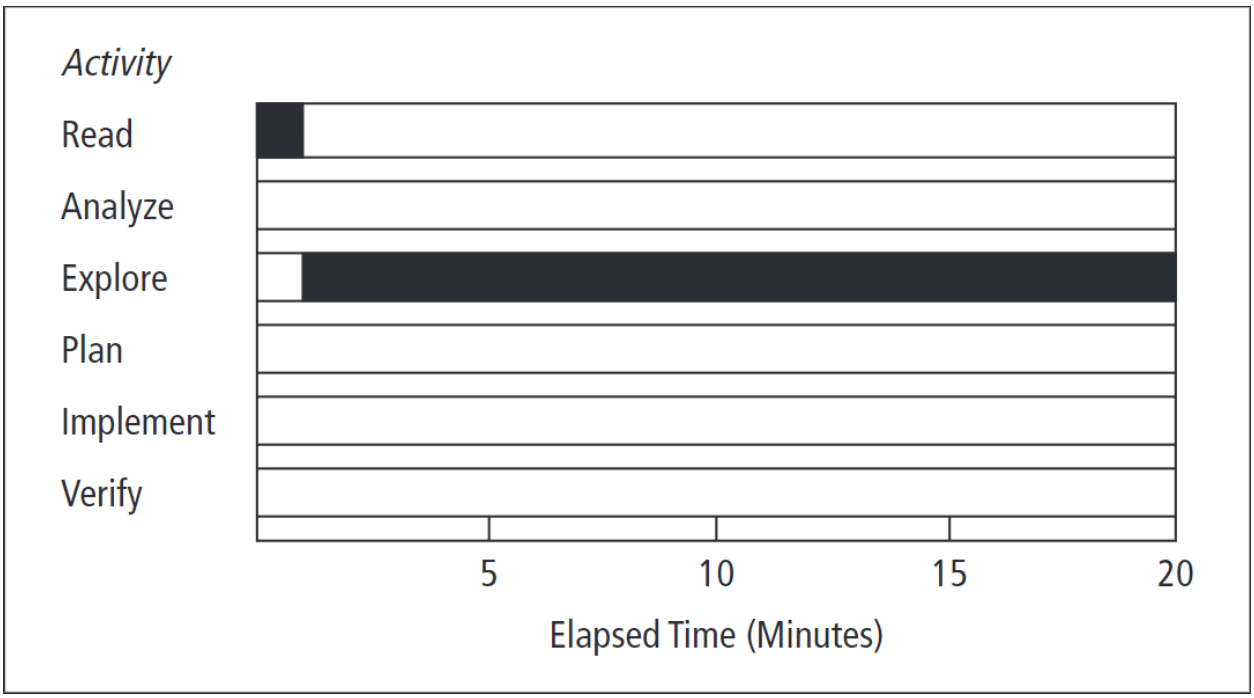
Schoenfeld (2016) trekker spesielt frem to studier som viser at problemløsningsstrategier elever ble undervist i, ikke var overførbare til andre emner. Et poeng som kommer tydelig frem hos Schoenfeld (2016) er at problemløsning handler i større grad om elevens tidligere erfaring og forhåndskunnskap og hvordan den bruker dette til å forstå problemet, heller enn at eleven kan den spesifikke problemløsningsstrategien som kreves. Problemene som lar seg løse av en person er derfor begrenset på bakgrunn av personens personlige problemløsningsstrategier (Schoenfeld, 2016).

Schoenfeld (2014) nevner flere kjente problemløsningsstrategier og utdyper to som ofte går igjen. Den første omhandler å gå ut fra at du allerede vet svaret på oppgaven når du gjør deg kjent med den, og derfra jobber bakover. Dersom du antar at du vet svaret, kan du få tilgang på informasjon som gir deg fremgang mot løsningen. Den andre handler om å låse alle variabler unntatt én og endre på denne variabelen for å se hvilket resultat endringen gir. Denne måten å problemløse på gir deg mulighet til å både få kjennskap om den spesifikke variabelen, og muligheten til å fokusere målrettet på et punkt av gangen. Slik problemløsning kan være nyttig

når du står overfor flere variabler og du ikke nødvendigvis har kunnskap om hva løsningen kan være (Schoenfeld, 2014).

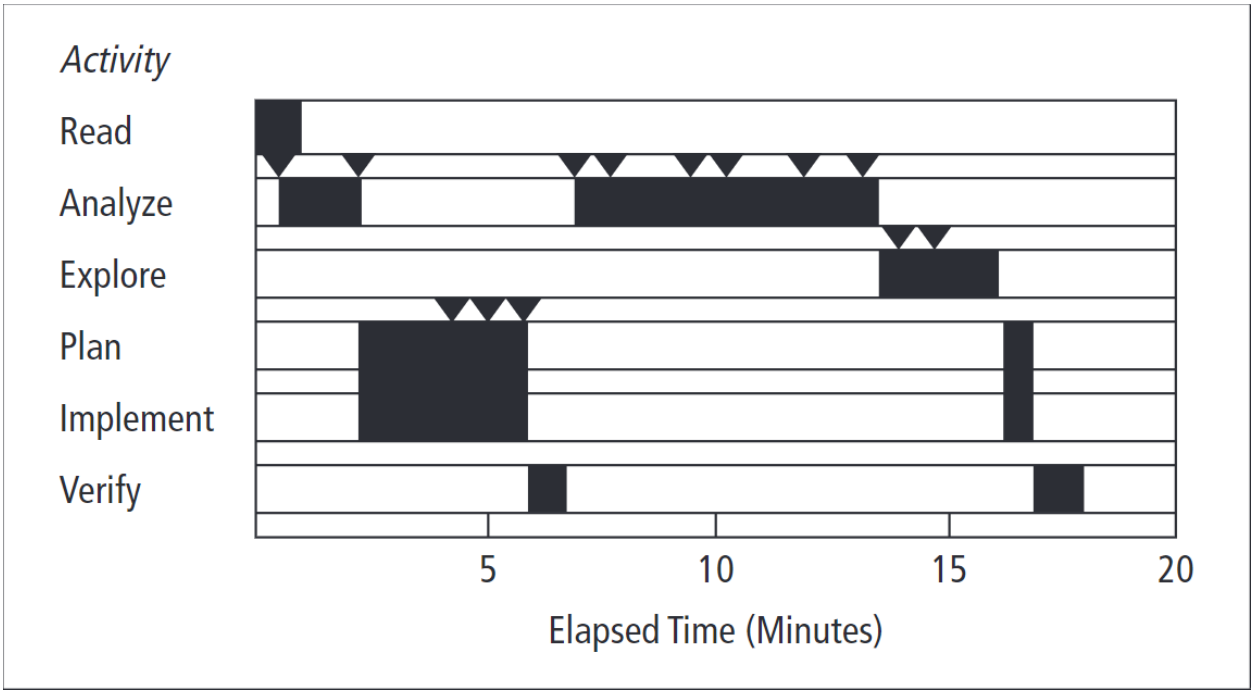
Når du arbeider med en oppgave og ser at oppgaven egentlig beveger seg i en retning du ikke forventet eller prosessen du har begynt på ikke gir et svar, kan det være lurt å kikke tilbake på hva oppgaven egentlig spør om. Dette er deler av tankene som ligger til grunn for begrepene metakognisjon og selvregulering (Schoenfeld, 2016). Metakognisjon er ifølge James P. Van Overschelde (2008) en studie av hjernens kapasitet til å kontrollere og monitorere seg selv, altså det å ha kjennskap til hva du har kunnskap om og hvordan du lærer. Selvregulering defineres som å ha kontroll over handlinger og tanker i møte med å komme frem til en løsning på et problem eller utfordring (Inzlicht, Werner, Briskin, & Roberts, 2021). Metakognitiv kunnskap kan være med på å hjelpe deg å ta gode valg i møte med problemer. Hvis du har kontroll over hva du kan og hvilke ferdigheter du besitter, kan du mye enklere møte problemer med beslutninger som stemmer overens med hva du kan klare (Flavell, 1979). Slike metakognitive ferdigheter lar seg ikke undervise direkte, men læres gjennom gjentatt utfordring og oppfølging av lærer i møte med problemer (Schoenfeld, 2016).

Schoenfeld (2016) sammenligner hvordan en erfaren matematiker og elever som ikke har noe problemløsningserfaring arbeider med matematiske problem. Dette gjør han for å vise hvor viktig det er å stille seg selv spørsmål om hvorfor du gjør det du gjør, slik at du kan ta stilling til om valgene du tar fører til en løsning. I figur 1 kan vi se hvordan elevene bruker veldig kort tid på å bestemme seg for hvordan de mener oppgaven bør løses og gjennomfører deretter denne planen. Ofte vil dette hurtige førstevalget vise seg å ikke være riktig. Selv om dette er sant, vil elever som ikke har noe spesiell erfaring innen problemløsning fortsette ned denne veien uten å vurdere om de burde endre strategi (Schoenfeld, 2016).



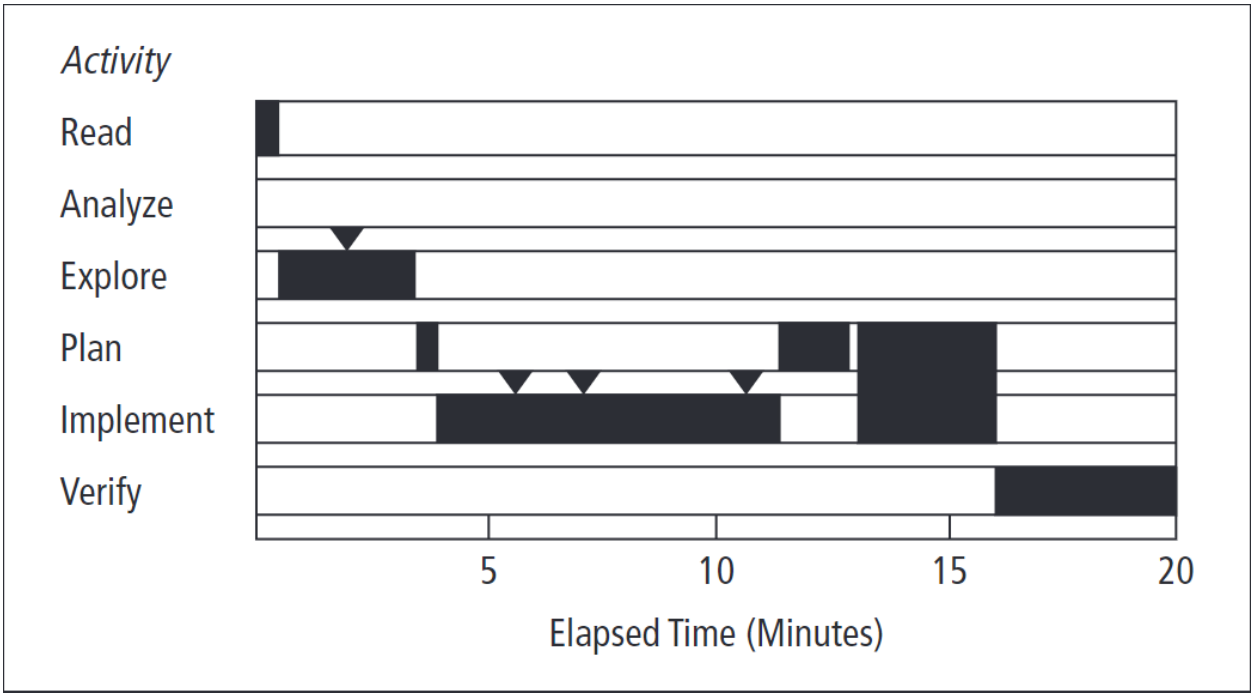
Figur 1: Tidslinje for en problemløsningsprosess for elever uten problemløsningserfaring – Fra «Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics (Reprint)» av Schoenfeld, A. H., 2016, *Journal of education*, 196, s. 24 (https://doi.org/10.1177/002205741619600202). Copyright 2016 ved Journal of education.

Figur 2 viser oss hvordan en erfaren matematiker arbeider med et problem. Her kan vi se et mye større fokus på å analysere problemet for å være helt sikker på hva som skal gjøres før hen planlegger og implementerer planen. Deretter går matematikeren tilbake og analyserer problemet om igjen, for å få et nytt blikk på problemet. De små trekantene indikerer kommentarer matematikeren har gitt underveis i prosessen, som for eksempel det å si høyt at hen er usikker på hvor hen skal starte. Det må spesifiseres at problemet som ble gitt er et problem sammensatt av to deler, derav initierte matematikeren to analyser.



Figur 2: Tidslinje for en problemløsningsprosess for en erfaren matematiker – Fra «Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics (Reprint)» av Schoenfeld, A. H., 2016, *Journal of education*, 196, s. 24 (https://doi.org/10.1177/002205741619600202). Copyright 2016 ved Journal of education.

Den siste figuren (figur 3) Schoenfeld (2016) presenterer er en beskrivelse av hvordan en gruppe elever arbeidet med et problem, etter at de har gjennomført ett av hans problemløsningskurs. I dette kurset gikk læreren rundt og veiledet elevene gjennom prosessen ved å stille gode spørsmål for å hjelpe dem med å tenke gjennom hvorfor de gjør det de gjør. De starter med å lese raskt gjennom oppgaveteksten før de bestemmer seg for en mulig løsning. Denne viser etter få minutter å være et blindspor, før de på ny prøver en annen løsning. Ettersom denne krevde en del komplisert utregning ble de værende lengre på denne delen før de fant ut at heller ikke denne var på rett vei. De stopper opp og finner ut at de må starte på nytt med blanke ark og ny plan, og finner da ganske fort en riktig vei ut av problemet. Denne figuren stiller seg i tydelig kontrast til figur 1 der elevene stort sett bruker tiden på å utforske problemet gjennom prøving og feiling. De bruker svært lite eller ingen tid på refleksjon over hva og hvorfor de gjør det de gjør. I det siste eksempelet finner vi derimot elever som etter å ha blitt veiledet av læreren med gode spørsmål, ender opp med å stille seg selv de samme spørsmålene og derfra viser en tydelig endring i selvregulering og monitorering (Schoenfeld, 2016).



Figur 3: Tidslinje for problemløsningsprosess for studenter etter problemløsningskurs – Fra «Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics (Reprint)» av Schoenfeld, A. H., 2016, *Journal of education*, 196, s. 24 (https://doi.org/10.1177/002205741619600202). Copyright 2016 ved Journal of education.

Her ser vi hvordan det å stille seg selv gode spørsmål underveis i en problemløsningsprosess er med på å få blikket opp og se hva det er du gjør og hvorfor. Slik kan du oppdage tidlig om du er på vei ned en retning som ikke leder til en løsning (Schoenfeld, 2016).

I Liljedahl, Santos-Trigo, Malaspina og Bruder (2016) presenterer Liljedahl, Perkins sine 4 typer problem som krever et «aha!-øyeblikk» for å løse: «wilderness of possibilities», «the clueless plateau», «narrow canyon of exploration», og «oasis of false promise». Hvis du ender opp i en «villmark av muligheter» så har du møtt et problem der du har veldig mange veier som virker tiltalende og derfor gjør det vanskelig å finne riktig vei til løsningen. Utveien vil da være å systematisk redusere antallet muligheter for å prøve å smalne inn antallet mulig veier (Liljedahl et al., 2016). I motsetning til denne beskriver Liljedahl (2016) et «uvitende platå» som et problem der du har få eller ingen hint om hvordan problemet kan løses. Veien videre vil da være å lete aktivt etter hint i problemet, gjerne i oppgavens tekst og ordlyd. En «smal utforskende dal» blir beskrevet som et problem der rammene har blitt så stramme at løsningen blir umulig å finne. Her vil det være lurt å prøve å se på problemet med nye øyne og se om rammene oppgaven har er selvpåførte eller en del av problemet (Liljedahl et al., 2016). Til beskriver han problemer av

typen som en «oase av falske løfter» være et problem som tidlig gir deg en vei å gå som nesten kommer frem til en løsning og derfor frister deg til å bli værende på denne veien. Dette kan føre til mye tids- og energibruk på en løsning som ikke treffer målet, men klarer du å se bort fra denne fristende oasen kan det hjelpe deg med å finne en vei til løsningen. Hvis du kjenner til disse 4 typene problem og hvordan du finner veien ut, vil kjennskap til type problem være en strategi for å nærme seg en løsning (Liljedahl et al., 2016).

Schoenfeld (2016) deler oppfatning i tre deler: elevens-, lærerens- og samfunnsmessig oppfatning. Ettersom det er elevenes oppfatning og samtaler som er fokuset for denne studien, velger jeg å legge hovedfokus på deres oppfatning. Elever som har gjennomgått vanlig skolegang har til sammen gjennom alle år måtte arbeide seg gjennom tusenvis av matematikkoppgaver som ikke krever mer enn et par minutter å løse (Schoenfeld, 2016). Dette er oppgaver der de på forhånd har fått en oppskrift eller algoritme som de nå enten skal trene på eller vise at de klarer å utføre. Schoenfeld (2016) forklarer hvordan elevene kan tenke; hvis jeg forstår temaet vi jobber med, kan jeg klare oppgavene. Derimot hvis jeg ikke kommer meg gjennom oppgavene innenfor en rimelig tidsramme, forstår jeg ikke temaet. En elev med et slikt tankesett mener som møter problemløsningsoppgaver mener Schoenfeld (2016) vil miste en mulighet ved å gi opp tidlig i prosessen, selv om problemet kunne blitt løst hadde eleven brukt litt mer tid. Han poengtere at uten å få trene på og bli utsatt for problemløsning, vil elever ikke få kvittet seg med tanken om at alle oppgaver løses med samme typer strategier, og deres bilde av hvordan gjøre matematikk blir dermed fattigere. Eleven vil gjerne tenke at problemet er en oppgave som eleven selv ikke mestrer, basert på egen oppfatning (Schoenfeld, 2016). En slik oppfatning hos elever bidrar til å redusere mengden metakognitive ferdigheter som brukes under problemløsningen, som da gjør at de mister muligheten til å kunne re-evaluere problemet og angripe det fra en ny vinkel.

Schoenfeld (2016) påpeker at læreres oppfatninger er viktige for elevers syn på matematikk og deres matematiske utvikling. Hvordan lærere oppfatter matematikken og hvordan dele denne videre legger til grunn for hvordan læringsmiljøet blir knyttet til matematikken. I tillegg er synet på matematikk preget av samfunnet rundt læringsarenaen. Samfunnsmessige oppfatninger gjør at elever og lærere fra forskjellige land har forskjellig syn på hva som kreves for å forbedres i matematikk (Schoenfeld, 2016; Stigler & Hiebert, 2009).

Shulman (1986) introduserte begrepet «pedagogical content knowledge» som på norsk kan oversettes til pedagogisk fagkunnskap, og beskriver lærerens kunnskap om undervisning av pensum. Denne kunnskapen legger til grunn for det som skjer i undervisningssituasjonene og hvilke rammer som legges til rette for elevene (Shulman, 1986). Elevers oppfatning baseres ganske tydelig på hvilke matematiske praksiser de har erfart, og endres da basert på det matematiske fellesskapet som skjer i klasserommet (Schoenfeld, 2016). Enkelte praksiser viser seg å være mer positive for læring enn andre, og Schoenfeld (2016) presenterer noen studier som viser oss slike praksiser. Som fellesnevner i disse studiene har læreren ikke en autoritær rolle, men brukes som en støttespiller og veileder i prosessene. Elevene angriper oppgaver sammen og blir utfordret til å stille spørsmål og samtale matematisk sammen med medelever og lærer. De blir utfordret i å kunne forsvare egne ideer samt utfordre andres tankemønster (Schoenfeld, 2016). Tanken bak slike praksiser er å få matematikken i skolen til å ligne mer på slik matematikk gjøres av matematikere (Lampert, 1990; Schoenfeld, 2016).

Fra Schoenfelds (2016) fem punkter for matematisk tenkning ser vi hvor sammensatt en elevs matematiske forutsetninger og tankeverden er. Elevens forhåndskunnskaper legger til rette for det faglige utgangspunktet eleven har for læring. Hvilke problemløsningsstrategier eleven har i repertoaret sitt har mye mindre å si for elevens evne til problemløsning enn gjentatt eksponering for problemer der læreren stiller som veileder i prosessen. Hvilke oppfatninger deltakerne i undervisningen besitter, legger grunnlaget for hvor effektiv den matematiske praksisen blir. Den matematiske praksisen beskriver hvordan elevene får store fordeler av å delta i en undervisning som fostrer matematisk undring som fellesskap (Schoenfeld, 2016).

Schoenfeld (2016) argumenterer for at matematikken er en meningsfull aktivitet som er sosial i bunn. Elevenes oppdagelser gjennom faget er hvordan de lærer seg å bruke og sanse matematikken. Derfor bør de matematiske opplevelsene elevene gjør være koblet opp mot denne meningsfulle aktiviteten, slik at elevene får bruk for matematikken de møter. En mengde litteratur ser på matematikk som en i hovedsak sosial og konstruktiv aktivitet (Brown, Collins, & Duguid, 1989; Lester Jr & Cai, 2016; Resnick, 1988; Schoenfeld, 2016). Resnick (1988) legger det frem slik:

> [T]he reconceptualization of thinking and learning that is emerging from the body of recent work on the nature of cognition suggests that becoming a good mathematical

> problem solver—becoming a good thinker in any domain—may be as much a matter of acquiring the habits and dispositions of interpretation and sense-making as of acquiring any particular set of skills, strategies, or knowledge. If this is so, we may do well to conceive of mathematics education less as an instructional process (in the traditional sense of teaching specific, well-defined skills or items of knowledge), than as a socialization process. (1988, s. 58)

2.2 ALGORITMISK TENKNING

Begrepet algoritmisk tenkning omhandler å kunne planlegge og stegvis løse et problem, og vurdere om det skal løses ved hjelp av teknologi (Utdanningsdirektoratet, 2019). I LK20 finner vi under kjerneelementet «Utforsking og problemløsning» en bredere beskrivelse av algoritmisk tenkning:

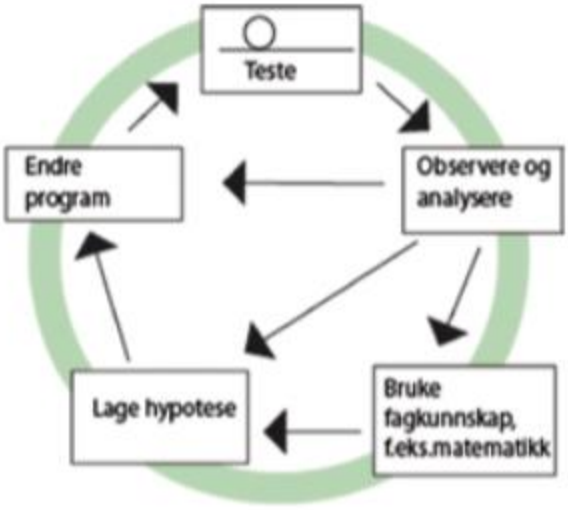
> Algoritmisk tenkning er viktig i prosessen med å utvikle strategier og framgangsmåter for å løse problemer og innebærer å bryte ned et problem i delproblemer som kan løses systematisk. Videre innebærer det å vurdere om delproblemene best kan løses med eller uten digitale verktøy. (Kunnskapsdepartementet, 2019)

Begrepet *Computational thinking* (heretter forkortet til CT) er blitt mer fremtredende i skolen i nyere tid. Originalt var dette et begrep som var avgrenset til informatikkens verden, før Jeanette Wing (2006) etterspurte ønske om å få begrepet inn i grunnskolen. I sin artikkel fra 2006 kommer hun med følgende utsagn om begrepet: «Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability» (Wing, 2006, s. 33). CT er det LK20 kaller algoritmisk tenkning. Algoritmisk tenkning er et paraplybegrep med en bred definisjon. Kort forklart beskriver algoritmisk tenkning hvordan angripe et problem på en mest mulig systematisk måte og designe en løsning som kan løses av mennesker, datamaskiner eller en kombinasjon (Bocconi et al., 2018; Gjøvik & Torkildsen, 2019). Videre i dette kapittelet vil jeg gå dypere inn i norsk og internasjonal forståelse av begrepet algoritmisk tenkning, samt presentere en modell for feilretting i programmeringsprosessen.

Gjøvik og Torkildsen (2019) oversetter og henviser til Bocconi et al. (2018) sin beskrivelse av CT med fem forskjellige begrep: *abstraksjon*, *algoritmebehandling*, *generalisering*,

automatisering og *dekomponering*. I begrepet *abstraksjon* ligger det å kunne luke ut unødvendig informasjon og finne kjernen i oppgaven. *Algoritmebehandling* beskriver det å kunne følge en trinnvis plan i tillegg til å kunne forklare hva du gjør (Gjøvik & Torkildsen, 2019). Ordet algoritmebehandling har Gjøvik og Torkildsen (2019) hentet fra det engelske begrepet «algorithmic thinking», som igjen er en underkategori av CT. Det er viktig å merke seg at «algorithmic thinking» ikke er det samme som algoritmisk tenkning, og de må derfor ikke forveksles. Det å kunne se sammenhenger, lage regler og kjenne igjen mønstre beskriver de under begrepet *generalisering.* Videre forklarer de at reduksjon av mengden menneskelig tilførsel ved programmering av arbeidet ligger i begrepet *automatisering. Dekomponering* mener Gjøvik og Torkildsen (2019) her at betyr å kunne ta fra hverandre problemet og se på hver del for seg selv.

Stenseth, Kaufmann og Forsström (2019) beskriver en del av programmeringsprosessen de kaller *feilretting* eller *korreksjon*. Ordet feilretting er nokså selvforklarende, men går ut på å rette opp feil i koden slik at koden blir riktig. Helt teknisk er feilkoder veldig enkle å rette opp i, men som de fleste med noe erfaring innen programmering sannsynligvis har opplevd, kan både små og store feil være vanskelige å finne. I tillegg til feilkoder omhandler denne modellen at programmet ikke gir deg det utfallet du ønsker, og da må finne ut av hvordan du kommer frem til riktig resultat. Denne utfordringen kan bidra til at programmering fremstår vanskeligere og mindre fristende enn det potensielt kan være. Dette kan vi imidlertid vinkle til å bli en viktig del av problemløsningsprosessen (Stenseth et al., 2019). Stenseth et al. (2019) illustrerer ved hjelp av figuren representert i figur 4, hvordan arbeidet med programmering foregår i en syklus.



Figur 4: 3 sykluser for feilsøking – av Stenseth et al., 2019, *Tangenten–tidsskrift for matematikkundervisning*, 30, s. 8 (http://tangenten.no/wp-content/uploads/2021/12/tangenten-2-2019-Stenseth-et-al.pdf). Copyright 2019 ved Caspar Forlag AS

Fra denne figuren kan vi se 3 forskjellige sykluser, hvor den første syklusen består testing, observasjon og analyse, og endring av program. Denne kan vi kalle en prøve-og-feile-metode. I tillegg til prøving og feiling består den neste syklusen av forming av hypoteser, som Stenseth et al. (2019) kaller «den kodefokuserte» syklusen. Dette krever at brukeren forstår hva de forskjellige delene av programmet gjør. Til slutt ser vi en syklus som tar i bruk fagkunnskap i tillegg til testing, analysering, endring av program og forming av hypotese. Her forstår brukeren både hvordan programmet fungerer og hvilke faglige kunnskaper som ligger bak, slik at hypotesen får en faglig god begrunnelse (Stenseth et al., 2019). Denne måten beskriver i større grad hvordan programmeringsprosessen til en programmerer kan se ut. De fem begrepene til Bocconi et al. (2018) beskriver derimot de mentale prosessene som skjer.

2.2.1 Programmering i skolen

«Programmering er ikke bare for spesielt interesserte. Det er for alle. Det betyr ikke at alle skal bli programmerere, men vi mener at alle trenger en viss forståelse av de grunnleggende byggesteinene i vårt digitale samfunn» (Haraldsrud et al., 2020, s. 15).

Programmering defineres ifølge Stenseth et al. (2019) som «prosessen knyttet til utvikling og implementering av instruksjoner for dataprogrammer slik at datamaskinen kan utføre spesifikke oppgaver, løse problemer og støtte menneskelige interaksjoner» (s. 7). Kunnskap om

programmeringsspråk, god dybde innen fag relatert til algoritmene de utvikler, samt evnen til logisk tenkning og analyse er egenskaper som kreves for å kunne bli en god programmerer (Stenseth et al., 2019). Videre vil evnen til å kunne løse og forstå problemer ved å verifisere algoritmiske krav og vurdere korrekthet og implementering (ofte referert til som koding) av algoritmen i et bestemt programmeringsspråk være essensielt (Stenseth et al., 2019). Grover og Pea (2013) finner at disse prosessene ofte knyttes til matematisk tenkning og algoritmisk tenkning. En NOU fra 2020 presiserer hvordan programmering knyttes til algoritmisk tenkning, og hvordan å lære algoritmisk tenkning kan bidra til matematisk tenkning (NOU 2020:2). Derfor er disse ferdighetene blitt viktige inn mot det digitale samfunnet vi lever i, og fremtiden vi står overfor.

De siste årene har flere europeiske land i økende grad innført programmering som en del av andre fag (Balanskat & Engelhardt, 2014). I 2015 lanserte regjeringen en NOU ved navn «Fremtidens skole» der de presiserer hvor viktig digitale verktøy kommer til å være inn i elevenes egen fremtid. Videre påpeker de at elevene derfor må få muligheten til å lære fagkunnskap sammenflettet med digitale verktøy (NOU 2015:8). Norge innførte i år 2020 LK20 og med den ble det innført programmering som del av flere fag, men med hovedbestanddel i matematikkfaget. Kompetansemålene «*utforske hvordan algoritmer kan skapes, testes og forbedres ved hjelp av programmering»* og *«utforske matematiske egenskaper og sammenhenger ved å bruke programmering»* er noen eksempler på kompetansemål på henholdsvis 8.- og 10.trinn i matematikk (Kunnskapsdepartementet, 2019). I LK20 står dybdelæring sentralt; at elevene skal sitte igjen med en større dybdekunnskap og forståelse som blir værende. Programmering har muligheten til å bidra positivt inn i dybdelæring ved å gi elevene stor mulighet til utforskning samt mulighet for tverrfaglighet (Haraldsrud et al., 2020).

Stenseth et al. (2019) velger å legge fokuset sitt på hvordan programmering og matematikk kan forsterke hverandre i skolen. Med et ønske om å se om de kan finne problemløsningsmetodikk i programmeringen som støtter matematikken. De presenterer modellen for feilretting presentert i figur 4, og bruker denne til å blant annet beskrive lærerens rolle i en programmeringsøkt. Læreren bør ifølge Stenseth et al. (2019) bruke mye tid på å planlegge godt slik at elevene arbeider med gode oppgaver. Underveis i prosessen bør læreren fungere som en veileder eller guide slik at elevene selv får mulighet til å utforske og løse problemer de møter. Denne måten å

veilede reflekteres i beskrivelsen av lærerens rolle i en problemløsningsprosess hos Schoenfeld (2016).

Psycharis og Kallia (2017) gjorde en undersøkelse for å se om programmering bidrar positivt til elevers problemløsning i matematikk. Resultatene deres viser for liten effekt til å kunne konstatere at det forbedrer elevenes problemløsning, men argumenterer at en del av grunnen til dette kan være elevenes manglende trening i algoritmisk tenkning. Algoritmisk tenkning kreves for å kunne lære programmering, slik at du kan klare å beskrive og finne en løsning til problemet (Psycharis & Kallia, 2017). For mange elever hopper over stegene som beskrives under algoritmisk tenkning og problemløsning og begynner på problemet før de har forstått hva de skal gjøre og lagt en plan for gjennomførelsen (Kazimoglu, Kiernan, Bacon, & MacKinnon, 2012).

2.2.2 Programmeringsspråk

Programmering deles inn i to deler, enten tekstbasert- eller blokkbasert programmering. Blokkbasert programmering er den mest visuelle programmeringsformen og tar i bruk blokker som fungerer som en form for puslespillbrikker (Haraldsrud et al., 2020). For å lage et program setter du sammen blokkene i den rekkefølgen du ønsker og endrer verdier inne i blokkene om nødvendig. Det mest utbredte blokkbaserte språket som brukes i dag heter Scratch, og ble utviklet ved Massachusetts Institute of Technology (MIT) og lansert i 2007 (Resnick, Maloney, Monroy-Hernández, Rusk, Eastmond, Brennan, Millner, Rosenbaum, Silver, & Silverman, 2009). Denne studien velger å fokusere på tekstbasert programmeringsspråk.

Tekstbasert programmering er det man gjerne ser for seg at en som jobber som programmerer arbeider med. I all enkelhet er tekstbasert programmering satt sammen av kodelinjer med ren tekst istedenfor blokker. Det første tekstbaserte programmeringsspråket til den moderne datamaskinen ble utviklet i 1949 og er det vi i dag kaller Assembly. Det gikk ut på å flytte 0-ere og 1-ere mellom porter i prosessoren til datamaskinen, og er det vi kaller «low-level»-programmering (Britannica, 2023). På slutten av 1960-tallet utviklet flere forskere ved MIT et programmeringsspråk som skulle rette seg inn mot undervisningen, dette kalte de LOGO (Hemmendinger, 2019). LOGO var utviklet for å være et språk for læring, og brukte «turtle geometry» eller skilpadde-geometri på norsk. Dette var en måte for brukeren å få visuell tilbakemelding fra koden. Du kunne skrive inn hvor du ville skilpadden skulle gå, så tegnet den det du hadde kodet inn. For eksempel kunne du skrive «forward 100», og skilpadden gikk 100

skritt fremover mens den tegnet en linje fra startpunkt til sluttpunkt (Hemmendinger, 2019; Solomon, Harvey, Kahn, Lieberman, Miller, Minsky, Papert, & Silverman, 2020).

Videre ble det utviklet mer og mer avanserte og programmeringsspråk som stort sett bygget på sine forgjengere, slik at vi i dag har programmeringsspråk som for eksempel Java, C og det språket jeg i denne studien har valgt å bruke: Python (Henderson, 2009). Python er det vi kaller et «high-level»-programmeringsspråk. Dette vil si at det du skriver i kodefeltet er mye mer likt et vanlig språk, som datamaskinen oversetter slik at den kan bruke informasjonen (Henderson, 2009). Python ble spesifikt laget for å ha en mest mulig oversiktlig, ryddig og lesbar syntaks. Dette er grunnen til at Python ofte er foretrukket av nybegynnere i feltet (Henderson, 2009; Van Rossum, 2007). Rekkefølgen på kodelinjene er ikke uvesentlig, men beskriver i hvilken rekkefølge koden skal kjøres, fra topp til bunn (Pratt, 1973). På grunn av at syntaksen til Python bruker færre symboler for å indikere hierarkiet i koden, bruker språket innrykk i koden for å spesifisere dette. Innrykk av kodelinjer indikerer at de tilhører forrige kodelinje ovenfor, som er utrykket (Grandell, Peltomaki, Back, & Salakoski, 2006).

Ettersom det er ganske mye du kan gjøre ved hjelp av programmering, og dette krever ganske mange forskjellige kommandoer, er det slik at språkene har delt inn kommandoer i forskjellige «bibliotek» (Henderson, 2009). Dette gjør at du kun trenger å hente de kommandoene du har behov for, i det spesifikke programmet du skal lage. Du kan for eksempel importere et matematikk-bibliotek dersom du ønsker å bruke matematiske kommandoer som kvadratrot og trigonometriske uttrykk. Turtle-biblioteket importerer kommandoer som gjør det mulig å bruke et grafisk grensesnitt, et pop-up vindu som visualiserer koden, i Python (Henderson, 2009). Dette gir oss muligheten til å bruke samme type kode som ble brukt i LOGO, men ved å programmere i Python i stedet.

3 METODE

I dette kapittelet vil jeg redegjøre for mine metodiske valg, samt se nærmere på hvordan jeg samlet inn og bearbeidet datamaterialet. Videre presenterer jeg utvalget av oppgaver og observasjonsobjekter, og fremlegger en forklaring på bakgrunn for valgene mine. Deretter presenterer jeg hvilke etiske vurderinger som har blitt tatt gjennom prosessen, samt greier ut om denne studiens troverdighet.

3.1 VALG AV METODE

Metode omhandler hvilke fremgangsmåter man benytter under innhenting av ny informasjon eller etterprøving av eksisterende informasjon (Dalland, 2017). Under denne studien har jeg hovedsakelig vurdert kvalitative opp mot kvantitative metoder. Kvantitative metoder gir målbar data som formidles i form av tall og innhentes som oftest gjennom spørreundersøkelser (Postholm & Jacobsen, 2018). Kvalitative metoder er ute etter å fange opp opplevelser og mening hos objektene for studien, og er derfor vanskelige å måle slik kvantitativ data måles. Slike metoder tar sikte på å fange opp mest mulig data fra få undersøkelsesobjekter og å finne sammenheng mellom dybde og helhet (Dalland, 2017). Harboe (2006) forklarer at de kvantitative metodene gir bedre overblikk, mens de kvalitative metodene brukes til å forstå dybden.

Med denne oppgaven ønsker jeg å finne ut av hvilke problemløsningsprosesser som kommer til uttrykk når elevene blir introdusert for tekstbasert programmering. Ettersom fokuset i oppgaven er på tanker og prosesser hos den enkelte elev, har jeg i denne oppgaven valgt å bruke en kvalitativ metodisk tilnærming, til fordel for en kvantitativ metodisk tilnærming. Ifølge Dalland (2017) er det flere elementer som har betydning for valg av metode. Valget av metode avhenger av hvilken metode som belyser problemstillingen best mulig. Ønsket med denne studien er å øke forståelse for hvilke prosesser elever står i, i møte med ukjent programmeringsoppgaver i matematikk. På denne måten legger problemstillingen tydelige føringer for hvilken metode for innsamling av data som er mest hensiktsmessig for denne studien (Dalland, 2017).

3.2 VALG AV FORSKNINGSDESIGN

Forskningsdesign beskrives som det teoretiske rammeverket som ligger til grunn for hvordan datamaterialet skal samles inn (Lerdal, 2009). Jeg har gjennomført en observasjonsstudie hvor jeg ønsket å observere hvilke prosesser elever står i, i møte med programmeringsoppgaver i matematikk. Mine observasjoner gjennom lyd- og skjermopptak danner grunnlag for mine funn. På grunn av oppgavens omfang og tidsbegrensninger vil funnenes validitet bli svekket av manglende mengde datamateriale. Her vil jeg redegjøre for hva en casestudie er og hvorfor jeg valgte dette forskningsdesignet.

3.2.1 Casestudie

I denne oppgaven har jeg valgt casestudie som bakgrunn for datainnsamlingen. Ordet «case» kommer fra det latinske ordet «kasus» og betyr tilfelle (Grønmo, 2020, 14. mai). En casestudie er definert som en fokusert studie på en eller flere personer eller tilfeller med tanke på å generalisere funn (Heale & Twycross, 2018). Den studerer en spesifikk situasjon i en naturlig setting, som i denne oppgaven var gruppeoppgaver sammen med andre elever i et grupperom (Harling, 2012) . En casestudie er som oftest knyttet til kvalitativ forskning, der det spesifikke tilfellet er i fokus og settingen datainnsamlingen skjer i, fungerer som bakgrunn for tilfellet (Bryman, 2016). Valget mitt av å observere grupper på 2-4 elever som gjennomfører programmeringsoppgaver, gjør at denne studien blir en casestudie der datamaterialet kommer fra observasjoner gjennom lyd- og skjermopptak.

Innsamling, transkribering og analyse av datamaterialet er en tidkrevende prosess som gjør at datamengden i denne studien er relativt liten. Det er derfor viktig å stille spørsmål ved hvor representative datamaterialet er for resten av Norges elever. Mine funn viser kun hvordan de spesifikke elevene tenker og vil ikke tilsi hvordan andre elever tenker uten sammenlignbar data. For at mine resultater skal bli mer pålitelige bør de derfor sammenlignes med andre skoler og elever. Deretter kan man si hvorvidt resultatene gjelder for en større populasjon (Kvale & Brinkmann, 2015). Som et generelt grunnlag kan man sjeldent generalisere resultatene fra én casestudie, men det betyr ikke at studien ikke kan ha akademisk verdi (Flyvbjerg, 2006). Ved bruk av en casestudie kan datamaterialet brukes av en annen forsker som kan validere eller bruke det som motsetning til annen data, for å se om resultatene er pålitelige (Bryman, 2016).

3.3 DATAINNSAMLING

Innsamlingen av datamaterialene til denne studien har foregått på én skole over en tid på 2 uker. Før innsamlingen ble gjennomført var det flere ting jeg måtte forberede. Jeg måtte i første omgang sette meg inn i det teoretiske rammeverket for denne studien, slik at jeg i neste omgang kunne utarbeide og finne programmeringsoppgaver som la til grunn muligheten for å få mest relevant data. Deretter måtte jeg innhente informanter og samarbeide med en lærer for å sette opp grupper som tilrettela for at elevene skulle ønske å dele mest mulig av det de tenkte.

3.3.1 Utvalg i studien

I perioden før innsamling av datamaterialet var det viktig å ta noen valg. Valgene som tidlig måtte tas var hvilken skole og hvilket trinn elevene jeg ønsket skulle gå på. Etter dette valget var tatt var det viktig å finne en lærer å samarbeide med, som kunne bistå til utdeling og innsamling av samtykkeerklæringer samt utvelgelse av grupper. I tillegg til dette var det viktig å velge og designe gode problemløsningsoppgaver som ga meg mest mulig verdifull data.

3.3.1.1 Valg av skole og elever

Interessen min for programmering og matematikk som to separate temaer og hvordan de samarbeider er i hovedsak bakgrunnen for valget av problemstilling. Jeg hadde tidligere kjennskap til matematikklæreren til elevene i studien og visste at hen hadde interesse for programmering. Det var derfor interessant å spørre hen om å være deltakende i denne studien. Læreren svarte ja, og har hjulpet meg mye med å få flest mulig elever til å delta i studien min, og ikke minst etterspørre samtykkeerklæringer.

Under innsamlingen av datamaterialet gikk informantene mine i denne studien på 10.trinn, ved en norsk ungdomsskole. Enkelte av elevene hadde noe erfaring med programmering fra før, mens andre hadde mer eller mindre ingen erfaring. Fra utdeling av samtykkeerklæringene tok det 3-4 uker før jeg fikk tilbakemelding fra nok elever. Etter en god del oppmuntring fra læreren deres fikk jeg til slutt inn til sammen ni elever som ønsket å delta i studien. Gjennom samarbeid med læreren ble elevene delt inn i grupper på fire, tre og to elever, selv om jeg ideelt sett skulle ønske det var tre elever på hver gruppe. Grunnen bak dette ønsket var å få flere synsvinkler, men fortsatt skape rom for at alle hadde mulighet til å delta. Inndelingen ble gjort slik fordi elevene var fra to forskjellige klasser og dermed hadde to forskjellige timeplaner. Bakgrunnen for

sammensetningen av gruppene, var å sette sammen elever som var trygge nok på hverandre til at de delte mest mulig av tankene sine.

3.3.1.2 Valg av oppgaver

For å få høyest mulig kvalitet på datamaterialet, ble valget av et oppgavesett en tidkrevende prosess. Oppgavene elevene fikk utdelt måtte bruke enkel nok programmering for uerfarne programmerere, i tillegg til at de både hadde et matematisk nivå som var relevant for 10.klasse elever og være tydelige problemløsningsoppgaver. Valget mitt stod mellom å gi elevene blokkprogrammering som jeg vet de har hatt i undervisningen fra tidligere år på skolen, eller tekstbasert programmering som mer eller mindre ingen av elevene har erfart. Planen med oppgavene var å se hvordan elevene tenker når de møter nye utfordringer de ikke har en konkret plan om hvordan de skal løse. Det var i tillegg viktig for meg at oppgavene ble laget på en måte slik at de fungerte som problemløsningsoppgaver. Derimot endte valget mitt opp på å bruke elevenes mangel på erfaring med tekstbasert programmering som kjernen til problemene i oppgavene. Dette gjorde at oppgavene i seg selv ikke endte opp med å være åpne problemløsningsoppgaver. Valg av programmeringsspråk havnet derfor på Python.

Python er et tekstbasert programmeringsspråk som er designet for at koden skal være mest mulig lesbar (Nosrati, 2011). Dette i tillegg til min egen erfaring med språket gjorde at da valget havnet på et tekstbasert programmeringsspråk, ble Python et naturlig valg. Takhøyden på oppgaver du kan designe eller finne er mye høyere enn i et blokkbasert programmeringsspråk som Scratch, derimot er inngangsterskelen deretter (Bueie, 2019; Mladenović, Krpan, & Mladenović, 2016). Derfor var det veldig viktig å finne oppgaver som ikke krevde noe tidligere programmeringskunnskap.

I utviklingen av oppgavene som jeg til slutt endte opp med, brukte jeg boka “Kaares kokebok i programmering” av Jørgensen og Dahl (2021). Boken er designet for elever og lærere ved videregående skole og ungdomsskole og skal fungere som en grunnleggende opplæring i programmeringsspråket Python (Jørgensen & Dahl, 2021). Boka er lagt opp til å stegvis lære deg Python programmering og bruker mye tid på turtle-biblioteket til Python. Dette biblioteket aktiverer bruken av et grafisk grensesnitt til å tegne linjer ved hjelp av en markør, og en skilpadde er ofte brukt som et symbol. Du kan for eksempel fortelle programmet *forward(100)* og “skilpadden” beveger seg da 100 skritt fremover. Ettersom dette gir programmet du skriver en

direkte grafisk feedback, gir det muligheten til å lete etter sammenheng mellom kodelinjer og bevegelsene til "skilpadden". Det grafiske grensesnittet arbeider i et koordinatsystem og du kan ved enhver tid hentet ut informasjon om hvilke koordinater markøren er på. Valget havnet da på tre geometri-oppgaver som elevene skulle løse ved hjelp av turtle-biblioteket i Python. Oppgavene er utvalgt og designet slik at elevene først skal skrive av koden på papiret inn i programmeringsprogrammet og deretter kjøre koden for å se hva som skjer. Først etter dette skal de løse en oppgave knyttet til programmet de nettopp har skrevet. Oppgave 1 og 3 er hentet fra «Kaares kokebok i programmering» (Jørgensen & Dahl, 2021, s. 88-90) og oppgave 2 er delvis hentet fra samme bok, men mest fra eget design. Jeg vil presentere oppgavene nærmere i neste kapittel (3.3.1.2.1).

3.3.1.2.1 Presentasjon av oppgavesett

I dette kapittelet vil jeg presentere alle tre oppgavene i oppgavesettet med en kort forklaring av relevante kommandoer. En full oversikt over oppgavesettet ligger i vedlegg D. For en mer oversiktlig forklaring av kommandoer, se kapittel 3.3.1.2.2. Oppgave 1 i oppgavesettet (figur 5) presenterer fire forskjellige kodelinjer for elevene. Kodelinjen *from turtle import* * sin funksjon er å importere turtle-biblioteket. Hvis dette ikke gjøres i starten av programmet vil ingen av kommandoene som er knyttet til dette biblioteket være definert, og du vil få en feilkode hvis du prøver å kjøre programmet. Alle kommandoene i oppgavesettet bortsett fra *for i in range( )* kommer fra dette biblioteket. *Setx( )* og *sety( )* er kommandoer som gir markøren nye koordinater og *hideturtle( )* gjøre markøren usynlig. Målet med denne oppgaven er å bli kjent med disse fire kommandoene, samt relevansen av rekkefølgen i en kode. Rekkefølgen på utførelsen av kodelinjene starter fra toppen og beveger seg nedover.

Oppgave 1

```
from turtle import *

setx(45)
sety(100)
setx(-45)
sety(100)
hideturtle()
```

Skriv av koden ovenfor. Plasser kodelinjene i riktig rekkefølge slik at resultatet blir bokstaven T.

Figur 5: Oppgave 1 i oppgavesettet

Oppgave 2 kan vi se av figur 6 og introduserer elevene for en for-løkke. En for-løkke kjører koden som er innrykket under kodelinjen *for i in range( )*: et antall ganger tallverdien som er inne i parentesen til *range*. Kommandoen *forward( )* får markøren til å bevege seg et gitt antall skritt i den retningen markøren peker, der standard retning er i positiv x-retning. Retningen bestemmes i form av en vinkel, denne defineres av kommandoen *left( )*. Hvis du skriver inn tallverdien 90 vil markøren nå peke i positiv y-retning. For-løkken i oppgave 2 vil tegne en rettvinklet trekant, og spør elevene om de kan endre tallverdiene slik at de lager de tre neste mangekantene.

Oppgave 2

```
from turtle import *

speed(2)
shape("turtle")

for i in range(3):
    forward(100)
    left(120)
```

Skriv av koden ovenfor. Kjør koden og se hvilke figur den lager.

Endre tallene i for-løkken slik at koden lager en:

1. *Firkant*
2. *Femkant*
3. *Sekskant*

Figur 6: Oppgave 2 i oppgavesettet

De fleste kommandoene i første del av oppgave 3 (figur 7), frem til der det står «Oppskrift:», finner vi i oppgave 1. Oppgaven ber elevene om å lese og skrive av koden for så å gjette på resultatet. Tanken bak dette er at elevene skal ha mulighet til å bruke informasjonen de lærte fra oppgave 1 til å tenke seg frem til hvordan markøren vil bevege seg. Deretter introduserer den to nye kommandoer uten forklaring, *penup( )* og *pendown( ). Penup( )* vil få markøren til å slutte å tegne, se for deg at du løfter pennen fra papiret, og *pendown( )* setter ned pennen på papiret og markøren kan fortsette å tegne når den er i bevegelse. Programmet vil tegne tre parallelle linjer som er sammenkoblet av to linjer som står vinkelrett på de parallelle linjene, og ser da ut som en kantete «S», se figur 9. Målet for elevene vil være å få programmet til å slutte å tegne de to ekstra linjene slik at de kun står igjen med tre parallelle linjer.

Neste del av oppgaven starter med en oppskrift skrevet i pseudokode, det vil si en kode forklart med vanlige ord istedenfor kommandoer. Oppgaven ber elevene om å få programmet til å tegne ti slike parallelle linjer de nettopp tegnet tre av, tidligere i oppgaven. De presenteres med én ny kommando, *ycor( )*, som henter ut informasjon om hvilken y-koordinat markøren har. Her åpner oppgaven opp for to mulige løsninger. De kan enten fortsette på samme måte som første del av oppgaven, gjentatt til de får ti linjer, eller forsøke å tolke og følge oppskriften.

Oppgave 3

```
from turtle import *

setx(200)
sety(50)
setx(0)
sety(100)
setx(200)
```

1. Les av koden ovenfor, og gjett på resultatet.
2. Skriv av koden, og kjør programmet.
3. Endre programmet slik at resultatet blir tre parallelle linjer. Du får bruk for kommandoene **penup()** og **pendown()**.

Oppskrift:

1. Penn opp
2. Sett y til -150
3. Gjenta følgende 10 ganger:
   (a) Penn ned
   (b) Sett x til 200
   (c) Sett x til 0
   (d) Penn opp
   (e) Øk y-koordinaten med 30

Endre programmet slik at du tegner 10 slike parallelle linjer. Se oppskriften ovenfor. Her er noen kodebiter du kan få bruk for:

```
penup()
pendown()
for i in range(10)
sety(ycor() + 30)
```

Figur 7: Oppgave 3 i oppgavesettet – Delt i to deler, før og etter oppskriften

3.3.1.2.2 Liste over kommandoer brukt i oppgavene

Her legger jeg ved alle kommandoene elevene fikk utlevert i oppgavesettet. Hver kommando er kort forklart, slik at leseren forstår hva elevene måtte utforske seg frem til. Denne oversikten ble ikke gitt til elevene for at de skulle måtte utforske programmet og oppdage funksjonene selv. Tabell 1 gir fullstendig oversikt over alle kommandoer og kodelinjer som ble brukt i elevenes prosess med å løse oppgavesettet.

Kommando	Forklaring
from turtle import *	Importerer turtle-biblioteket slik at vi kan bruke kommandoer knyttet til grafisk tegning
setx()	Gir markøren en x-koordinat
sety()	Gir markøren en y-koordinat
hideturtle()	Gjemmer figuren som brukes for å vise hvor markøren er til enhver tid
speed()	Endrer hastigheten markøren beveger seg med
shape("turtle")	Endrer utseende på markøren fra den originale pilen til en liten skilpadde (derav turtle)
for i in range():	En løkke som kjører ethvert program som ligger innrykket under denne kommandoen antall ganger som står inne i parentesen.
forward()	Beveger markøren et gitt antall skritt i den retningen den peker, antall skritt er gitt av tallet inne i parentesen.
left()	Endrer retningen markøren peker, bruker grader og går da fra 0 til 360
penup()	"Løfter" pennen fra papiret slik at når markøren beveger seg etter denne kommandoen er gitt, vil markøren ikke tegne der den beveger seg.
pendown()	Setter ned pennen på papiret igjen og reverserer da effekten penup() kommandoen har
ycor()	En variabel som lagrer hvilken y-koordinat, markøren har til enhver tid

Tabell 1: Forklaring av kommandoer brukt i oppgavesettet

3.3.2 Lydopptak, skjermopptak og transkripsjon

I denne studien valgte jeg kun å bruke lydopptak uten videoopptak, ettersom jeg ikke vurderte det nødvendig å trenge video av elevene da jeg heller brukte skjermopptak. Videoopptak fanger opp mer sensitiv informasjon enn lydopptak og i tillegg var videoinformasjon ikke nyttig nok i denne studien til å forsvare bruken. På Sikt (u.å.) sine nettsider spesifiseres det at du bør ha et

godt grunnlag for å filme barn, ettersom dette kan føre med seg etiske utfordringer. I tillegg påpeker Tjora (2021) at videoopptak kan fort føre til for stor mengde dokumentasjon. Lydopptaket plukker opp samtalene mellom elevene og sammen med skjermopptaket ser jeg på dette som nok dokumentasjon til å kunne innhente informasjon jeg var ute etter.

For å få et bilde av hvordan elevene bruker programmeringsprogram og hvordan de skriver kode så jeg det relevant å bruke et skjermopptaksprogram. I denne studien brukte jeg programmet «Open Broadcaster Software» (OBS) som er et gratis program med åpen kildekode (Bailey, u.å.). Åpen kildekode vil si at andre kan undersøke og endre programkoden selv dersom de ønsker det, på egen datamaskin (Perens, 1999). Skjermopptak er en datainnsamlingsmetode som går ut på å bruke en programvare til å ta opptak av en dataskjerm eller et nettbrett, og har blitt mer populært i nyere tid. Dette gir deg mulighet til å få med deg alle de små valgene elevene tar på datamaskinen. Hvis elevene for eksempel angrer på det de har gjort og sletter dette, vil du på skjermopptaket få med deg hva de gjorde først og hvordan de valgte å endre (Beiler, Brevik, & Christiansen, 2021). Ved å ta opptak av skjermen de arbeider på, får jeg mulighet til å se hvordan elever som har liten eller ingen programmeringserfaring skriver og feilsøker kode. Sammen med lydopptaket vil dette gi meg et bedre bilde av hvordan elevene tenker.

Da jeg var ferdig med å samle inn datamaterialet begynte jeg å transkribere all data. Dette beskrives av Kvale og Brinkmann (2015) som en endring fra lydspråk til skriftspråk. Ettersom jeg ikke valgte å bruke videoopptak var det viktig å starte denne prosessen fortløpende for å ha observasjonen friskt i minnet. Ut fra kvalitet på opptak og egen evne er transkripsjon en tidkrevende prosess (Kvale & Brinkmann, 2015). Jeg brukte i gjennomsnitt 1 time per 10 minutter opptak på første gjennomgang, og med et opptak på til sammen rundt 100 minutter tok dette meg 10 timer. I transkripsjonsprosessen valgte jeg å skrive på bokmål både for å skjerme identiteten til skolen og for at det skulle være enklere å bruke transkripsjonene i denne oppgaven, som er skrevet på bokmål. For å få mest mulig informasjon ut av datamaterialet valgte jeg å skrive alt sa, gjennom hele lyd- og skjermopptaket.

3.4 METODE FOR ANALYSE

For at denne studien skal være til nytte for annen fremtidig forskning knyttet til samme materiale, er det viktig å beskrive hvilke analytiske modeller som har blitt brukt i møte med dette

datamaterialet. Det vil være vanskelig å sammenligne eller analysere forskning hvis vi ikke beskriver nettopp dette (Braun & Clarke, 2006). I dette kapittelet vil jeg gjøre rede for valg av metode for analyse og hvordan jeg gikk frem for å analysere datamaterialet.

3.4.1 Tematisk analyse og deduktiv/induktiv tilnærming

Tematisk analyse er en metode for å analysere og identifisere et datamateriale, samt organisere og beskrive det i detalj (Braun & Clarke, 2006). Gjennom en tematisk analyse er målet å identifisere hvilke prosesser elevene i dette datamaterialet gjennomgår. Når vi velger å bruke en tematisk analyse, kan vi velge to forskjellige måter å identifisere temaer: induktiv og deduktiv tilnærming. Ved bruk av en induktiv tilnærming velger du deg ut temaer underveis i den analytiske prosessen der du oppdager at visse temaer går igjen eller er interessante. Dette gjør at du får en analyse som plukker opp mer dybde av tanker og prosesser. Det må likevel påpekes at ettersom forskeren ofte har en viss bakgrunn i teori, vil det alltid være en type vinkling og tanke bak temaene som blir oppdaget eller plukket ut (Braun & Clarke, 2006). En deduktiv eller teoretisk analyse bærer preg av å lete etter allerede utplukkede temaer før selve analysen har blitt gjennomgått. Analysen vil være mer spisset og vil da potensielt sile bort data som kan være interessante, men ikke passer inn i de forhåndsvalgte temaene. Valget av tematisk analyse står og faller på hvilken måte du ønsker å kode datamaterialet, enten med et spesifikt forskningsspørsmål i tankene (deduktiv) eller underveis i prosessen (induktiv) (Braun & Clarke, 2006). I denne studien har jeg valgt å bruke en induktiv tematisk analyse, men med den baktanke at kodene og temaene jeg danner, må passe inn under forskningsspørsmålet mitt.

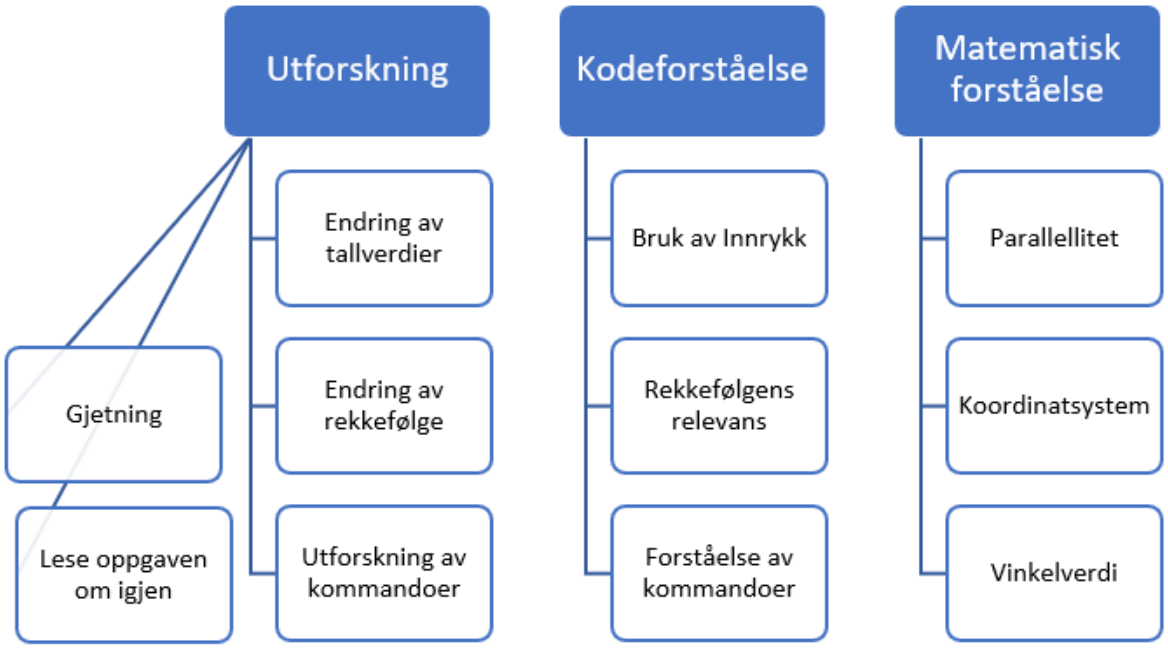
3.4.2 Analyse av datamaterialet

Som første del av den tematiske analysen er det viktig å bli godt kjent med datamaterialet (Braun & Clarke, 2006; Nowell, Norris, White, & Moules, 2017). Gjennom observasjonen av datamaterialet oppdaget jeg at de ulike elevgruppene arbeidet mye på samme måte, som ga meg et grunnlag for ideer til noen innledende koder i tematiseringen. Jeg brukte deretter tid på å se og lytte meg gjennom skjerm- og lydopptakene slik at transkripsjonene ble så nøyaktige som mulig. I tillegg ga dette meg muligheten til å registrere enda mer av hva elevene samtalte om, og hvilke tanker og idéer de delte med hverandre. Dette gjorde jeg flere ganger slik at jeg skulle få med meg mest mulig av tankerekkene som dukket opp.

For å få en bedre oversikt og lettere tematisere datamaterialet valgte jeg å importere transkripsjonene inn i et analyseprogram for kvalitativ data, NVivo (UiS, 2022). Andre del av den tematiske analysen er å danne noen innledende koder som igjen skal danne et grunnlag for videre tematisering (Braun & Clarke, 2006; Nowell et al., 2017). Jeg arbeidet meg systematisk og grundig gjennom hele datamaterialet, og formet koder som jeg anså som relevante opp mot forskningsspørsmålet mitt. Braun og Clarke (2006) påpeker viktigheten av akkurat dette, og at hver del av datamaterialet har like stor verdi. I mitt datamateriale observerte jeg derimot at noen av datamaterialene var enklere å hente ut relevante koder fra enn andre. De innledende kodene jeg benyttet var: «Endring av tallverdier», «Endring av rekkefølge», «Gjetning», «Lese oppgaven om igjen», «Utforskning av kommandoer», «Koordinater», «Parallellitet», «Vinkelverdi», «Bruk av innrykk», «Forståelse av kommandoer» og «Rekkefølgens relevans».

Etter den innledende kodingen skal disse kodene sammenfattes under forskjellige overordnede tema. Ifølge Braun og Clarke (2006) skal analysens del tre gå ut på å danne disse temaene. Deretter skal temaene gjennomgås, ses over om mulig endring i det tematiske hierarkiet og til slutt navnsettes. I denne prosessen er det viktig å tenke godt over hvor mye tid du skal bruke på akkurat dette. King (2004) poengterer at du kan holde på med disse punktene i all evighet. Den vanskeligste delen med dette er å vite når du skal stoppe opp og være fornøyd med prosessen.

Da kodene var innledet, valgte jeg å sortere dem i tre ulike tema og to innledende koder ble stående uten tema. Kodene «Endring av tallverdier», «Endring av rekkefølge» og «Utforskning av kommandoer» ble sammenfattet under temaet «Prøve og feile». Under temaet «Kodeforståelse» samlet jeg kodene «Bruk av innrykk», «Rekkefølgens relevans» og «Forståelse av kommandoer». Det siste temaet fikk navnet «Matematisk forståelse» og inneholdt kodene «Parallellitet», «Koordinatsystem» og «Vinkelverdi». Utenom disse stod kodene «Gjetning» og «Lese oppgaven om igjen». I neste del av analysen gjennomgikk jeg temaene på nytt. Dette fikk meg til å både navngi «Prøve og feile» om til «Utforskning» og samle kodene «Gjetning» og «Lese oppgaven om igjen» under dette temaet. Grunnlaget for dette var å gi temaet et bredere omfang og samle de to utenforstående kodene under et overordnet tema. Som neste del av arbeidet valgte jeg å visualisere kodehierarkiet som en figur som jeg designet i Microsoft Word, se figur 8.



Figur 8: Tematisk analytisk modell over temaer og respektive koder knyttet til datamaterialet

3.4.2.1 Tidslinjer

Schoenfeld (2016) presenterer tre forskjellige tidslinjer som beskriver problemløseres forskjellige problemløsningsprosesser. Etter å ha tematisk analysert datamaterialet mitt, valgte jeg å designe en tidslinje til hver av gruppene. Jeg brukte de samme seks temaene i min inndeling og som jeg har oversatt til norsk, med følgende temaer: lese, analysere, utforske, planlegge, implementere og verifisere[1]. Disse skrev jeg inn i Microsoft Excel for å markere hvor mye tid elevene brukte på hvert enkelt av temaene, og da dette skjedde i prosessen. Dette var en vanskeligere prosess enn originalt tenkt på to måter. I første omgang var det utfordrende å definere når elevene leste oppgaven og når de analyserte oppgaven, når utforsket de og når la de en plan. Etter å ha hørt og sett gjennom skjermopptakene flere ganger mens jeg leste meg samtidig gjennom transkripsjonene, landet jeg på de tre tidslinjene som jeg presenterer i mine funn i kapittel 4.4. Den andre mer uforventede grunnen til at dette ble vanskelig, var å finne ut hvordan den skulle designes. For å få et design som lignet mest mulig på det som presenteres av

[1] read, analyze, explore, plan, implement and verify (Schoenfeld, 2016, s.24)

Schoenfeld (2016), endte jeg opp med å designe seks sammenhengende horisontale stolper der jeg skjuler alle de delene elevene ikke gjorde underveis i prosessen.

3.5 STUDIENS TROVERDIGHET

Postholm og Jacobsen (2018) presiserer at ettersom forskning ofte etterprøves, utfordres og bygges videre på i senere tid, vil kvaliteten på forskningen ikke kun beskrives av funnene. Det er derfor viktig at forskeren presenterer i hvilken kontekst studien er i og hvilke valg som ble tatt underveis i prosessen. Kvaliteten vil styrkes av å forankre forskningen i annen forskning. For å diskutere forskningens kvalitet er det nødvendig å se på to aspekter: forskningens validitet (gyldighet) og reliabilitet (pålitelighet) (Postholm & Jacobsen, 2018).

3.5.1 Forskningens reliabilitet

Forskningens reliabilitet handler om hvor troverdigde resultatene av studien er og har ofte sammenheng med i hvilken grad resultatene kan repliseres av andre forskere. I en kvalitativ undersøkelse vil det være vanskelig å få akkurat samme resultat på grunn av at møtet mellom forsker og deltaker vil alltid være ulikt (Postholm & Jacobsen, 2018). I løpet av prosessen med denne studien har jeg prøvd å være så transparent som mulig om mine valg knyttet til påliteligheten til forskningen min. Jeg har ingen tidligere kjennskap til elevene i studien og innsamlingen av datamaterialet vil derfor ikke påvirkes av dette. Ved intervju av elever kan du stå i en situasjon der eleven kun svarer det de tror du ønsker å høre (Postholm & Jacobsen, 2018). Ettersom det ikke ble utført et intervju, vil ikke dette påvirke datamaterialets kvalitet. Forskningen min tar utgangspunkt i at elevene ikke har forhåndskunnskaper i programmeringsspråket Python og funnene mine er fortsatt interessante selv om elevene starter med lav programmeringskompetanse. Denne studien ble skrevet i 2023, tre år etter at LK20 innførte programmering inn i skolene. Det er derfor nødvendig å se denne studien i en kontekst der elevene mangler programmeringskunnskapen som etter hvert kreves. Mangel på tid og tilbakemeldinger fra samtykkeerklæringene gjorde at utvalget ble nokså lite, som påvirker mangfoldet i resultatene mine. Ved å bruke skjerm- og lydopptak fikk jeg mulighet til å gå nøye over alt som ble gjort og sagt slik at minst mulig data forsvant i analyseprosessen (Postholm & Jacobsen, 2018). Gjennom observasjonsprosessen valgte jeg ved noen hendelser å gå bort fra min rolle som observatør og inn i en rolle som veileder, som ifølge Dalland (2017) kan føre til en

ødeleggelse av informasjonsverdien. Valgene ble gjort for å sikre mest mulig datamateriale over den korte tiden jeg hadde til rådighet.

3.5.2 Forskningens validitet

Forskningens validitet beskriver hvorvidt konklusjonen fra studien stemmer overens med det som er undersøkt; svarer studien på det den spør om? En god kilde til høy validitet er at forskningen forankres i annen relevant forskning (Postholm & Jacobsen, 2018). Gjennom studien har jeg stilt meg kritisk til om hvorvidt forskningsspørsmålet mitt blir besvart, og om mine tolkninger representerer det som blir studert. Analysearbeidet ble gjort av meg alene, som kan være med på å svekke denne studiens validitet ettersom jeg ikke har hatt noen andre å diskutere arbeidet med. I kvalitativ forskning beskriver Postholm og Jacobsen (2018) at begrepene som dannes under analysen bør være meningsfulle, både for leseren, men også for forskningsfeltet. Begrepene jeg dannet i analysen beskrives i kapittel 3.4.2 der jeg i tillegg beskriver prosessen for begrepenes utvalg. De ble dannet gjennom flere falsifikasjoner og revisjoner for at datamaterialet skulle bli mest mulig troverdig, og dermed styrke studiens validitet (Kvale & Brinkmann, 2015).

Inkludert i validitet er om funnene er generaliserbare og dermed kan overføres til andre studier og kontekster (Postholm & Jacobsen, 2018). I denne studien deltar ni elever ved 10. trinn på én ungdomsskole og er dermed for få informanter til å kunne si noe om hvorvidt mine resultater stemmer overens med resten av landets elever. Derimot kan resultatene brukes i andre studier og være med på å bidra inn i videre forskning. Ettersom utvalget mitt ikke er stort eller bredt fordelt utover landet, har det vært viktig for meg å gå tydelig frem hvordan jeg har gjennomført min analyse slik at resultatene kan brukes mest mulig riktig av annen forskning. I tillegg til dette har jeg prøvd å forankre mine resultater i godt etablert forskning for å øke studiens validitet (Postholm & Jacobsen, 2018).

3.6 FORSKNINGSETISK VURDERING

Etiske problemer kan dukke opp når du utforsker menneskers tanker og erfaringer, og deretter deler dette med offentligheten (Kvale & Brinkmann, 2015). Derfor er det ditt ansvar som forsker å planlegge og ta hensyn til etiske problemer underveis i hele forskningsprosessen. For å vite at jeg har gått mest mulig etisk frem i studien har jeg forholdt meg til NESH (*National Committee for Research Ethics in the Social Sciences and the Humanities*) sine retningslinjer for

forskningsetikk (NESH, 2022). Denne studien har blitt godkjent av Sikt (se vedlegg C), tidligere NSD, og prosjektskissen av studien ble godkjent av UiS. For å få lov til å gjennomføre en studie er det et krav at Sikt godkjenner studien på forhånd. I meldeskjemaet ble jeg bedt om å presentere studien, beskrive hvilke personopplysninger jeg skulle behandle, begrunnelse på nødvendigheten av å behandle personopplysninger, beskrivelse av utvalg av informanter, varighet på studien samt behandling og sikkerhet av de innsamlede personopplysningene. Som svar på meldeskjemaet fikk jeg råd for at behandlingen av personopplysningene skulle være lovlige, som jeg har brukt som retningslinjer gjennom hele studien.

For at deltakerne i studien skal bli gitt oppmerksom på studiens design, formål og potensiell risiko eller fordeler det innebærer å delta i studien var det viktig å gi et informert samtykke (Kvale & Brinkmann, 2015; NESH, 2022). Jeg informerte først deltakerne i studien ved en muntlig forklaring av hva det å delta i studien innebar og hva som krevdes av dem hvis de ville delta. Deretter fikk de utdelt en skriftlig samtykkeerklæring (se vedlegg A og B) som læreren skrev under på selv og elevene måtte få foreldre eller foresattes underskrift hvis de ønsket å delta. I samtykkeerklæringen spesifiserer jeg hva som blir gjort med den innsamlede dataen, hvordan den lagres, detaljer om deltakelsens anonymitet, hvordan deltakerne kan trekke seg fra studien når som helst i prosessen, muligheter til å få innsikt, slette eller endre feilaktige personopplysninger om seg selv eller sitt eget barn. NESH (2022) spesifiserer barns spesielle krav til beskyttelse. Det er derfor viktig å tilpasse studiens metode etter hva som er barnets beste. I denne studien valgte jeg å droppe videoopptak for å minimere mengden innsamlede personopplysninger.

Deltakeres anonymitet er med på å skjerme deres identitet og integritet og gjøres for å ikke kunne spore informasjonen i studien tilbake til deltakerne (NESH, 2022). Gjennom denne studien har jeg fokusert på å holde deltakerne anonymisert så langt det lar seg gjøre til enhver tid. Etter å ha samlet datamaterialet og transkribert ble alle navn endret for å bevare anonymiteten til deltakerne. Skjerm- og lydopptakene ble lagret på en kryptert minnepenn som ved enhver tid var sikkert låst inne. Dette ble gjort for å bevare datamaterialet på en mest mulig sikker måte. Etter endt analyse ble lyd- og skjermopptak slettet for å unngå å oppbevare sensitive personopplysninger over for lang tid.

4 RESULTAT

Jeg presenterer i dette kapittelet funnene mine fra analysen av datamaterialet som belyser forskningsspørsmålet: Hvilke problemløsningsprosesser kommer til uttrykk i elevers introduksjon til tekstbasert programmering i matematikk på ungdomstrinnet? Analysen baseres på transkripsjoner fra datamaterialet. Transkripsjonene er analysert opp mot det teoretiske rammeverket i kapittel 2. Kapittelet deles inn i fire deler, hvor jeg starter med å presentere hvordan elevene brukte utforskning til å arbeide med problemløsningsoppgavene (kap. 4.1). Deretter presenterer jeg hvordan elevene etter hvert begynte å forstå seg på hvordan selve koden fungerte (kap. 4.2). I kapittel 4.3 presenterer jeg hvor elevene viser en matematisk forståelse i problemløsningsprosessen. Avslutningsvis presenterer jeg en oversikt over tidsbruken til elevgruppene knyttet til Schoenfelds (2016) tidslinjemodeller (kap. 4.4).

4.1 UTFORSKNING

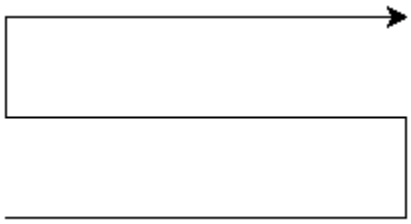
Underveis i den tematiske analysen valgte jeg å definere fem koder som falt under temaet utforskning. Kodene jeg definerte var: «Endring av tallverdier», «Endring av rekkefølge», «Utforskning av kommandoer», «Gjetning» og «Lese oppgave om igjen». Alle de fem kodene som beskrives videre i dette kapittelet viser til en del av elevenes problemløsningsprosess der de lette etter forståelse i oppgaven. Elevene brukte ulike metoder for å utforske programmeringsverktøyet, koden og oppgaven for å gjøre fremskritt mot en løsning på problemet. Resultatene fra dette temaet presenteres i de fem neste delkapitlene, der jeg i tillegg vil sammenligne de tre gruppene av datamaterialet.

4.1.1 Endring av rekkefølge

Et programmeringsverktøy leser kommandoer fra topp til bunn og kjører dem i denne rekkefølgen, derfor er rekkefølgen elevene bruker i koden utslagsgivende for resultatet. Oppgave 1 og 3 er oppgaver der elevene forsøkte å endre rekkefølge på kodelinjene for å prøve å løse oppgaven. I oppgave 1 informeres elevene i oppgaveteksten at de må endre rekkefølgen på kodelinjene for at programmet skal tegne en «T». Alle gruppene begynte på oppgaven uten å tilstrekkelig lese oppgaveteksten og gruppene forstod derfor ikke at oppgaven ba dem endre rekkefølgen på kodelinjene. Når de først leste hele oppgaveteksten, fant elevene løsningen på oppgaven. De benyttet seg derfor ikke av endring av rekkefølge for å løse et problem, men som

en ren instruks. Funn av lesing av oppgavetekst om igjen vil jeg presentere nærmere i kapittel 4.1.5.

I motsetning til i oppgave 1 gir oppgave 3 ingen indikasjoner i oppgaveteksten at elevene trenger å endre rekkefølgen på kodelinjene for å løse oppgaven. Funnene mine viser imidlertid at gruppene på eget initiativ endret rekkefølgen på kodelinjene i et forsøk på å komme nærmere en løsning på problemet. Oppgaven presenterte et program som tegner tre parallelle linjer, men som vist i figur 9 er disse linjene sammenkoblet.



Figur 9: Oppgave 3 – Tre sammenkoblede parallelle linjer

Elevenes mål er å fjerne disse sammenkoblingene ved hjelp av kommandoer for å kun beholde de tre parallelle linjene. I andre del av oppgaven skal elevene lage ti parallelle linjer som videreutvikling av første del av oppgaven. Begge deler av oppgaven krever at elevene forstår hvor de skal plassere kommandoene *penup( )* og *pendown( )* for at markøren skal slutte å tegne på de ønskede områdene. Mine funn viser at alle gruppene prøvde seg frem ved å plassere disse kommandoene inn mellom kodelinjene til resultatet ble tre parallelle linjer uten sammenkoblinger. Et utdrag fra gruppe 3 viser hvordan de testet ut hva som skjedde da de plasserte kommandoen *penup( )* mellom kodelinjene for å få koden til å slutte å tegne mellom de parallelle linjene.

206. **Mia**: Da må vi jo ha, men den kan ikke gå opp, vi skal ha 3 linjer.

207. **Kåre**: Ja det kan ikke være noe imellom linjene.

208. **Mia**: Men vi bruker de der, går det ikke an å bruke de der noe.

209. **Kåre**: Jo, penup.

210. **Mia**: Ja. At vi tar de i midten her kanskje.

211. **Kåre**: Ok.

212. **Mia**: En strek og så. Ok.

Selv om de bruker denne kommandoen korrekt valgte de å plassere tallverdier inne i parentesen til kommandoen som gjorde at de fikk en feilkode. Dette gjorde at det tok flere minutter med testing før de oppdaget at det var tallverdien som utløste feilkoden.

4.1.2 Endring av tallverdier

En strategi som ble mye brukt av elevene tidlig i hver av oppgavene var å teste ut hva som skjer hvis de endrer tallverdiene i koden. I tabell 2 ser vi fordelingen av prosent av samtaler som angikk endring av tallverdier. Av denne figuren kan vi se at i alle gruppene ligger rundt 10 % av samtalene på endring av tallverdier, der gruppe 1 samtaler litt mer og gruppe 2 litt mindre.

Gruppe nummer	Prosent av samtale ang. endring av tallverdier
1	12 %
2	9.5 %
3	10 %

Tabell 2: Analyseresultat – Oversikt over mengde av all samtale som angikk endring av tallverdier

Oppgave 1 i oppgavesettet går ut på å endre rekkefølge på kodelinjer for å få programmet til å tegne bokstaven «T». Oppgaven består av seks linjer med kode, der det kun er fire av linjene som trenger å endre rekkefølge for å få ønsket resultat. Gruppe 1 og gruppe 2 brukte tid på å prøve å endre tallverdiene i denne oppgaven. Dette forekom ikke hos gruppe 3. I gruppe 1 dukket følgende kommentar opp da de var usikre på veien videre.

56. **Anna**: Vi kan jo bare prøve å skrive inn sånn forskjellige tall inntil vi får (ler). Det, det går an, jeg kan liksom notere hva vi har gjort, hvilken rekkefølge vi har gjort.
57. **Ida**: Ja det kan du.

Et forslag dukket opp hos gruppe 2 om endringer av tallverdi, men en elev på gruppa poengterte at det var rekkefølgen som var viktig.

60. **Johan**: Ok, bytt tallene da, istedenfor 45 så skriver du 46

61. **Stein**: Vi skal bare kopiere koden?
62. **Tor**: Vent da, vi må
63. **Line**: Prøv å ta minus 45
64. **Tor**: Se her, plasser i riktig rekkefølge, det er jo ikke i rett rekkefølge.

I oppgave 2 ble elevene presentert for en kode som inneholdt en for-løkke som tegnet en trekant når de kjørte programmet. Målet med oppgaven var å få programmet til å tegne en firkant, femkant og en sekskant ved å endre vinkelverdier og verdiene på antall iterasjoner av løkken. Gruppe 3 klarte ikke å finne ut at tallverdien i kommandoen *left( )* var antall grader i den ytre vinkelen til figuren og endte opp med å prøve seg frem til de fant riktig tall.

95. **Kåre**: Ta den opp igjen på sånn 100. Og så tar du. Den var størrelse ikke sant?
96. **Mia**: Ja det skulle ikke ha noe å si.
97. **Kåre**: Ok, da tar vi den på sånn 100. 130.
98. **Mia**: Ok.
99. **Kåre**: Eller, jeg vet ikke.
100. **Mia**: Vi kan prøve. Jeg tror ikke det går. Ok, men ta den på mindre enn 100 da.
101. **Kåre**: Ok.
102. **Mia**: Ta den på sånn 80 (.) Ok den var litt nærmere da.
103. **Kåre**: [utydelig]
104. **Mia**: 90?
105. **Kåre**: Ja
106. **Mia**: Ah! Fantastisk.

Gruppe 1 forstod i møte med denne oppgaven at verdien som skulle endres var en vinkelverdi, men måtte gjette seg frem til løsningen på femkanten da de ikke visste hvilken vinkelverdi hvert hjørne krevde for å tegne en femkant, i dette programmet. Derimot forstod gruppe 2 både at verdien de skulle endre var en vinkelverdi og de klarte å regne seg frem til hvilken verdi de skulle frem til, ved å dividere 360 grader på antall sider i mangekanten.

4.1.3 Utforskning av kommandoer

Ingen av elevene i datamaterialet hadde tidligere erfaring med programmeringsspråket Python og de brukte derfor mye tid på å utforske hvilken effekt hver av kommandoene hadde i koden. Alle oppgavene er laget slik at elevene skal skrive av koden i oppgaveteksten og kjøre programmet før de begynner å endre noe. Funnene mine viser at dette ga mye informasjon til elevene om hva kommandoene samlet gjorde, og de brukte da tid på å endre enkeltkommandoer for å se hvilken effekt denne endringen hadde på programmet. Dette vises i følgende utdrag fra gruppe 2.

> 105. **Stein**: Vi bare ser hva det blir hvis vi skriver det slik som dette her, så kan vi, ja, endre det.
>
> 106. **Johan**: Vi bare skriver av, og så ser vi hva vi har først
>
> 107. **Stein og Tor**: Ja

Her kan vi se at de nettopp har skrevet av programmet, og nå vil se hva programmet gjør før de ønsker å endre noe. Videre viser funnene mine at elevene prøvde å endre tallverdiene, ikke for å komme frem til en løsning slik som presenteres i kapittel 4.1.2, men for å forstå hva kommandoen gjorde. I oppgave 2 er målet å endre på tallverdier i en kode for å få programmet til å tegne en firkant, en femkant og en sekskant. De har mulighet til å endre tre tallverdier, men kun to av verdiene er nødvendige å endre for å få ønsket resultat. Hvis tallverdier i kommandoen *forward( )* endres, vil lengden på linjene som tegnes bli lengre og figuren vil øke i størrelse. Derimot hvis du endrer tallverdien til kommandoen *range( )* i for-løkken vil antall streker som tegnes øke. I følgende utdrag fra transkripsjonen av datamaterialet kan vi se at gruppe 3 forstod først hva tallverdien i *range( )* endret, deretter hva tallverdien i *forward( )* endret, før de tilslutt forstod hva tallverdien i den siste kommandoen *left( )* endret.

> 81. **Kåre**: Ok, prøv å ta range på 4, siden det var 3 nå så ble det en trekant, så hvis vi tar 4 så blir det kanskje en firkant. Og så.
>
> 82. **Mia**: Ok, det ble fortsatt en trekant. Åh, nå gikk den bare en til bort sånn.
>
> 83. **Kåre**: Ja.
>
> 84. **Mia**: Ok, ja. Og så

85. **Kåre**: Så. Vett ikke. Vi bare prøver noe. Ehm.

86. **Mia**: Nå tegnet den vertfall 4 streker da.

87. **Kåre**: Hvis vi tar forward til sånn 150 liksom. Da ble den bare større.

88. **Mia**: Ok da er det. Ja, da må det være den da, vi må stille på.

89. **Kåre**: Ja

4.1.4 Gjetning

Når du møter en oppgave, kan det være vanskelig å få satt ord på hva du tror oppgaven gjør uten å ha tilstrekkelig informasjon fra før. I oppgave 3 er målet med første del av oppgaven at elevene skal sette ord på akkurat dette. Her må de skrive av koden og gjette på resultatet av programmet før de kjører koden. Når de kjører koden vil de se at den tegner tre parallelle linjer som er sammenkoblet av vinkelrette linjer, se figur 9. Kodelinjene bruker samme kommandoer som i oppgave 1, *setx( )* og *sety( ),* men har én ekstra kommando av denne typen og tallverdiene er annerledes. Ettersom de nettopp har løst oppgave 1 som krever at de får programmet til å tegne bokstaven «T» kommer det frem fra funnene at både gruppe 1 og 2 på et tidspunkt i gjetningen trodde programmet i oppgave 3 tegnet en bokstav. Gruppe 1 gjettet først på at programmet tegnet en figur før de så sammenheng mellom kommandoene brukt i oppgave 1 og 3, og gjettet at koden laget en bokstav.

181. **Ida**: Kanskje den lager en figur da?

182. **Anna**: Ja jeg tror det óg. Men når den går null da, kanskje den bare bytter retning, men bare den beveger seg ikke, jeg vet ikke.

183. **Ida**: Eller kanskje den lager en bokstav.

184. **Anna**: Ehm, ok hva tror vi.

185. **Ida**: Skal vi velge om vi tror det blir en figur eller en bokstav eller noe sånt? Fordi på oppgave 1 så ble det jo en bokstav. Og på det der for in range og alle de der, da ble det liksom en figur.

186. **Anna**: Kanskje bokstav.

Gruppe 2 tenkte i samme baner som gruppe 1, men påpekte at de trodde det kunne være en mangekant, ettersom de nettopp hadde løst oppgave 2 som krevde at de skulle designe tre forskjellige mangekanter.

197. **Stein**: Les av koden ovenfor og gjett på resultatet. Hva tror, hva tror vi det blir? Jeg tror det kommer til å bli en sånn skikkelig ti-kant eller noe

198. **Tor**: Tretten-kant

199. **Stein**: Tretten-kant

200. **Johan**: Hvorfor er det en form? Kanskje det blir en

201. **Stein**: [utydelig]

202. **Johan**: Kanskje det blir en bokstav?

203. **Line**: Ja

204. **Tor**: Jojo, men har det vært noe annet

205. **Johan**: Alfakrøll

206. **Line**: Ja, det har vært én «T»

207. **Johan**: «T»-en hadde minus i seg den óg

208. **Stein**: Men denne

209. **Tor**: Siden denne, denne her har du jo, den går bort 45 og så går den tilbake

210. **Stein**: Det blir en, det blir en. Ok, jeg tipper det blir en firkant, eller en type kant.

211. **Johan:** Ja jeg tror ikke det blir en bokstav.

4.1.5 Lese oppgave om igjen

I en problemløsningsprosess vil det være essensielt å kunne oppdage når du har kommet inn på et blindspor. Da vil du få muligheten til å se tilbake og lese over oppgaven på ny for å se om du har gått glipp av eller oversett informasjon (Schoenfeld, 2016). Koden «lese oppgave om igjen» ble notert i analysen av transkripsjonen samlet sett 23 ganger for alle gruppene, der gruppe 1 hadde

tydelig færrest instanser. I alle oppgavene i oppgavesettet får elevene tydelig beskjed i starten av hver oppgave å skrive av koden. I tillegg til dette står det spesifikke instrukser på hva oppgaven ønsker at elevene skal komme frem til etter å ha skrevet av kodelinjene inn i programmeringsprogrammet de arbeider i. Fra funnene mine finner jeg mange eksempler på at elevene kom til et punkt der de ikke helt forstod hvorfor programmet ikke vil gjøre det de ønsket og måtte da se tilbake på hva oppgaven spurte om. Det kommer i tillegg frem at de ikke alltid fikk med seg all informasjon fra koden i oppgaveteksten inn i sitt eget program. En kodelinje som alle gruppene overså viste seg å være linjen *from turtle import **, som importerer turtle-biblioteket og gjør at koden kan bruke alle kommandoer den trenger for å bruke et grafisk grensesnitt. Et eksempel på dette kan vi se fra følgende utdrag fra transkripsjonen av gruppe 2.

19. **Stein**: Error, hmm interessant.

20. **Johan**: Er det S-T-Y

21. **Stein**: Det er riktig er det ikke? Har vi gjort noe meget feil nå?

22. **Tor**: From turtle to import

23. **Line**: Har vi skrevet from turtle to import

24. **Johan**: Nei det har vi ikke.

Etter å ha skrevet inn koden og kjørt programmet fikk elevene en feilkode som sa at kommandoene de brukte ikke var definert. Elevene leste ikke hva feilkoden sa, men fikk med seg at noe var feil og leste oppgaven om igjen for å se etter mangler. De oppdaget at de ikke hadde skrevet inn importerings-kodelinjen og prøvde dette med vellykket resultat.

I oppgaveteksten til oppgave 1 finner vi følgende instruks: «… Plasser kodelinjene i riktig rekkefølge …» Dette måtte alle gruppene lese om igjen etter å ha innledningsvis lest oppgaveteksten. Ingen av gruppene klarte å løse oppgaven uten å lese oppgaveteksten om igjen. Transkripsjonen fra gruppe 2 i kapittel 4.1.2 avsluttes i linje 64 med at den ene eleven påpekte at det var rekkefølgen som skulle endres på. Samtalen fortsatte slik:

64. **Tor**: Se her, plasser i riktig rekkefølge, det er jo ikke i rett rekkefølge.

65. **Johan**: Vi har jo ikke lest oppgaven en gang.

66. **Tor**: Nei det er jo nettopp det. Det er jo ikke i rett rekkefølge.

67. **Stein**: Åja, herregud

68. **Tor**: Vi må jo endre på rekkefølgen

Funnene som bygger opp koden «lese oppgave om igjen» finner vi spesielt tydelig i andre del av oppgave 3 i oppgavesettet. I første del av oppgaven skal de modifisere en kode slik at den lager tre parallelle linjer. Videre i oppgaven får de beskjed om å endre programmet slik at det nå tegner ti slike parallelle linjer. I tillegg til dette får de en oppskrift på hvordan dette kan gjøres, skrevet med alminnelig tekst som kode, også kalt pseudokode. Denne delen av oppgaven nådde ikke gruppe 3 å gjøre, men både gruppe 1 og 2 endte opp med å prøve på et eget design uten å ha først lest hele oppgaveteksten. Gruppe 1 valgte å prøve å kopiere koden fra første del av oppgaven. Ettersom programmet deres tegnet tre linjer valgte de å kopiere og lime inn to ekstra ganger i tillegg til en ekstra linje slik at den skulle tegne ti linjer. Ved å kjøre dette programmet oppdaget de at det nå tegnet ti linjer, men linjene overlappet hverandre slik at det fortsatt ble stående igjen som tre parallelle linjer. Det de ikke hadde tatt med i beregningene var at tallverdiene i kommandoene også ble kopiert. Etter å ha brukt rundt fem minutter på å regne seg frem til hvilke verdier de trengte, viser funnene mine følgende utdrag:

322. **Ida**: Ok, 200, 50 og så 100 så 200. Så her, 50, hva skjer hvis vi skriver 400 her. Skal vi se om det skjer noe stor endring?

323. **Anna**: Vi bør ikke kanskje bruke oppskriften?

324. **Ida**: Nei, oi. Ja, vet du hva. Jeg tror vi prøve oppskriften.

325. **Anna**: [utydelig] freestyle. Ok, (ler) skal vi delete alt og bruke oppskriften?

326. **Ida**: Ja, er du enig?

327. **Ola**: Mm

Funnene fra gruppe 2 i samme oppgave viser at de også ønsket å bruke det de allerede hadde, men heller bruke en for-løkke som de brukte i oppgave 2 til å gjenta dette ti ganger. Denne metoden er en del av løsningen, men den mangler noen trinn for at den ikke ender opp på samme måte som løsningen til gruppe 1. Videre viser funnene at ingen av elevene i gruppe 2 helt visste

hvordan de skulle gjøre dette, før Line tok datamaskinen og begynte å følge oppskriften mens de andre elevene diskuterte videre. Dette resulterte i at problemet ble løst og de klarte å få programmet til å tegne ti parallelle linjer.

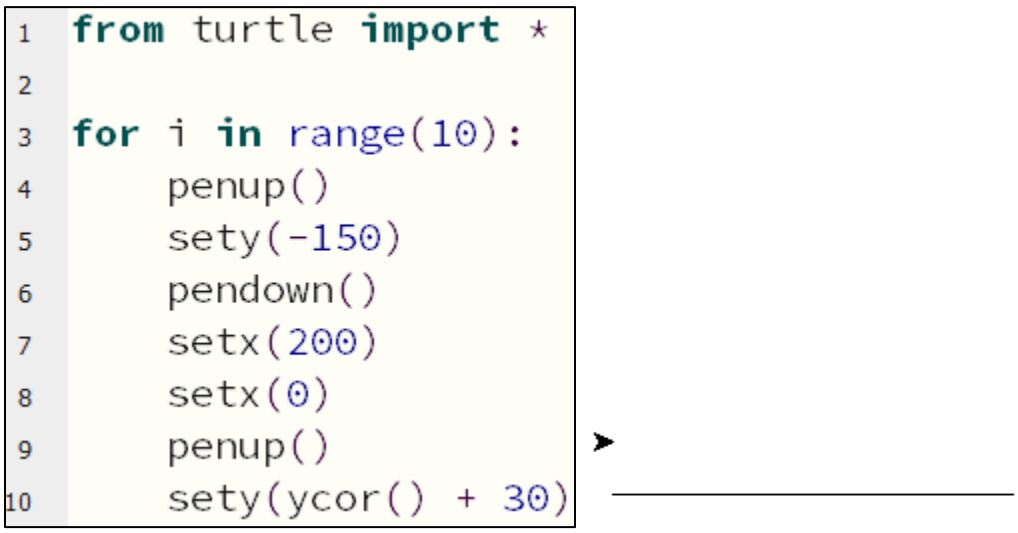
4.2 KODEFORSTÅELSE

Etter hvert som du arbeider med et programmeringsspråk, vil du forstå mer og mer av hvordan syntaks og kommandoer fungerer individuelt og sammen. I analysen av datamaterialet mitt definerte jeg flere koder som beskriver når elevene viste forståelse underveis i problemløsningsprosessen. Disse kodene ble fordelt under to forskjellige temaer. Ettersom noen beskriver elevenes forståelse av programmering og hvordan koden fungerer, mens andre beskriver elevenes forståelse innen matematiske emner valgte jeg å kalle de to temaene: «Kodeforståelse» og «Matematisk forståelse». Under temaet kodeforståelse havnet kodene: «Rekkefølgens relevans», «Forståelse av kommandoer» og «Bruk av innrykk». I de neste tre delkapitlene vil jeg presentere resultatene fra analysen knyttet til disse tre kodene samtidig som jeg sammenligner funnene fra de tre forskjellige gruppene.

4.2.1 Rekkefølgens relevans

Dette kapittelet bygger videre på funnene fra kapittel 4.1.1, der vi kan se hvordan elevene utforsker hvilken effekt rekkefølgen på kodelinjene har på koden. Funnene jeg skal presentere i dette kapittelet viser ikke hvordan elevene prøver seg frem, men hvordan de viser at de har forstått relevansen av rekkefølgen på kodelinjene i et program. I oppgave 1 er målet for elevene å endre rekkefølgen av kodelinjene slik at programmet tegner bokstaven «T». Funnene mine viser at etter gjennomførelsen av denne oppgaven innså elevene at rekkefølgen til kodelinjene i et program var relevant. Hensikten med å ha denne oppgaven som en introduksjonsoppgave i starten av oppgavesettet var nettopp å fremme denne forståelsen tidlig hos elevene. Ettersom oppgave 2 kun krever endring av tallverdier, var det ingen funn som viste noe av relevans knyttet til dette kapittelets kode. Derimot viser funnene fra analysen at alle gruppene på forskjellige måter viste forståelse for relevansen av rekkefølgen til kodelinjer. I andre del av oppgave 3 når elevene skal få programmet til å tegne 10 parallelle linjer viste gruppe 1 denne typen forståelse. De fulgte oppskriften og da de kjørte programmet tegnet det ikke ti parallelle linjer, men ti linjer som overlappet til én linje. Etter 2-3 minutter med tenking og lesing av oppgaven om igjen, viste Anna til at hun trodde at deler av koden måtte skje før for-løkken iverksettes. Fra figur 10 kan vi

se hvordan elevene originalt skrev koden sin. Her ser vi at all kode skjer ti ganger inne i forløkken, som fører til at programmet i praksis tegner kun én linje ti ganger. Figur 11 viser oss hvordan elevene løste oppgaven etter å ha forstått forskjellen mellom inne i en løkke og på utsiden av en løkke.



Figur 10: Gruppe 1's første kode av oppgave 3 del 2 etter oppskrift – Resultatet av kjørt program ga én tegnet linje.

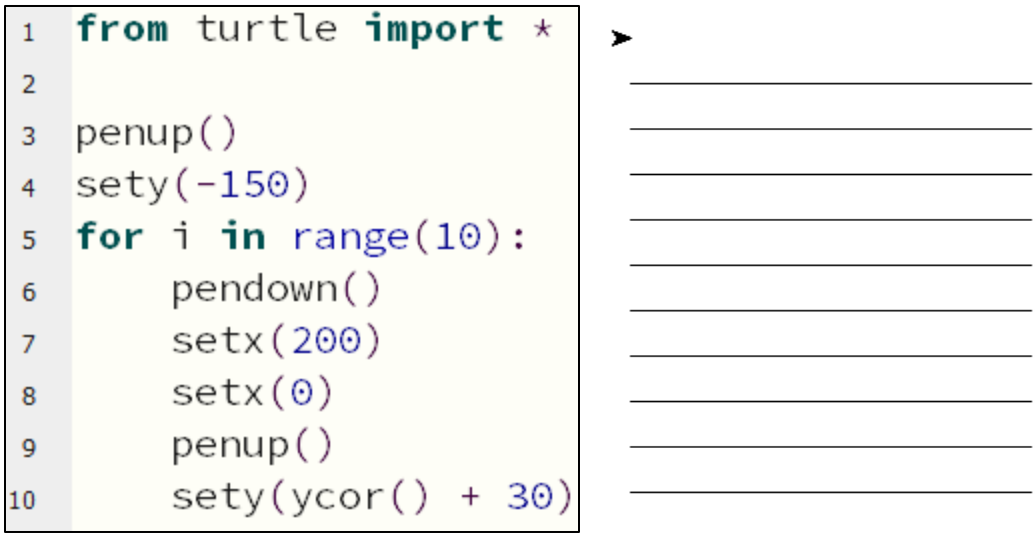
Et utdrag fra transkripsjonen viser oss samtalen som skjedde mellom figur 10 og figur 11:

386. **Anna**: Med at penn opp og den sett y, at de må være over kanskje for in range.

387. **Ida**: Vent. Og så. Åh!

388. **Ola**: Ja.

389. **Anna**: Yaaay!



Figur 11: Gruppe 1's ferdige kode av oppgave 3 del 2 – Resultatet av kjørt program ga 10 tegnede parallelle linjer

I første del av oppgave 3 brukte gruppe 2 omtrent 15 minutter av tiden på å utforske hva programmet gjorde og hva som skjedde hvis de endret på tallverdiene. Dette gjorde at de endret såpass mye av rekkefølge og tallverdier at de selv ble forvirret av hvor de var i oppgaven. Til slutt ble jeg som observatør spurt om de var: «… helt på bærtur?». Dette innledet følgende samtale hentet fra transkripsjonen:

448. **Observatør**: Neida, dere kan ha gode av å vite at dere trenger ikke å endre på noen av tallene, og dere kan bruke de i den rekkefølgen

449. **Stein**: Åja, vi må bare legge inn penup og pendown og sånn.

450. **Johan**: Så vi må bare skrive de imellom de

451. **Tor**: Ok, hvis du sletter det og skriver det på ny, så kan vi bare

Selv om jeg måtte inn og gi dem et spesifikt tips om at rekkefølgen på de originale kodelinjene ikke skulle endres, klarte de selv å se at det var derfor nødvendig å putte kommandoene *penup( )* og *pendown( )* mellom kodelinjene. Her viser elevene at de forstår relevansen av rekkefølgen på kodelinjene.

Funnene fra analysen av transkripsjonen fra gruppe 3 viser at underveis i denne oppgaven forstod Kåre hvordan rekkefølgen på kodelinjene er relevante og forklarte dette videre til Mia mens han prøvde å løse oppgaven. Det er relevant å vite at i dette utdraget er det Mia som skrev inn i

programmet mens Kåre forklarte hva hun skal skrive. Vi kan i tillegg observere fra utdraget akkurat da Mia forstod hva kommandoene *pendown( )* og *penup( )* gjorde.

282. **Kåre**: Ta pendown, siden når du tar penup så tar du pennen opp og da tegner du ikke. Du må ta pendown etter, set

283. **Mia**: Åja, at du løfter. Åja, den løfter, ok.

284. **Kåre**: Ok, etter set x null så kan du skrive pendown. Og så, vent vent vent, vi er ikke ferdige. Og så etter det så skriver du penup. Nei ikke der, under set x. Skriver du penup. Og så under der skriver du set x null.

285. **Mia**: Nei (kommentar til skrivefeil ikke til Kåres utsagn)

286. **Kåre**: Og så under der skriver du pendown. Og så set x 200.

(5 sekunder med tenkepause)

287. **Mia**: Nei. Ehm, da må vi jo legge til den da, er det ikke den vi mangler? Legge på den her nede.

288. **Kåre**: Jo (.) Etter penup, men det er lengre nede da.

4.2.2 Forståelse av kommandoer

Et programmeringsspråk er satt sammen av kommandoer som er spesifikke til det programmeringsspråket du bruker, men med en god del likheter til andre programmeringsspråk. Disse kommandoene har en streng syntaks og må brukes og skrives slik, ellers vil du få en feilkode når du kjører programmet. Selv om Python er et programmeringsspråk som higer etter å være så lesbart nært vanlig engelsk som mulig, vil det fortsatt være vanskelig for en nybegynner å forstå hva en kommando gjør. I tillegg er datamaterialet hentet fra 10. klassinger som selv påpeker følgende kommentar, hentet fra transkripsjonen av datamaterialet:

348. **Tor**: Burde hatt et engelsk kurs før dette her.

Funnene mine viser at det stemmer for alle gruppene at det er vanskelig å forstå en kommando som nybegynner. Ingen av gruppene påpekte at de kunne forstå hva en kommando ville gjøre uten å ha først kjørt programmet minst én gang. Her kan det være viktig å påpeke at ingen av

elevene ga noe som helst muntlig indikasjon på at de gjettet på funksjonaliteten til kommandoen første gang de ble møtt av en ny kommando. Da elevene derimot hadde fått mulighet til å teste ut programmene viser funnene mine at elevene i forskjellig grad forstod hvordan kommandoene fungerte. Enkelte kommandoer viste seg å være lettere å forstå, mens andre var vanskeligere å tyde.

Gruppe 1 er den eneste gruppen som viser i funnene at de forstod egenskapene til kommandoene *setx( )* og *sety( )*. De forstod underveis i oppgave 3 at kommandoene ga x- og y-koordinater som markøren skulle flytte seg til, i tillegg til at de så at hele det grafiske grensesnittet fungerte som et koordinatsystem. Dette presenteres i detalj i kapittel 4.3.2, ettersom det viser en tydelig matematisk forståelse knyttet til programmeringen de arbeidet med. Et utdrag fra funnene mine viser kommandoforståelse knyttet til denne oppgaven. Her ser vi hvordan Anna forstod ved hjelp av Ida hvordan du uttaler kommandoene, og dermed forstod hva navnet tilsier at kommandoen gjør.

330. **Ida**: Og så sett y til, er ikke det sånn

331. **Anna**: Sett, ah det er det det betyr, sett y, jeg forstod det ikke før nå. Jeg leste det som sety (ler).

En kommando som alle gruppene klarte å resonere seg frem til en forklaring på var: *for i in range( ):*. Dette er en kodelinje som initierer en løkke som gjennomføres et antall ganger lik den tallverdien som står inne i parentesen etter *range*. For-løkken ble elevene eksponert for i oppgave 2, der de skulle få programmet til å lage flere mangekanter. Oppgaveteksten ga elevene et program som designet en trekant og kodelinjen så da slik ut: *for i in range(3):*. Alle gruppene prøvde å endre tallverdien fra 3 til 4 for å lage en firkant, og fikk da programmet til å tegne fire linjer. Da alle gruppene valgte å endre denne verdien og så kjøre programmet uten å endre flere verdier, tegnet programmet fortsatt en trekant, men med fire linjer slik at den siste linjen overlappet den første. Funnene av analysen av transkripsjonen viser nettopp dette fra et utdrag fra gruppe 2.

131. **Johan**: Vi skal lage en firkant, så da mangler vi jo, firkant delen. Så da, ett til hjørne. Mm.

132. **Line**: Kan vi ikke bare prøve å skrive, 4 inn her.

133. **Stein**: Ja, det kan du.

134. **Johan**: Ja prøv det.

135. **Tor**: Det ble en trekant til

136. **Johan**: Men nå gikk den 4 ganger. Så den skal være, det skal være 4 inni der. Og så må den bare gå.

Her kan vi se at Tor påpekte at figuren fortsatt tegnet kun en trekant og da viste til at det ikke skjedde noen endring. Johan påpekte derimot at markøren beveget seg fire ganger, i motsetning til de tre bevegelsene markøren beveget seg før de endret tallverdien i *range( )* fra 3 til 4. Denne forståelsen kan vi fra funnene mine finne igjen da samme gruppe arbeidet med oppgave 3. Her skal de prøve å tegne ti parallelle linjer og hadde allerede en kode som tegnet tre parallelle linjer. Et utdrag fra transkripsjonen viser følgende utsagn fra eleven Tor hvor han viste at han forstod hvilken effekt han trodde kodelinjen *for i in range(10)* hadde.

502. **Tor**: Går det an å bare ta det vi hadde og så for in range 10 da?

Av funnene kommer det samme frem fra gruppe 1 i samme oppgave der Anna sa følgende:

337. **Anna**: for i in range 10, er det ikke det? At det da, det gjør det 10 ganger

Ettersom gruppe 3 ikke nådde å gjøre denne delen av oppgaven finnes det ikke noe datamateriale på om de forstod kommandoen godt nok til å bruke denne informasjonen videre i en senere oppgave slik både gruppe 1 og 2 viste.

Gruppe 3 viste også forståelse av kommandoer, som kommer tydeligst frem av funnene i oppgave 2. De prøvde som de andre to gruppene å endre tallverdien i *range( )* kommandoen for å se om det er denne som avgjorde antall sider av mangekanten. I tillegg valgte de å prøve å endre på verdien inne i kommandoen *forward( )* fra 100 til 150. Her brukte de utforskningsmetoden som er presentert i kapittel 4.1.2, men da de oppdaget hva denne endringen gjorde med programmet, ytret de en forståelse av kommandoens egenskaper. Dette fremkommer av linje 87 i utdraget fra gruppe 2 i samme kapittel. De oppdaget at kommandoen *forward( )* endret lengden på linjestykkene og var derfor ikke relevant å endre i denne oppgaven. Ettersom denne kommandoen ikke deltok i andre oppgaver, viser ikke funnene mine om de klarte å bruke denne

informasjonen videre i en annen oppgave. Dette er den eneste gruppen som endret verdien til denne kommandoen, og de andre gruppene ga ingen indikasjoner på hvorvidt de forstod hvilken effekt den hadde på programmet.

4.2.3 Bruk av innrykk

Syntaksen til et programmeringsspråk er tydelig på hvordan du får lov til å skrive uten å få feilkoder. Innrykk i koden brukes stort sett for å øke leseligheten til en kode og vil dermed ikke gi feilkoder ved «feil» bruk. Derimot brukes dette i Python for å poengtere nivåforskjeller på kodelinjene (Henderson, 2009). Av figur 12 ser vi to ulike kodeversjoner av oppgave 2 i oppgavesettet. Koden til venstre bruker innrykk korrekt for å poengtere at koden under *for i in range(3):* er en del av løkken og skal da kjøres tre ganger. Koden til høyre presenterer en kode der vi ikke bruker innrykk og for-løkken vil da være tom, og vi vil få innrykks-feilkoden «IndentationError: expected an indented block».

```
1  from turtle import *
2
3  speed(2)
4  shape("turtle")
5
6  for i in range(3):
7      forward(100)
8      left(120)
```

```
1  from turtle import *
2
3  speed(2)
4  shape("turtle")
5
6  for i in range(3):
7  forward(100)
8  left(120)
```

Figur 12: Eksempel på riktig (venstre) og feil (høyre) bruk av innrykk i kode

Elevenes forståelse av bruken av innrykk dukket opp enkelte steder i funnene mine. Både hos gruppe 1 og 3 måtte jeg som observatør bryte inn og gi noen tips for at de skulle forstå hva som var feil i møte med feilkoder. I begge gruppene fikk de en «IndentationError», men forstod ikke hva det betydde. Jeg så meg nødt til å hinte at de burde undersøke om koden deres var helt lik koden i oppgaveteksten. Etter å ha dobbeltsjekket koden spurte gruppe 1 om de trengte at kodelinjene *forward(100)* og *left(120)* skulle være innrykket under *for i in range(3):*, som jeg bekreftet. I gruppe 3 viste Mia at hun forstod deler av viktigheten med innrykk, og rykket den

relevante koden inn, slik at den stemte overens med oppgaveteksten. Dessverre fullførte hun aldri tankerekken sin høyt, og vi står igjen med følgende utdrag:

67. **Mia**: Å, det, ta sånn mellomrom kanskje. Sånn at den starter samtidig som

Gruppe 2 fikk også en «IndentationError», men spurte ikke meg om bistand. De brukte tid på å undersøke og fant raskt ut at de manglet kolon etter *range( ).* Før de igjen kjørte programmet passet Line på å legge inn et innrykk på kodelinjene under *for i in range( ):*, som gjorde at de løste problemet.

121. **Stein**: Kanskje vi skal ta bort den der. To prikker etter 3-en.

122. **Line**: Å, ja.

123. **Stein**: Kolon.

124. **Johan**: Ja

125. **Line**: Sånn

126. **Johan**: Ok, run det nå

127. (Line bruker innrykk på linjene slik at forward(100) og left(120) er hoppet inn under for-løkken)

4.3 MATEMATISK FORSTÅELSE

Funnene fra både kapittel 4.1 og 4.2 viser hvordan elevene først brukte forskjellige strategier for å bli kjent med programmeringsspråk og programvare. Deretter ser vi at elevene begynte å forstå hvordan kommandoer fungerer og deler av hvor viktig programmeringsspråkets syntaks er. Funnene i dette kapittelet fokuserer, som introdusert i kapittel 4.2, på hvorvidt elevene viste matematisk forståelse knyttet til programmeringsoppgavene. Oppgavene er bygget opp slik at du ikke nødvendigvis må kunne et høyt nivå av matematikk, men hvis du forstår hvordan kommandoene og matematikken samarbeider vil det gi deg en fordel i problemløsningsprosessen. De tre kodene som sammenfatter dette temaet i analysen av transkripsjonen er: «Parallellitet» , «Koordinatsystem» og «Vinkelverdi».

4.3.1 Parallellitet

En del av oppgaveteksten i oppgave 3 lyder som følger: «Endre programmet slik at resultatet blir tre parallelle linjer.» For å kunne ha mulighet til å løse denne oppgaven krever det at elevene vet hva som definerer parallellitet. To parallelle linjer er per definisjon to linjer som aldri kommer til å krysse hverandre; matematisk sett har linjene samme stigningstall (PARALLELLITET KILDE). Det kommer frem av funnene at alle gruppene har en viss forståelse som gruppe av hva det vil si at to linjer er parallelle. Gruppen som stikker seg ut er gruppe 3, der de aldri sa med ord hva det vil si at to linjer er parallelle. Likevel forstod de poenget med oppgaven og kom frem til løsningen. Denne prosessen tok rundt ti minutter, men dette viste det seg å være andre grunner til, som vi skal se nærmere på i kapittel 4.3.2.

I begge gruppene 1 og 2 finner vi samme type utsagn der elevene tydelig beskriver hva de mener er parallellitet. I gruppe 1 har vi følgende samtale mellom Anna og Ida:

237. **Anna**: Ok, eh 3 parallelle linjer og. Er ikke det når de ikke «toucher» hverandre?

238. **Ida**: Jo, de bare følger hverandre. Opp sånn da?

Videre i funnene fra gruppe 2 finner vi følgende samtale om hvorvidt linjene de nettopp har prøvd å tegne er parallelle eller ikke. Her legger vi merke til at denne samtalen skjer på linje 237 hos gruppe 1, men ikke før linje 459 hos gruppe 2.

459. **Tor**: De er ikke parallelle, men det er 3 linjer

460. **Johan**: De er jo parallelle

461. **Line**: Ja, parallelle linjer er sånne linjer som aldri kommer til å treffes

4.3.2 Koordinatsystem

Turtle-biblioteket introduserer muligheten til å bruke et grafisk grensesnitt. Dette grafiske grensesnittet er bygget opp som et koordinatsystem, der posisjonen til markøren til enhver tid har en definert x- og y-koordinat. Ved å endre tallverdiene i *setx( )* og *sety( )* endrer du henholdsvis x- og y-koordinatene til markøren, som får den til å bevege seg og tegner en linje fra den nåværende koordinaten til den nye koordinaten du har plottet inn. Funnene i denne studien viser at alle gruppene forstod at hvis du endrer verdiene vil du få programmet til å tegne. I tillegg viste

alle en forståelse av at *setx( )* forflytter markøren i horisontal retning og *sety( )* forflytter markøren i vertikal retning.

Prosessen ved å først ikke forstå, deretter oppdage ny informasjon og så bruke denne nye informasjonen kommer veldig tydelig frem i funnene av analysen av gruppe 1. De startet med å være forvirret av hvordan de to bevegelseskommandoene fungerte i oppgave 1. Spesielt var det kommandoen *setx(0)* som elevene ikke fikk til å gi mening. Denne kommandoen med tallverdi null, viser funnene at går igjen som et forvirringsmoment hos flere av elevene. Gruppen begynte med å se på kommandoene som bevegelse i en viss retning et visst antall lengdeenheter. Da ga det ikke mening å bruke en tallverdi null til å forflytte markøren.

213. **Anna**: 200 der, og så går den 50 opp da, men så går den 0

214. **Ida**: Og så 100 og så

215. **Anna**: Det gir ikke mening. Hmmm, den gjør liksom den samme bare den liksom bytter retning

Det er ikke før de leste oppskriften i oppgave 3 at Anna forstod hvordan matematikken fungerer sammen med disse kommandoene.

217. **Anna**: Oppskrift. 1 penn opp, 2 sett y til minus 150, 3 gjenta følgende 10 ganger … A penn ned, B sett x til 200, C sett x til 0, D penn opp, E øk y koordinatene med 30. Åja det er koordinater! Det har jeg ikke tenkt på før nå**.**

Basert på denne informasjonen klarte Anna å forstå hvordan kommandoen *setx(0)* fungerer. Videre viser funnene hvordan Anna brukte denne tilegnede kunnskapen til å forklare de to andre på gruppen hvordan oppgaven skulle løses.

242. **Ida**: Og så, var det den opp her. … Har de tallene noe å si, de som er inni

243. **Anna**: Ja, fordi du har jo sånn x sant? Da starter den jo på 0, men siden den er x 200 så går den liksom til verdien på x-linjen gange 200

244. **Ida**: Ja

245. **Anna**: Og så, blir verdien på y-aksen bare 50 så da går den opp her 50 sånn at det liksom y er 50 her. Og så x er 200 her. Og så blir den da er x 0 igjen så da istedenfor å liksom

gå tilbake igjen her, så er den jo fortsatt 50 på y. Sånn den går her, og så beveger den seg lengre

I funnene fra gruppe 2 kan vi se at Tor oppdaget hvorfor tallverdien null ga mening, uten å først ha lest informasjonen i oppskriften fra oppgave 3. Dette prøvde han å forklare til sine medelever uten at dette ble nærmere diskutert. Det kommer ikke frem av funnene hvorvidt medelevene forstod hva han mente med følgende utsagn:

305. **Tor**: Jo men se her, dette er jo på en måte kanten på, rett før den går opp, så er dette kanten. Så hvis den går minus 100 så går den lengre ut, og hvis den går pluss så går den lengre inn. Når den er på null så er den det samme som der den startet.

Gruppe 1 hadde nesten brukt opp all tid da de fortsatt stod fast på første del av oppgave 3. For å få elevene i mål, tok jeg valget om å gi et hint om at tallverdiene representerte koordinater. Dette førte til at de endret én verdi og dermed løste oppgaven.

4.3.3 Vinkelverdi

I oppgave 2 møter elevene en kodeblokk som bruker kommandoene *left( )* og *forward( ).* Kommandoen *left( )* er laget slik at tallverdien du putter inn i parentesen representerer en endring i retning for markøren, mens *forward( )* får markøren til å bevege seg et antall skritt angitt av tallverdien i parentesen. Hvis du skriver *forward(100),* deretter *left(90)* og til slutt *forward(100)* vil markøren tegne en horisontal strek som deretter snur 90 grader og tegner en vertikal strek som er like lang.

Funnene mine viser at både gruppe 1 og 2 forstod etter hvert dette. Fra de først fikk avklart hvilken av kommandoene som får markøren til å bevege seg i en annen retning, gikk det kort tid til de resonnerte seg frem til å bruke vinkelverdier i kommandoen *left( ).* Den første mangekanten de fikk beskjed om å tegne var en firkant. Begge gruppene prøvde da å skrive tallverdien 90 inne i denne kommandoen, med vellykket resultat. Dette kommer frem av samtalen mellom elevene i gruppe 2, der de først diskuterte om det er vinkelsummen av figuren de burde bruke. Før de til slutt landet på å teste om det stemte å bruke 90 grader.

141. **Line**: Ja, det var den, hva, hva er summen av en firkant?

142. **Johan**: 360

143. **Line**: Da kan vi sikkert skrive 360

144. **Tor**: Nei, men nei. Da skulle det stått 180 der. Siden det er jo 90 grader på en rettvinklet trekant

145. **Johan**: 90 på den her?

146. **Stein**: Skal vi sjekke hva vi får først

147. **Johan**: Ja bare se hva som skjer nå. Om det blir helt.

148. **Stein**: Nei, det

149. **Johan**: Det endret seg ikke. Det, det er left

150. **Stein**: Eh jeg vet, se.

151. **Tor**: 180, prøv å sett den på 360 og se hva som skjer.

152. **Johan**: Sett den høyere, må ikke den være høyere.

153. **Stein**: 90 og så

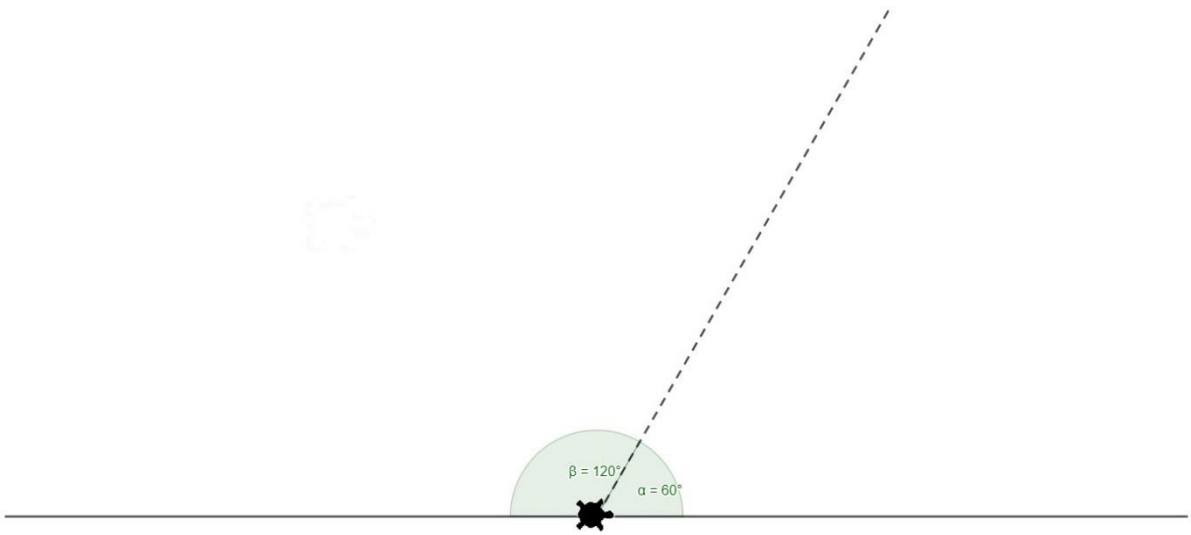
154. **Johan**: Hva 90 90? Åh da har vi det.

Videre viser funnene at store deler av resten av oppgaven gikk ut på hvordan de fant hvilken tallverdi hver ytre vinkel har i en femkant og sekskant. Metoden for å løse dette problemet er forskjellig hos gruppe 1 og gruppe 2. Gruppe 2 gikk for en enkel løsning og dermed spurte meg om de kunne bruke en kalkulator. De brukte følgende metode for å finne korrekt verdi for den ytre vinkelen i mangekanten:

$$\frac{360°}{antall\ sider\ i\ mangekanten} = ytre\ vinkel$$

Gruppe 1 brukte samme metode som gruppe 2, men tenkte at de måtte bruke hoderegning. Dette gjorde at de visste cirka hvilken verdi vinkelen var, og brukte et utgangspunkt til å prøve og feile seg frem til riktig svar. I forsøket på å finne den ytre vinkelen i en femkant, prøvde de først 60 grader og så at dette ga dem en perfekt regulær sekskant som de brukte i siste del av oppgaven.

Denne oppgaven starter med å oppgi vinkelverdien 120, som bidrar til å tegne en trekant. Ettersom programmet tegner en regulær trekant kan dette ikke være verdien til den indre vinkelen i trekanten, og må derfor være verdien til den ytre vinkelen. En sekskants indre vinkel er 120 grader og dens ytre vinkel må derfor være 60 grader. Turtle-programmet tegner ikke vinkler basert på et fast punkt, men roterer de gitte antall gradene rundt dens egen retning. Dette medfører at når du skal tegne en regulær sekskant ved hjelp av programmet i oppgave 2, kan du ikke gi beskjed om den indre vinkelverdien. Da vil programmet først tegne en 120 graders vinkel som går mot venstre og skilpadden vil være på toppen av denne. Neste steg vil da måtte være å få programmet til å gå mot høyre i en vinkel med verdi 300 eller negativ verdi 60, men ettersom dette skal være en løkke vil programmet tegne nok en 120 graders vinkel og du vil til slutt tegne en regulær trekant. Figur 13 visualiserer hvorfor det er viktig å bruke de ytre vinkelverdiene til en mangekant i konstruksjonsprosessen. I denne figuren vil den ytre vinkelen vil alltid være konstant 60 grader med utgangspunkt i retningen skilpadden peker i.



Figur 13: Eksempel på hvorfor verdiene turtle-programmet bruker er de ytre vinklene

Gruppe 3 har viser ingen indikasjoner på å forstå at det er snakk om vinkelverdier og ender opp med å teste seg frem til riktig svar. De bruker fem forsøk på å finne 90 grader, fire forsøk på å finne 72 grader og ett forsøk på å finne 60 grader.

4.4 TIDSLINJER FOR PROBLEMLØSNINGSPROSESSENE

I analysen av transkripsjonene mine valgte jeg å fremstille gruppenes problemløsningsprosess i en tidslinje basert på figurene som presenteres i Schoenfeld (2016, s. 24). I inndelingen av tid har jeg brukt samme temaer som hans fremstillinger og deler dermed tidsbruken i prosessen inn i følgende seks løst oversatte deler: lese, analysere, utforske, planlegge, implementere og verifisere. I dette kapittelet presenteres funnene mine fra denne analysen. Jeg presenterer i dette kapittelet de tre gruppenes tidslinjer for problemløsningsprosessen i form av tre forskjellige figurer og legger frem hvordan hver gruppe brukte den allokerte tiden. Tabell 3 gir en oversikt over prosent av tiden brukt på de forskjellige kategoriene. Det er viktig å påpeke at under kategorien lesing har jeg i tillegg tatt med tiden de brukte på å skrive inn kodelinjene fra oppgaveteksten. Det er hovedsakelig for gruppe 3 dette er mest relevant informasjon, ettersom de brukte en god del mer tid på dette enn de to andre gruppene.

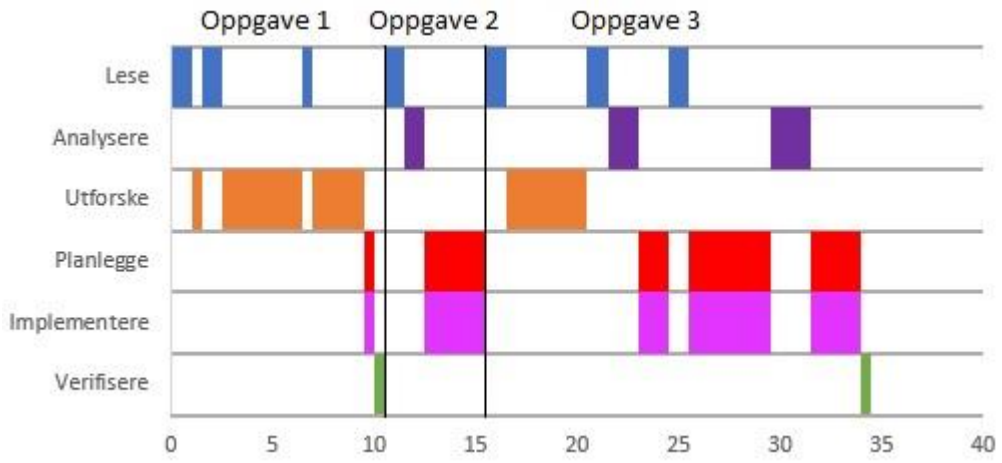
Gruppe nr.	Lese	Analysere	Utforske	Planlegge og implementere	Verifisere
1	18.9 %	13.0 %	31.9 %	33.3 %	2.9 %
2	23.4 %	1.7 %	47.5 %	23.7 %	3.4 %
3	30.7 %	3.2 %	46.7 %	19.4 %	0 %

Tabell 3: Oversikt over tidsbruk til gruppene i prosent

4.4.1 Tidslinje for gruppe 1

Figur 14 viser oss gruppe 1 sin problemløsningsprosess og vi kan se at de begynte prosessen med en god del utforsking avbrutt av en feilkode og et blindspor som gjorde at de måtte lese oppgaveteksten to ganger om igjen. Da de til slutt fant ut at det var rekkefølgen til kodelinjene som skulle endres, brukte de kort tid på å legge en plan og implementere denne planen. Dette var den eneste gruppen som verifiserer hvorfor oppgaveløsningen fungerte. Videre i oppgave 2 brukte de kort tid på å lese gjennom oppgaven før de brukte cirka like lang tid på å prøve å forstå og analysere delene av koden de hadde fått fremstilt. Etter analysen tok det kort tid før de forstod kommandoene med en rask sjekk at teorien deres stemte, og deretter planla og implementerte de

løsningen deres. Oppgave 3 er en sammenhengende todelt oppgave og bærer preg av dette i tidslinjen til gruppe 1. De startet igjen med å lese oppgaven og gjettet på funksjonaliteten til programmet, før de brukte en del tid på å utforske hvordan kodelinjene interagerte med hverandre. Da elevene begynte å forstå mer av kodesammensetningen, sjekket de om igjen hva oppgaven ba dem om. Deretter brukte de tid på å prøve å forstå oppgaveteksten i sammenheng med de kodelinjene de nettopp hadde utforsket. Ved å bruke tid på dette skjedde utførelsen av planen de la uten feil. Neste del av oppgaven ga elevene mulighet til å bruke den koden de allerede hadde laget og videreutvikle denne. Elevene så for seg at de hadde en plan for hvordan utføre dette og brukte nærmere 5 minutter på dette før de fant ut at det mulig ville være lettere å bruke oppskriften de var blitt gitt. Ved å analysere pseudokoden i oppskriften tok det kort tid før de hadde en løsning på andre del av oppgave 3. Denne løsningen ble sjekket og verifisert slik at alle elevene var enige om hva de nettopp hadde gjort.



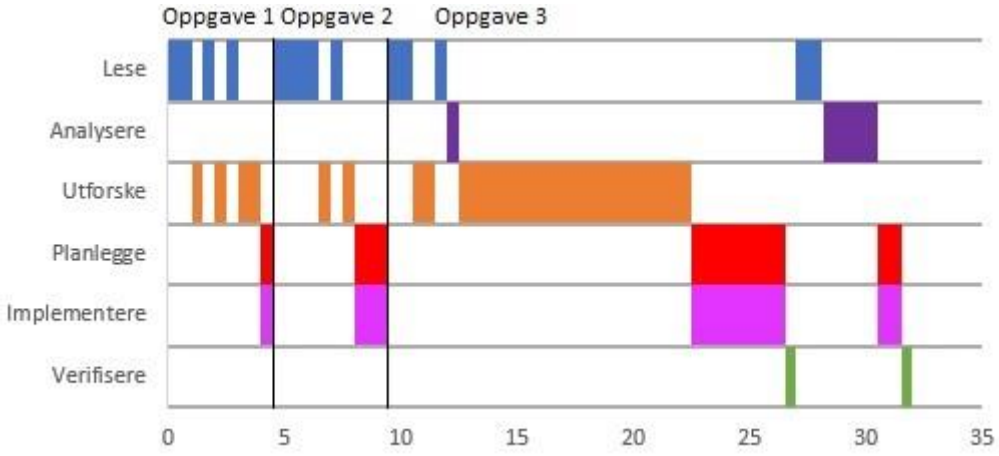
Figur 14: Tidslinje for problemløsningsprosessen til gruppe 1

4.4.2 Tidslinje for gruppe 2

Problemløsningsprosessen til gruppe 2 kan vi lese av tidslinjen i figur 15, som kommer frem av funnene av transkripsjonen til denne gruppen. I oppgave 1 møtte gruppen en del feilkoder da de prøvde å skrive inn og utforske koden. Dette gjorde at de måtte lese oppgaveteksten to ekstra ganger over ganske kort tid. Etter å ha lest om igjen for andre gang påpekte en elev tekstlinjen i oppgaven som presiserte at det var rekkefølgen som skulle endres. Da tok det kort tid før de

bestemte seg for hva de skulle gjøre, og klarte å gjette riktig rekkefølge på første forsøk. Gjennomføringen av oppgave to bestod av en god del lesing i starten for å unngå feil i innskrivingen av koden. Én feilkode dukket opp og ble løst ved å dobbeltsjekke oppgaveteksten. Da de forstod hvilke tallverdier som skulle endres i koden, tok det kort tid før planen ble gjennomført.

Oppgave 3 startet med lesing av oppgave som ba dem om å gjette på resultatet av koden de skrev av. Basert på informasjonen fra løsningene til oppgave 1 og 2 gjettet de at det enten ble en bokstav eller en mangekant. Etter å ha kjørt koden brukte de tid på å prøve å forstå hvorfor figuren ble tegnet slik. De leste at de kunne ha nytte av kommandoene *penup( )* og *pendown( )* og valgte å bytte ut to av de eksisterende kodelinjene med disse kommandoene. En feilkode oppstod da de satt inn tallverdier i parentesene til kommandoene, men ingen leste oppgaveteksten på nytt før de kom frem til en løsning. Ti minutter gikk til utforskning til å forstå syntaksen og funksjonaliteten til *penup( )* og *pendown( )*. Jeg ga gruppen et tips om at programmet forklarer hver kommando i en pop-up boks når du begynner å skrive kommandoen, og én elev forstod hvordan de kunne bruke kommandoene fra denne informasjonen. Løsningen kom til slutt i form av tre parallelle, men ikke like lange, linjer. Ettersom de brukte mye tid på å endre deler av programmet de ikke hadde behov for, ble linjelengdene endret fra den originale lengden de hadde. Dette fikk elevene til å måtte verifisere at løsningen deres var korrekt ut fra oppgaven de var gitt. I inngangen til andre del av oppgave 3 leste de oppgavetekst og oppskrift før de brukte rundt to minutter på å forstå hva de burde gjøre. Etter mange forskjellige forslag på hva oppgaven krevde, tok én elev datamaskinen og fulgte oppskriften etter beste evne. Én retting av skrivefeil senere hadde de løsningen de var ute etter, men sjekket at den stemte overens med oppdraget i teksten.

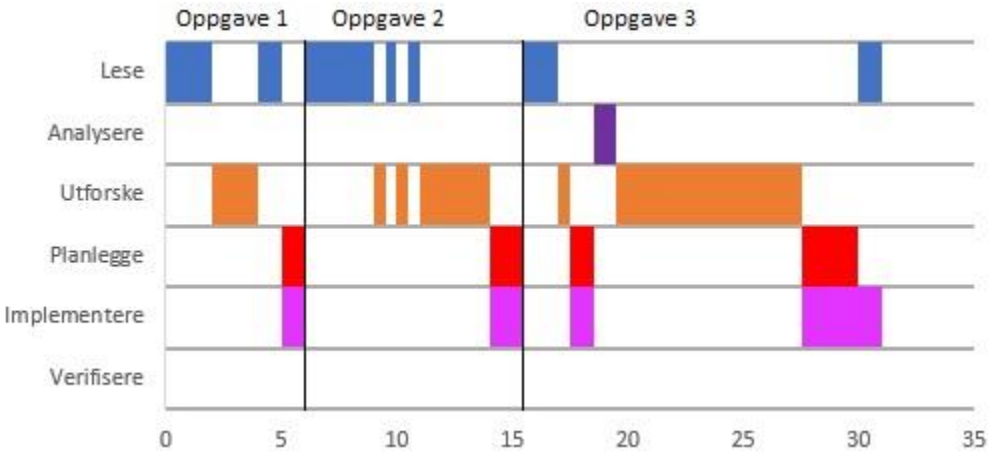


Figur 15: Tidslinje for problemløsningsprosessen til gruppe 2

4.4.3 Tidslinje for gruppe 3

Gruppe 3 sin tidslinje kan vi se i figur 16, der det er viktig å påpeke at under kategorien lesing ligger i tillegg avskriving av oppgaveteksten. Det startet med at Kåre spurte meg om hjelp 5 ganger i løpet av de første 2 minuttene, mens Mia prøvde å finne ut hva de skulle gjøre. Hun ble utvalgt til å være skribent, selv om hun påpekte at hun skriver veldig sakte, men Kåre var tydelig på at han ikke ønsket å skrive. Oppgave 1 bestod derfor av en del rolig lesing, avskrift og utprøving av kommandoer før de sjekket hva oppgaven spurte om og gjettet riktig rekkefølge på første forsøk. Teksten i oppgave 2 brukte de opp mot tre minutter på å lese og skrive av, før de kjørte koden med en feilkode om feil bruk av innrykk (se kapittel 4.2.3 angående dette). Deretter gikk gruppen frem og tilbake mellom lesing og testing før de fikk programmet til å fungere uten feil. Etter mer utforskning oppdaget de hvilke tallverdiendringer som påvirket antall sider i mangekanten og størrelse på den ytre vinkelen. Dette ga dem informasjonen de trengte til å snakke sammen om hvilke verdier de måtte endre til, og gjennomførte dette med hver av de tre mangekantene. Gruppe 3 nådde akkurat å gjennomføre første del av oppgave 3, derfor er den siste planleggings- og implementerings-fasen i figur 16 knyttet til at de klarte å tegne tre parallelle linjer. De startet derimot oppgaven nok en gang med å lese oppgaveteksten et par ganger, før de sjekket resultatet av koden. Denne informasjonen gjorde at de trodde de hadde en plan som kunne fungere, hvilket den imidlertid ikke gjorde. Sammen brukte de tid på å prøve å forstå oppgaven slik at de klarte å finne en løsning. Videre brukte de rundt åtte minutter på å

teste ut de forskjellige kommandoene, nye tallverdier og endre rekkefølge på kodelinjene. Den ene eleven husket tilbake til hvordan de brukte kommandoene *penup( )* og *pendown( )*, og brukte deretter denne informasjonen til å finne en nesten riktig løsning. Da tiden nærmet seg slutt, ga jeg dem et tips om at tallverdiene i *setx( )* og *sety( )* fungerer som koordinater. Én tallverdiendring senere og løsningen var klar.



Figur 16: Tidslinje for problemløsningsprosessen til gruppe 3

5 DISKUSJON

I dette kapittelet skal jeg i lys av funnene mine fra kapittel 4 og den teoretiske bakgrunnen fra kapittel 2, drøfte dette opp mot forskningsspørsmålet mitt: Hvilke problemløsningsprosesser kommer til uttrykk i elevers introduksjon til tekstbasert programmering i matematikk på ungdomstrinnet?

Elevene som deltok i denne casestudien ble gitt et oppgavesett med 3 programmeringsoppgaver knyttet til geometriske figurer innen matematikk. Ingen av gruppene hadde noe tidligere erfaring med programmeringsspråket Python. Selv om elevene hadde varierende mengde erfaring innen blokkprogrammering, ble kodeforståelsen knyttet til tekstbasert programmering en stor del av problemløsningsprosessen. Dette kapittelet deler jeg i tre deler der jeg undersøker og diskuterer i hvilken grad elevenes matematiske- og algoritmiske tenkning kommer til syne underveis i prosessen, samt hva elevene brukte tiden på i problemløsningsprosessen.

5.1 Elevenes matematiske tenkning

For å undersøke elevenes problemløsningsprosess vil det være relevant å observere hvordan elevenes matematiske tenkning stemmer overens med det teoretiske rammeverket. I kapittel 2.1.2 ser vi at Schoenfeld (2016) presenterer fem elementer som det er en konsensus blant forskere at matematisk tenkning deles inn i: kunnskapsbasen, problemløsningsstrategier, monitorering og kontroll, oppfatning og følelser, og praksis. I dette kapittelet vil jeg se nærmere på hvordan elevenes samtaler kan knyttes til elementene.

Det vil være vanskelig for meg å si akkurat hvilken kunnskapsbakgrunn elevene hadde, men basert på at dette var normalt presterende elever i 10. klasse, går det an å danne seg et godt bilde. Det matematiske nivået i oppgavesettet var bygget rundt at de allerede hadde gjennomgått store deler av det geometriske pensumet i matematikkundervisningen, som ifølge LK20 legges på 9. trinn (Kunnskapsdepartementet, 2019). Selv om oppgavene bærer preg av at programmeringen er i sentrum, baseres alle oppgavene på matematikk knyttet til geometri. I oppgave 1 vil det være en fordel å forstå at kommandoene *setx( )* og *sety( )* gir markøren nye koordinater og at det grafiske grensesnittet fungerer som et koordinatsystem. For én av gruppene kommer denne forståelsen tydelig frem, men ikke før i oppgave 3. I delkapittelet «Koordinatsystem» (4.3.2) under kapittelet «Matematisk tenkning» (4.3) leser vi at Anna fra gruppe 1 leste videre i oppgaveteksten i

oppgave 3 til hun kom til linjen «(e) Øk y-koordinaten med 30». Hun stoppet opp og forstod at verdiene de strevde med å forstå i oppgave 1 er koordinater. Dermed klarte hun å bygge bro mellom hennes matematiske forkunnskaper om koordinatsystem og programmeringen. Hun oppdaget ny informasjon som omhandler koordinater, dette har hun allerede liggende i kunnskapsbasen Schoenfeld (2016) skriver om, men hjernen har ikke sett nødvendigheten til å trenge å hente ut denne informasjonen enda. Hun brukte dermed informasjonen hun kunne om koordinater og knyttet det til hvorfor programmeringen fungerte slik den gjorde. Pólya (2004) vektlegger at eleven er helt avhengig av kunnskap om temaet problemet ligger under, for å være i stand til å utvikle en plan.

I oppgaven om å tegne tre forskjellige mangekanter kreves det å kunne regne seg frem til eller å vite hvordan finne den ytre vinkelverdien til de regulære mangekantene firkant, femkant og sekskant, samt å vite hvor mange sider disse mangekantene har. Som antatt viste alle gruppene at de visste hvor mange sider hver av de tre mangekantene hadde. Dette vil derfor ikke være en matematisk forkunnskap som hindret elevene i å kunne gjennomføre oppgaven. I kapittel 4.3.3 kan jeg se hvordan vinkelverdien til den ytre vinkelen ble belyst av gruppe 1 og 2, men ikke gruppe 3. Gruppe 3 endte opp med å prøve og feile seg frem til riktig svar. Ettersom de gikk metodisk til verks og testet først med å øke verdien med små marginer for å så redusere verdien da vinkelen gikk feil vei, viste denne metoden seg til å være nokså effektiv i denne oppgaven. Gruppe 1 og 2 så tidlig at tallverdien i kommandoen *left( )* representerte vinkelverdien i mangekanten. Gruppe 1 tenkte seg tidlig frem til at verdien bør være 90 da det var snakk om å konstruere en firkant, og oppdaget at den ytre vinkelen til en regulær sekskant er 60 grader da de skulle prøve å resonnere seg frem til vinkelverdien i en femkant. Ved å delvis gjette seg frem og delvis regne ut landet de på at den ytre vinkelen til en regulær femkant er 72 grader. Gruppe 2 resonnererte seg frem til at de har 360 grader å fordele antall sider av mangekanten på, og etter å ha sjekket med meg om å fikk lov til å bruke kalkulator, regnet de ut alle vinkelverdiene. Denne oppgaven gir meg et bilde av elevenes kunnskapsbase knyttet til vinkelverdier og mangekanter.

Pólya (2004) introduserer oss for mange forskjellige typer problemløsningsstrategier, derfor er det interessant å trekke frem hvilke jeg observerte av elevene i denne studien. Den strategien som var aller mest fremtredende gjennom hele studien er prøve-og-feile-metoden. Elevene testet tallverdier og kommandoer både målrettet og helt i blinde for å prøve å nærme seg en løsning. I

kapittel 4.1 beskriver jeg fem forskjellige understrategier av det jeg kaller «Utforskning». Disse strategiene er de fem forskjellige strategiene jeg observerte elevene utførte da de prøvde seg frem. Strategien ved å låse alle variabler unntatt én og endre denne variabelen for å se hva som skjer, nevner Schoenfeld (2014) som en av de mer brukte problemløsningsstrategiene. Denne strategien kommer veldig tydelig frem gjennom hele datamaterialet og presenteres tydelig i kapitlene 4.1.1 og 4.1.2: «Endring av rekkefølge» og «Endring av tallverdier». Det er interessant å observere hvilke strategier som dukker opp, men uten ekstra kunnskap om elevenes tanker gir det oss svært lite informasjon om elevenes matematiske tenkning som ligger bakenfor disse spesifikke strategiene.

Elevenes metakognitive ferdigheter beskrives under elementet monitorering og kontroll (Schoenfeld, 2016). Disse ferdighetene bidrar med å hjelpe elevene å tenke over hva de gjør og hvorfor. Dette kom frem i enkelte situasjoner der elevene hadde gått seg vill i oppgaven og valgte å lese oppgaveteksten om igjen. Jeg kan derimot ikke si at elevene viste disse ferdighetene hver gang de leser oppgaveteksten om igjen. Da elevene stoppet opp og poengterte at de må gå tilbake, kan jeg være sikrere i beskrivelsen av slike ferdigheter. Flavell (1979) beskriver det som å ha kontroll over hva du kan og hvilke ferdigheter du har, som er vanskelig å beskrive med mindre elevene poengterer dette selv. Fra eksempelet om Anna som oppdaget koordinatsystemet i programmeringen kan jeg se hvordan hun viste kunnskap om hva hun kunne, og knyttet dette til problemet de stod ovenfor og nettopp har løst. Metakognisjon er studien av hvordan hjernen monitorerer seg selv og har kontroll over hvilke kunnskaper du besitter (Van Overschelde, 2008), slik hun viste i denne oppgaven.

I kapittel 4.1.5 ser vi oppgaven der målet er å få programmet til å tegne ti parallelle linjer. Her finner vi et tydelig eksempel på når elevene oppdaget at de var på feil vei og ønsket å bytte til et annet startpunkt. Gruppe 1 begynte med å videreutvikle programmet som tegner tre parallelle linjer istedenfor å lese oppgaven nøye og se at det finnes en oppskrift de kan bruke. De brukte en god del minutter på denne veien før Anna kom med kommentaren: «Vi bør ikke kanskje bruke oppskriften?». Dette førte til at de fjernet alle kodelinjene og begynte å oversette oppskriften fra pseudokode til Python, og noen få minutter senere kom de frem til løsningen. De begynte å se at den metoden de hadde valgt ble innviklet og Anna valgte å se tilbake til hva oppgaven spør om. Nok en gang viste Anna hvordan hun klarte å ha kontroll over egne tanker og trekke seg ut av

tankerekken og spørre gruppa om de skulle prøve en annen vinkling. Hun viste da tydelig og god monitorering og kontroll etter Schoenfelds (2016) beskrivelser.

Liljedahl (2016) presenterer Perkins' fire typer problem som krever en aha!-opplevelse for å løse. Fra min analyse finner jeg ingen av gruppene som begrenser rammene til et problem, og ville havnet i kategorien «smal utforskende dal»-problem. Det virket som om grunnen til dette var at oppgavene ikke var åpne nok til at elevene hadde behov for å lage egne rammer. I oppgave 1 stod gruppe 1 overfor et problem av kategori «uvitende platå»-problem og så ingen utvei som ledet til en løsning. Løsningen deres, slik som i oppgave 3 i avsnittet over, var å se tilbake på hva som stod i oppgaveteksten. Denne metoden for å løse et slikt problem er den Liljedahl (2016) selv presenterer. Problemet som elevene møtte i oppgaven beskrevet i avsnittet over var av typen «oase av falske løfter»-problem. Elevene brukte sin tidligere løsning og fortsatte på denne fordi de forstod hvordan og hvorfor den fungerte. Dette gjorde at de brukte mye tid på et forslag til en løsning som kun nesten fant frem til svaret. Ved å velge å se bort fra denne «fristende» metoden, så de tilbake på oppgaveteksten og fant en bedre vei. Det skal sies at det ikke er helt riktig å si at dette er et problem av typen «oase av falske løfter»-problem slik det fremlegges av Liljedahl (2016), da de faktisk hadde kommet frem til en løsning hvis de hadde fortsatt ned samme vei. De hadde brukt mye tid og mange flere kodelinjer, men de hadde oppdaget løsningen. Elevene opplevde derimot denne veien de har valgt som et slikt problem, og det er derfor interessant å ta med i betraktningen knyttet til elevenes monitorering og kontroll fra Schoenfelds (2016) beskrivelser.

Av figur 15 og presentert i kapittel 4.4.2 kan vi se at gruppe 2 brukte mye tid på utforskning i starten av oppgave 3. De testet ut mange forskjellige rekkefølger og tallverdier på kommandoer for å lete etter en løsning. Dette problemet beskriver jeg som kategorien «villmark av muligheter»-problem etter Liljedahls (2016) forklaringer. Elevene hadde for mange valgmuligheter og gjorde dermed problemet mye vanskeligere for seg selv. Det endte opp med at en elev spurte om de var helt på bærtur og jeg så meg nødt til å spesifisere at de ikke trengte å endre på hverken tall eller rekkefølge av eksisterende kode. I ettertid vil jeg argumentere for å ikke gi dem det tipset, men heller sett hvordan de hadde kommet seg ut av denne «villmarken» uten bistand. Valget mitt ved å gi dem veiledning var todelt: (1) Jeg ville at de skulle komme seg videre i prosessen på grunn av manglende tid igjen av observasjonen. (2) Oppgaven spesifiserer

ikke at de ikke skal endre på eksisterende kode. Grunn nummer 2 gjør at jeg ikke er sikker på om de hadde kommet seg videre i problemet, innenfor tidsrammen.

Elevenes egne oppfatninger kom hovedsakelig til syne i gruppe 3. Fra analysen av funnene presentert i kapittel 4.4.3 ser vi hvordan Kåre tidlig ga opp og det så ut som om han selv mente de ikke mestrer oppgaven. Han spurte meg om hjelp fem ganger i løpet av de første to minuttene av observasjonen, mens Mia prøvde så godt hun kunne å forstå oppgaveteksten. Dette kan tyde på at Kåre hadde en oppfatning av at han selv ikke mestret å løse problemer, knyttet til enten matematikk eller programmering. Dette stemmer overens med det Schoenfeld (2016) beskriver som en elev som har en oppfatning av å ikke mestre, og derfor gir tidlig opp. Hadde det ikke vært for Mias egen oppfatning som gjør at de fortsetter uten veiledning fra min side, er det en sannsynlighet for at de ikke hadde kommet i gang på egenhånd. Det er heller vanskelig å si i hvilken grad de andre gruppene hadde en bedre oppfatning av sin evne til å løse problemer, men ingen viste motløshet i samme grad som kom frem hos Kåre. Basert på at ingen av de andre elevene i datamaterialet viste noen tegn til å gi opp, kan jeg med stor sikkerhet si at flesteparten av elevene mest sannsynlig hadde en bedre oppfatning av relevante evner knyttet til oppgavesettet. Det må spesifiseres at Kåre og Mia var den gruppen med færrest antall elever og det kom derfor tydeligere frem slike følelser. Elevenes følelser er ikke denne studien laget for å plukke opp, og for å lete frem disse burde jeg ha eventuelt intervjuet elevene knyttet til dette.

Elevenes matematiske praksis så nokså forskjellig ut blant de forskjellige gruppene. Jeg får gjennom denne studien ingen direkte innsikt i hvordan en undervisningssituasjon i klassene elevene gikk i, ser ut. Derfor er det vanskelig å beskrive lærerens rolle knyttet til elevenes kunnskaper og evner, slik Schoenfeld (2016) og Shulman (1986) presenterer. Jeg kan derimot observere gjennom måten elevene angriper problemene at spesielt gruppe 1, gruppe 2 i mindre grad, har fått muligheten til å trene på å løse problemer sammen i en sosial kontekst. Dette kommer frem av måten Ida og Anna konstant kommuniserte, stilte hverandre spørsmål og utfordret tankerekker. Et godt eksempel på deres kommunikasjonsevne kom frem av utdraget fra transkripsjonen i kapittel 4.1.5. Her observerer vi hvordan de arbeidet i en retning de hadde bestemt seg for, frem til Anna utfordret og spurte om de heller skulle starte på ny og prøve en annen retning. Hos gruppe 2 var dette til stede, men ikke like tydelig. Gruppe 3 viste liten grad av slik kommunikasjon, men i løpet av økten kan vi se hvordan Mia og Kåre samtalte og

samarbeidet i mye større grad enn i starten av økten. Ettersom min rolle var å være observatør var det ikke et poeng at jeg skulle fungere som en veileder slik Schoenfeld (2016) argumenterer for. Dette skjedde derimot ved flere anledninger der elevene enten stilte meg direkte spørsmål fordi de satt fast og ikke kom videre, eller jeg observerte at tiden begynte å gå ut og de var såpass nær en løsning at de trengte kun et lite tips fra min side. Etter å ha fått veiledning og tips fra meg kunne jeg observere at elevene raskt fikk omstilt seg og forstod hva de måtte gjøre. Et eksempel på dette kan vi lese av utdraget fra transkripsjonen til gruppe 2 i kapittel 4.2.1, der elevene spurte om de er «på bærtur» og jeg poengterte at de ikke trengte å endre noen av tallverdiene eller rekkefølgen.

I prosessen med å analysere funnene mine observerte jeg at det ikke var vanskelig å kunne beskrive de tre første punktene i Pólyas (2004) fire steg i en problemløsningsprosess. Ved å undersøke tidslinjene til gruppene fra kapittel 4.4 kan jeg se at alle gruppene viser at de begynner med prøve å forstå problemet. Selv om de brukte ulik mengde tid på å utforske oppgaven, endte alle opp med å forstå hva det var de skulle gjøre før de gikk videre. Deretter kan vi se at alle i forskjellig grad la en plan for veien videre, det må derimot påpekes at mye av planleggingen de gjorde enten skjer underveis i utførelsen eller internt i eleven som utførte planen. Dette kan skyldes at de enten ikke var godt nok trent i problemløsning til å vite at det er viktig å ta seg tid til å legge en god plan, eller at de etter å ha forstått problemet raskt klarte å se en vei til løsningen og problemet ikke lenger var et problem for dem. Selv om de utviste forskjellig grad av planlegging, viste alle gruppene at de utførte det de hadde tenkt. Ettersom problemløsningsmodellen er syklisk ser vi at elevene ofte brukte tid på å gå tilbake og prøve å forstå problemet på nytt da planen ikke viste seg å stemme. Elevenes evne til å se tilbake og reflektere over det de nettopp har gjort, kommer svært lite tydelig frem. Pólya (2004) selv mener dette punktet ofte blir oversett, selv om det muligens er det viktigste hvis du skal ta med deg lærdom av prosessen videre inn i andre problemer. Basert på å observere hvor enkelt det er å plassere deler av elevenes prosesser inn i de forskjellige punktene til Pólya, sier jeg meg enig i at hans oversikt over en problemløsningsprosess er for vid og lite beskrivende til å ha en analytisk verdi slik Schoenfeld (2016) argumenterer.

Da både Schoenfeld (2016) og Resnick (1988) presenterer matematikken som en sosial aktivitet, er det interessant å se hvordan gruppedynamikken er forskjellig mellom gruppene. Gruppe 1

bestod av tre elever, gruppe 2 av fire elever og gruppe 3 av to elever. I gruppe 3 samtalte elevene på lik linje og ingen tok i stor grad mer plass enn den andre. I gruppe 1 og 2 var det tydelig én elev på hver gruppe som enten meldte seg ut av samtalen eller ble overkjørt. Anna og Ida i gruppe 1 var tydelige på å si hvordan de tenkte og resonnerte og utfordret hverandre, men Ola tok ikke like stor plass, selv om jentene prøvde å få han til å si hva han tenkte. I gruppe 2 var det Line som havnet på utsiden av samtalen, og måtte jobbe for å bli hørt av medelevene. Men det som er fellesnevner for alle gruppene var hvordan samtalene hjalp elevene å forstå hvordan de andre tenkte og dermed enten forstod de programmet eller plukket opp feil. Gruppe 1 og 2 viste at de hjalp hverandre å forstå programmet i kapittel 4.3.3 og gruppe 3 i kapittel 4.2.1. Gruppe 2, presentert i kapittel 4.1.2, fremviser samtaleevner sine når flere av elevene prøvde å løse oppgaven med å endre tilfeldige verdier før Tor poengterte at det er rekkefølgen de skulle endre på.

5.2 Elevenes algoritmiske tenkning

I denne delen av diskusjonen ønsker jeg å se på hvordan elevene viser sine algoritmiske tankeprosesser gjennom prosessen med å løse oppgavesettet. Ettersom algoritmisk tenkning både knyttes til problemløsning og programmering ser jeg det som veldig relevant å diskutere elevenes algoritmisk tankerekker (Bocconi et al., 2018; Gjøvik & Torkildsen, 2019; NOU 2020:2, 2020). Bocconi et al. (2018) deler inn algoritmisk tenkning inn i 5 deler som Gjøvik & Torkildsen (2019) utdyper, og oversetter til: abstraksjon, algoritmebehandling, generalisering, automatisering og dekomponering. I tillegg vil det være interessant å undersøke i hvilken grad Stenseth et al.s (2019) modell for sykluser i feilsøkingsprosessen, er til stede i elevenes problemløsningsprosess.

Elevene viser at de praktiserer abstraksjon ved flere anledninger i gjennomførelsen av oppgavesettet gjennom å vise at de klarer å se bort fra unødvendig informasjon. I oppgave 1 brukte gruppe 1 noen minutter med utforsking gjennom endring av tallverdier og utforskning av kommandoer i programmet før de leste at oppgavens kjerne er å bytte rekkefølgen på kodelinjene, slik vi leser i kapittel 4.2.1. Det kan være interessant å påpeke at elevene gjerne ikke visste at de hadde mulighet til å endre rekkefølgen på kodelinjene, før oppgave 1 ber dem om det. Ved denne oppdagelsen la de bort all annen informasjon og fokuserte kun på å stille opp kodelinjene i riktig rekkefølge. I datamaterialet er det ingen av gruppe 1 eller 2 som henviste til

kommandoen *forward( ),* og brukte ikke denne for å finne en løsning på problemet. Ingen av gruppene henviste til kommandoen med ord og det er derfor vanskelig å si om elevene aktivt valgte å ikke endre denne kommandoen fordi de mente den ikke var relevant for endring av figuren. Det som mest sannsynligvis har skjedd hos begge gruppene er én av to følgende hendelser: (1) De har først endret kommandoen *left( )* og har observert at denne endrer vinkelen, som er det de er ute etter. (2) De har lest navnene på kommandoene og tenkt seg til at en kommando som heter «fremover» ikke vil påvirke utseende til en mangekant mer enn i lengden på linjestykkene. Om dette viser at de aktivt velger bort å endre denne kommandoen kan jeg dermed ikke si med sikkerhet, ettersom dette ikke blir diskutert innad i gruppene.

I kapittelet om utforskning av kommandoer presenterer jeg hvordan gruppe 3 endret tallverdien i *forward( ),* Kåre poengterer at figuren kun ble større og Mia forklarte at det er *left( )* de må endre på. Her viser de evnen til å oppdage ny informasjon og bestemme om denne er relevant eller irrelevant for oppgaven de skal løse. Elevene valgte aktivt eller passivt å se bort fra informasjon de mente var unyttig eller irrelevant, etter Gjøvik og Torkildsens (2019) beskrivelse av abstraksjon. I tidslinjen til gruppe 1 ser jeg, i andre del av oppgave 3, et eksempel på at gruppe 1 ikke alltid viste denne evnen. De endte opp med å se bort fra relevant informasjon i form av oppskriften i oppgaven, og brukte dermed 4-5 minutter på å utprøve en løsning som de til slutt måtte gi opp.

Algoritmebehandling viser til hvordan du klarer å følge trinnvise instruksjoner i tillegg til å forklare hva du gjør (Gjøvik & Torkildsen, 2019). Alle oppgavene gir elevene mulighet til å følge en allerede eksisterende trinnvis plan, men dette kommer tydeligst frem i oppgave 3. Oppgaven gir dem både tre trinn de skal følge for å så gi dem en oppskrift de må oversette fra pseudokode til kodelinjer. Ettersom tidslinjene presentert i kapittel 4.4 viser at elevene brukte mye tid på å måtte lese oppgavene om igjen, ser det ut som om algoritmebehandling er en del av den algoritmiske tenkningen elevene ikke mestret helt. I tidslinjen til gruppe 1 i oppgave 2 finner vi det eneste eksempelet der en gruppe ikke måtte lese en oppgave om igjen for å klare å løse den. Denne elevgruppen brukte lengst tid på oppgave 1 av gruppene på grunn av at de ikke leste oppgaveteksten nøye nok. Jeg ser dermed at elevene lærte av denne feilen og leste nøye gjennom oppgaveteksten mens de skrev inn koden. Dette førte til at de forstod hva de måtte gjøre og utførte løsningen uten store problemer.

Da elevene forklarte hva de gjorde, viser analysen av funnene at medelevene forstod hvordan de tenkte. Dette bidro enten mot løsningen (gruppe 1 i kapittel 4.3.2) eller oppretting av misoppfatninger (gruppe 2 i kapittel 4.3.3). Etter å ha jobbet seg gjennom to oppgaver viste elevenes gjennomførelse av oppgave 3 at flere forstod sammenhengen mellom koordinater og kommandoene *setx( )* og *sety( ).* I kapittel 4.3.2 observerte jeg at Anna forstod denne sammenhengen etter å ha lest videre i oppgaveteksten. Dette gjorde at hun forstod hvordan linjenes bevegelser sammenfaller med tallverdiene i kommandoene. Hun forklarte dette for sine medelever og beskrev hvordan hun mente koden måtte se ut. Denne måten å beskrive en trinnvis prosess på er den tydeligste noen av gruppene kom frem til. Jeg fikk her anledning til å observere elementer av algoritmisk tenkning i elevgruppen der de forklarte sine egne tanker i tillegg til at de viste at de evnet å følge en trinnvis prosess. Dette er de to elementene som til sammen danner begrepet algoritmebehandling (Bocconi et al., 2018; Gjøvik & Torkildsen, 2019).

For å kunne generalisere gir Gjøvik & Torkildsen (2019) to forklaringer: gjenkjenne mønstre og sammenhenger, og lage regler som stemmer for andre oppgaver. I funnene mine finner jeg ingen eksempler på at elevene beskriver at de lager noen generelle regler. Dette kan forklares med at ingen av oppgavene verken krever eller etterspør dette av elevene. Dersom noen av elevene hadde vist evnen til å lage en generalisering, på eget initiativ, basert på oppgavesettet i denne studien, ville jeg blitt overrasket. Det vil være enklere å legge opp til en slik generalisering hvis oppgavene er bygget opp slik at det er det endelige målet. For eksempel ved å utforske en rettvinklet trekant med det mål om å komme frem til Pytagoras læresetning.

Derimot klarer flere av elevene å vise at de gjenkjenner sammenhenger på tvers av oppgavene. I kapittelet om elevenes gjetning kan vi observere et godt eksempel på forskjellene mellom gruppene på dette punktet, i første del av oppgave 3. Her kan vi se hvordan både gruppe 1 og gruppe 2 brukte informasjonen de plukket opp underveis i gjennomførelsen av de to foregående oppgavene. Begge gruppene nevnte forslaget om muligheten for at programmet tegner en bokstav, ettersom både oppgave 1 og 3 bruker de samme kommandoene, *setx( )* og *sety( ),* og målet med oppgave 1 var å tegne bokstaven «T». Gruppe 1 stod mellom å velge en figur eller en bokstav, men Ida påpekte at da de laget en figur brukte de kodelinjen *for i in range( ),* og de landet derfor på å gjette en bokstav.

Det å si at to programmer gjør det samme selv om de bruker de samme kommandoene er selvsagt ikke korrekt, selv om det viste seg å nesten stemme for oppgave 3. Ettersom dette oppgavesettet er elevenes første møte med Python har de ingen grunnlag for å si at dette ikke er tilfellet, og jeg definerer dette derfor som at de gjenkjenner en sammenheng mellom oppgavene. Det interessante ikke hvorvidt de har rett, men heller hvorfor de resonnerer slik de gjør. Fra gruppe 3 kan vi se at de leter etter en annen sammenheng, de henviser til det oppgaven i neste linje beskriver, parallelle linjer. Ettersom oppgaven var å lage tre parallelle linjer, foreslo Mia at programmet kanskje laget tre linjer som ikke var parallelle. De bruker ikke veldig mye tid på å diskutere dette, og går dermed heller videre etter ett forslag til en gjetning. Jeg tror ikke jeg kan si at det de gjør er å finne en sammenheng, mer enn å gi et gjett fordi oppgaven etterspør akkurat dette.

I kapittel 4.3 beskriver jeg elevenes evne til å klare å knytte sammenhenger mellom oppgavene og den matematiske bakgrunnskunnskapen de har. Dette kommer tydelig frem fra utdraget fra gruppe 1 i kapittelet om koordinatsystem, der Anna forklarer hvordan det grafiske grensesnittet er oppbygget som et koordinatsystem. Dette blir tydeliggjort av Tor i samme kapittel. Av generaliseringsbegrepets tre deler, introdusert av Gjøvik og Torkildsen (2019), observerer jeg at elevene til en viss grad viser evne til å se sammenhenger og gjenkjenne mønstre, men ikke konstruering av regler.

Ved å velge å løse et problem med hjelp av digitale hjelpemidler kan vi si at du automatiserer problemet (Gjøvik & Torkildsen, 2019). Ettersom rammene til dette oppgavesettet var at elevene skulle arbeide med programmeringsoppgaver i Python, ble dette valget tatt for elevene på forhånd. Det eksisterer flere forskjellige måter å automatisere oppgaver ved hjelp av programmering, der en vanlig metode er å bruke løkker. I oppgave 2 ble alle gruppene introdusert for en for-løkke, og forstod ganske umiddelbart hvilken effekt denne hadde. Denne løkken hadde de i tillegg bruk for i andre del av oppgave 3. Her fikk de beskjed i oppskriften at de skulle gjenta en pseudokode ti ganger, men ikke hvordan de skulle gjøre dette. Ettersom gruppe 3 ikke nådde denne delen av oppgavesettet, vil det være umulig for meg å si hvordan de hadde løst denne oppgaven.

Både gruppe 1 og 2 gjennomførte derimot denne oppgaven. Etter å ha designet et program for å tegne tre parallelle linjer, fikk de beskjed om at neste del av oppgaven var å tegne ti parallelle

linjer. Av kapittelet angående tidslinjen til gruppe 1 ser vi at de tidlig bestemte seg for å prøve å duplisere kodelinjene de allerede hadde laget for å se om dette konstruerte ti linjer. De brukte en del tid på å prøve å løse oppgaven på denne måten, før Anna spurte om de heller burde bruke oppskriften de var gitt. Det første forsøket deres ga dem 29 linjer med kode, men etter å ha fulgt oppskriften satt de igjen med 8 linjer med kode. Dette viser oss verdien av å kunne automatisere et problem, slik Gjøvik og Torkildsen (2019) fremhever. For å kunne gjøre det på denne måten må elevene forstå hvordan en for-løkke fungerer, og det var nettopp det som var poenget med å introdusere en slik løkke i oppgave 2. Hvis elevene allerede hadde hatt erfaring med Python ville jeg ikke brukt en oppgave som spesifiserte at de skulle få en handling til å skje et gitt antall ganger. Da ville det være mye mer hensiktsmessig å la elevene finne ut av dette på egenhånd. Ettersom elevene ikke hadde denne kompetansen så jeg det hensiktsmessig å gi dem denne informasjonen. Da kunne jeg se om de forstod sammenhengen mellom teksten *for i in range( )* og at en handling skjer et gitt antall ganger, indikert av tallet inne i parentesen. Stenseth et al. (2019) beskriver denne delen av programmeringsprosessen som «den kodefokuserte» syklusen, der elevene viser at de forstår koden og kan bruke denne forståelsen til å komme nærmere en løsning. I oppgave 3 viser elevene at de forstår kommandoer underveis i programmeringsprosessen og dermed løser én del av problemene de møter; elevenes manglende programmeringskunnskap.

For å dekomponere en oppgave, tar du fra hverandre problemet og ser på de individuelle delene hver for seg selv (Gjøvik & Torkildsen, 2019). Elevene i denne studien brukte mye tid på å utforske gjennom forskjellige metoder. Enkelte av metodene brukte elevene for å få mer informasjon om oppgavene (kapittel 4.1.5), mens andre var mer målrettet mot forståelse av kodens syntaks og kommandoenes funksjoner (kapitler 4.1.1-4.1.3). I oppgave 2 velger alle gruppene å målrettet fokusere på én og én kommando i koden. Alle starter med å endre *range(3)* til *range(4)* ettersom de tror, og har rett i, at denne kommandoen styrer antall streker som blir tegnet. Deretter ser vi gruppe 1 og 2 prøve å endre på kommandoen *left( )* og ser at den endrer verdien til vinklene. Gruppe 3 går en omvei om kommandoen *forward( )* men går fort videre. Fra dette kan vi tydelig se hvordan elevene dekomponerer koden ned til de tre kodelinjene som har en tallverdi knyttet til seg, og går enkeltvis til hver og én i rekkefølge for å avkrefte eller bekrefte dens relevans.

Stenseth et al. (2019) introduserer i sin artikkel en modell for å beskrive elevers prosess med å løse feil i et program. Modellen viser oss tre sykluser bestående av til sammen fem handlinger. Prøve-og-feile-syklusen de beskriver kommer veldig tydelig frem hos elevgruppene i denne studien. Fra kapittel 4.1 kan vi se hvordan de to første oppgavene starter med at elevene kjører programmet, ser hva som skjer og endrer programmet i en syklus. Oppgave 3 ber elevene om å gjette på resultatet av programmet og tvinger derfor elevene til å lage en hypotese, elevene beveger seg dermed inn i neste syklus. Gjennom oppgavene kan vi se hvordan elevene etter å ha vært i første syklus, beveger seg over i andre syklus når de begynner å forstå hvordan programmet er satt sammen og kommandoenes funksjoner. Tredje syklus Stenseth et al. (2019) presenterer krever, i tillegg til første og andre syklus, at elevene klarer å bruke matematiske forkunnskaper til å finne en løsning på oppgaven. Slik jeg presenterer i kapittel 4.3, kan vi se elevene vise at de behersket å bruke denne kunnskapen og dermed knyttet den til programmeringen de utførte. Både gruppe 1 og 2 viste at de beveget seg gradvis fra å prøve seg frem, til å forstå programmet på et stadium som gjorde at de kunne planlegge prosessen videre og til slutt se matematikken inn i programmet. Spesielt er det i oppgave 2 og 3 de viste dette. Kapittel 4.3.1 og 4.3.2 presenterer hvordan elevene først viste vinkelforståelse knyttet til konstruksjon av mangekanter og deretter viste forståelse av parallellitet. Til slutt ser vi at begge gruppene i litt forskjellig grad viste at de klarte å knytte det grafiske grensesnittet og tallverdier til henholdsvis koordinatsystem og koordinater. Gruppe 3 endte opp med å klare å bevege seg inn i tredje syklus i starten av oppgave 3 da de viste at de forstod parallelliteten. Utenom dette ble de værende i «den kodefokuserte» syklusen i resten av oppgavene (Stenseth et al., 2019).

5.3 Hva bruker elevene tiden på?

I kapittel 4.4 introduserte jeg en oversikt over alle gruppenes tidslinjer etter Schoenfelds (2016) modell, inndelt i seks deler. Oppgave 1 ser overraskende forskjellig ut mellom de tre gruppene. Gruppe 1 brukte mye tid på utforsking og verifiserte resultatet sitt etter de hadde funnet en løsning, de er i tillegg den gruppen som brukte mest tid på denne oppgaven. Gruppe 2 brukte tid på å gå frem og tilbake mellom å utforske og å lese oppgaveteksten om igjen, og er den gruppen som brukte kortest tid på denne oppgaven. Gruppe 3 brukte mer av tiden på å lese og skrive av oppgaven enn å utforske, men løste oppgaven raskere enn gruppe 1. Deler av grunnen til at denne oppgaven ser såpass forskjellig ut mellom de tre gruppene beskrives ganske tydelig da det

gikk opp for de enkelte gruppene at det eneste de skulle gjøre var å endre på rekkefølgen. Den første gruppen brukte så lang tid, fordi de valgte å ikke gi opp med å utforske og fortsatte å lete etter en løsning helt til Anna leste hva oppgaven spurte om. Den andre gruppen fikk enkelte feilkoder i innskriften av koden og leste dermed oppgaven nøyere mens de feilsøkte. Mia i den tredje gruppen leste oppgaven veldig nøye mens hun skrev av og fikk dermed med seg hva som stod.

Oppgave 2 går nokså fort for både gruppe 1 og 2, men tidsbruken deres ser nokså forskjellig ut. Gruppe 1 leste raskt gjennom oppgaven før de brukte tid på å analysere hva oppgaven faktisk spurte etter og løste dette deretter. Gruppe 2 fikk nok en gang en feilkode og måtte lese oppgaven om igjen før de fant en løsning. Gruppe 3 brukte over dobbelt så lang tid på denne oppgaven som de andre gruppene. Mer eller mindre halvparten av tiden gikk til lesing av oppgavetekst og den andre halvparten gikk til utforsking av programmet i tillegg til en rask løsningssekvens. Den tredje gruppen så ikke sammenhengen mellom vinkelverdier og programmet og brukte derfor mye tid på nokså blind gjetting av verdier.

Den tredje oppgaven deles i to deler, der en del består av å lage tre parallelle linjer og den andre å lage ti parallelle linjer. Gruppe 1 brukte i underkant av fem minutter på å utforske programmet før de leste oppgaven på ny og gjorde en kort analyse. De kjørte programmet med nye endringer og klarte å finne en løsning. Deretter valgte de å ta i bruk kun halvparten av oppgaveteksten og prøvde å finne sin egen løsning ved hjelp av første del av programmet. Dette gjorde at de brukte rundt 5 minutter på noe de selv indikerte at var et blindspor, før de valgte å ta avstand fra denne metoden og fulgte oppskriften videre. De avsluttet programmet ved å gjøre en rask verifisering av hvorfor og hvordan programmet fungerte.

Gruppe 2 leste oppgaveteksten noen ganger og gjorde en rask analyse av hva den spurte om, før de brukte over ti minutter på å prøve å finne en løsning. Mye av tiden gikk ut på å utforske og forstå kommandoene *penup( )* og *pendown( )*. Etter å ha funnet en tilnærmet løsning, samtalte de sammen og verifiserte løsningen, før de deretter leste videre. De brukte 3-4 minutter på å prøve å forstå neste del av oppgaven, før Line tok over kontrollen på datamaskinen og skrev inn det hun mente var en løsning ifølge oppskriften. Deretter gikk de raskt gjennom hvorfor denne løsningen stemte.

Den tredje gruppen nådde ikke å gjøre andre del av oppgaven, men ble akkurat ferdig med å få programmet til å tegne tre parallelle linjer. Etter å ha lest oppgaveteksten prøvde de seg på en rask løsning, som ikke ga resultater. De prøvde å forstå oppgaveteksten gjennom en nøyere analyse av hva den spurte om, før de brukte rundt 8 minutter på å utforske kommandoer, og endre tallverdier og rekkefølge på kodelinjene. Til slutt fant de en løsning som gjorde at de nesten kom i mål og jeg valgte å gi dem et lite tips for å rette opp i en siste tallverdi.

Av tabell 3 i kapittel 4.4 kan vi se en oversikt over tidsbruken til elevene. Rett under halvparten av tiden brukt i gruppe 2 og 3 gikk til utforsking, mens kun en tredjedel av tiden ble brukt til det samme hos gruppe 1. Gruppe 1 brukte over 10 % mer av tiden på å analysere oppgavene enn de to andre gruppene, som kan forklare hvorfor de brukte en del mindre tid på utforskning. Den tiden gruppe 1 brukte på analyse, viser igjen hos de andre gruppene på tid brukt på å lese og lese om igjen oppgavene. Tabellen viser oss at tiden brukt på å verifisere oppgavene var tilnærmet likt hos gruppe 1 og 2, med et 0.5 % skille. Gruppe 3 brukte ingen tid på verifisering, som kan begrunnes med elevenes faglige oppfatninger i gruppa. I tillegg hadde denne gruppen to deltakere, og dermed færre samtalepartnere. Schoenfeld (2016) og Resnick (1988) argumenterer for det sosiale aspektet ved matematikken, som kan være en grunn for denne gruppens prestasjoner. Derimot var det hovedsakelig Anna og Ida som deltok i gruppe 1, der Ole sjeldent deltok aktivt i prosessen. Gruppe 1 brukte også en del mer tid på planlegging og implementering enn de to andre gruppene, da de var mer samkjørte og sammen la planer for hvordan de skulle løse oppgavene og gjennomførte disse.

Tabell 2 i kapittel 4.1.2 beskriver hvor mye tid elevene bruker på å samtale rundt endring av tallverdier. I gjennomsnitt gikk rundt 10 % av tiden brukt på samtaler til å diskutere endring av tallverdier. Dette er ikke en verdi jeg tror stemmer overens med flertallet, og det hadde vært interessant å se denne verdien gjennom et mye større utvalg. Ettersom det å endre tallverdier er en ganske naturlig problemløsningsmetode for å utforske kommandoene i programmet, er det veldig interessant at denne problemløsningsmetodens tidsbruk er såpass lav.

6 KONKLUSJON

I dette kapittelet vil jeg prøve å svare på forskningsspørsmålet mitt med utgangspunkt i funnene fra datamaterialet og diskusjonen i kapittel 5. På bakgrunn av mengden timer observert og antall elever i utvalget, kan jeg ikke fremlegge noen generelle slutninger basert på denne studien. Jeg kan likevel peke på hvilke deler av elevenes matematiske- og algoritmiske tenkning som kommer til uttrykk i elevenes møte med tekstbasert programmering i matematikk. Dette kan være med på å styrke grunnlaget for å bruke programmering til å fostre problemløsningsferdigheter blant elevene. Forskningsspørsmålet mitt i denne studien var:

Hvilke problemløsningsprosesser kommer til uttrykk i elevers introduksjon til tekstbasert programmering i matematikk på ungdomstrinnet?

Tanken var først å prøve å finne hvilke problemløsningsstrategier elevene brukte, men jeg endte opp med at det var mer interessant å lete etter hvilke prosesser elevene står i, i møte med ukjent programmering. Det var derfor interessant å undersøke hvilke handlinger elevene brukte tiden på i prosessen med å arbeide med programmeringen. For å besvare dette spørsmålet valgte jeg å undersøke elevenes matematiske- og algoritmiske tenkning.

Matematisk og algoritmisk tenkning går hånd-i-hånd med programmering i skolen. De er begge meget relevante knyttet til problemløsning, og vil være essensielle å trene på om du ønsker å bli en bedre problemløser. Det var derfor veldig interessant å se hvilke tankeprosesser som kom til uttrykk blant elevene for å se hvordan programmering kan bidra til økt problemløsningsferdigheter i matematikk. Elevenes matematiske tenkning kommer tydelig til uttrykk i gjennomførelsen av programmeringsoppgavene. De tre gruppene viste i forskjellig grad hvordan de tenkte og samarbeidet gjennom utprøvinger og utforskning av oppgavene. Elementene som deler inn matematisk tenkning, kommer til syne gjennom elevenes samtaler og programlinjene de produserte. Resultatene fra denne studien viser at når elevgruppene ble introdusert for tekstbasert programmering (Python), kan jeg observere at elevene utviste matematiske tankeganger. Dette vil si at når de ble utfordret med ny og ukjente programmeringsoppgaver, fikk de mulighet til å trene på deres matematiske tenkning og problemløsningsferdigheter.

Studiens resultater viser i tillegg hvordan elevenes algoritmiske tenkning kommer til uttrykk. De algoritmiske tankeprosessene knytter seg tydeligere opp mot programmeringsferdighetene enn de matematiske og er derfor enda enklere å observere. Elevene viste hvordan de på eget initiativ gikk frem for å finne ut hvordan de skulle angripe og løse ukjente programmeringsoppgaver sammen som gruppe. Gjennom samtalene mellom elevene observerte jeg hvordan de sammen løste problemene som oppstod. Studiens resultater viser at elevene demonstrerte tydelige tegn til algoritmisk tenkning underveis i problemløsningsprosessen gjennom å arbeide med ukjent tekstbasert programmering.

Ettersom storparten av problemene elevene møtte i oppgavesettet var knyttet til elevenes kodeforståelse, var det interessant å se hva de brukte tiden på i problemløsningsprosessen. Storparten av tiden brukte elevene på forskjellige former for utforskning av koden og oppgaven, der et hyppig eksempel var å endre tallverdier for å se utfallet. Det kommer tydelig frem av resultatene at elevene brukte mye tid på å lese oppgaven om igjen flere ganger for å prøve å forstå hva de måtte gjøre.

Av studiens resultater kan jeg anslå at ved å introdusere tekstbasert programmering på ungdomstrinnet, kan elevenes matematiske- og algoritmiske tankeprosesser bli utfordret og øvd på. Dette vil ifølge studiens teoretiske rammeverk bidra til at elevene får øve på og utfordret sine problemløsningsferdigheter.

Denne studien baseres på et datamateriale med begrenset utvalg over en kort periode. Det vil derfor være interessant å se om funnene fra denne studien kan generaliseres for en større populasjon, eller om den ikke samsvarer med flertallet. Videre kan man undersøke hvorvidt min konklusjon stemmer for programmeringsspråk elevene allerede har erfaring med fra før, eller om elevenes matematiske- og algoritmiske tenkning kommer tydeligst frem i introduksjonen av programmeringen. Ettersom selve kodeforståelsen er hovedelementet i problemene til elevene, kan man videre undersøke om elever med lav matematisk motivasjon kan få økt motivasjon gjennom tekstbasert programmering i matematikk. Avslutningsvis vil det være interessant å undersøke elever med et sterkere fundament i programmering som blir utfordret med mer åpne problemløsningsoppgaver, og se om resultatene vil være sammenlignbare med denne studien.

7 LITTERATURLISTE

Bailey, H. (u.å.). *The OBS Project Contributors*. Open Broadcasting Software. Hentet 21. mai 2023 fra https://obsproject.com/

Balanskat, A., & Engelhardt, K. (2014). *Computing our future: Computer programming and coding* (Priorities, school curricula and initiatives across Europe. European Schoolnet, Issue. 10.13140/RG.2.1.5029.9048

Begle, E. G. (1979). *Critical Variables in Mathematics Education: Findings from a Survey of the Empirical Literature*. Mathematical Association of America.

Beiler, I. R., Brevik, L. M., & Christiansen, T. (2021). Skjermopptak som forskningsmetode i og utenfor klasserommet. In E. I. Andersson-Bakken & C. Pedersen Dalland (Eds.), *Metoder i klasseromsforskning: forskningsdesign, datainnsamling og analyse* (pp. 239-260). Universitetsforlaget.

Bocconi, S., Chioccariello, A., Earp, J., & Group, N. B. S. (2018). *The Nordic approach to introducing Computational Thinking and programming in compulsory education*. https://doi.org/10.17471/54007

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

Britannica, T. E. o. E. (2023). assembly language. T. E. o. E. Britannica (Ed.), *Encyclopedia Britannica*.

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *1989*, *18*(1), 32-42.

Bryman, A. (2016). *Social research methods* (5 ed.). Oxford university press.

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

Dalland, O. (2017). *Metode og oppgaveskriving* (6 ed.). Gyldendal Norsk Forlag AS.

Flavell, J. H. (1979). Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist*, *34*(10), 906.

Flyvbjerg, B. (2006). Five misunderstandings about case-study research. *Qualitative inquiry*, *12*(2), 219-245. https://doi.org/10.1177/1077800405284363

Gjøvik, Ø., & Torkildsen, H. A. (2019). Algoritmisk tenkning. *Tangenten–tidsskrift for matematikkundervisning*, *30*(3), 31-37. http://tangenten.no/wp-content/uploads/2021/12/tangenten-2-2019-Stenseth-et-al.pdf

Grandell, L., Peltomaki, M., Back, R.-J., & Salakoski, T. (2006). Why complicate things?: introducing programming in high school using Python. ACM International Conference Proceeding Series,

Grønmo, S. (2020, 14. mai). *Case-studie*. Store norske leksikon. https://snl.no/case-studie

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43. https://doi.org/10.3102/0013189X12463051

Halmos, P. R. (1980). The heart of mathematics. *The American Mathematical Monthly*, *87*(7), 519-–524. https://doi.org/https://doi.org/10.2307/2321415

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

Harboe, T. (2006). *Kvalitative og kvantitative metoder. Indføring I samfundsvidenskabelig metode* (4 ed.). Forlaget Samfundslitteratur.

Harling, K. (2012). An overview of case study. *SSRN*. https://doi.org/https://dx.doi.org/10.2139/ssrn.2141476

Heale, R., & Twycross, A. (2018). What is a case study? *Evidence-based Nursing*, *21*(1), 7-8. https://doi.org/10.1136/eb-2017-102845

Hemmendinger, D. (2019). Logo. T. E. o. E. Britannica (Ed.), *Encyclopedia Britannica*.

Henderson, H. (2009). *Encyclopedia of computer science and technology* (rev. utg. ed.). Infobase Publishing.

Inzlicht, M., Werner, K. M., Briskin, J. L., & Roberts, B. W. (2021). Integrating models of self-regulation. *Annual review of psychology*, *72*, 319-345.

Jørgensen, K. E., & Dahl, S. A. (2021). *Kaares kokebok i programmering: for matematikk i ungdomsskolen og VG1* (2 ed.). Jørgensen Matematiske Ressurser.

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012). Understanding computational thinking before Programming: Developing guidelines for the design. *Developments in Current Game-Based Learning Design and Deployment*, 316. https://doi.org/10.4018/978-1-4666-1864-0.ch023

King, N. (2004). Using Templates in the Thematic Analysis of Text. In C. Cassell & G. Symon (Eds.), *Essential guide to qualitative methods in organizational research* (pp. 257 - 270). Sage Pubicationss Ltd. https://doi.org/10.4135/9781446280119.n21

Kunnskapsdepartementet. (2019). *Læreplan i matematikk 1.-10. trinn (MAT01-05)*. Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2020 Retrieved from https://www.udir.no/lk20/mat01-05?lang=nob

Kvale, S., & Brinkmann, S. (2015). *Det kvalitative forskningsintervju* (3 ed.). Gyldendal akademisk.

Lampert, M. (1990). When the problem is not the question and the solution is not the answer: Mathematical knowing and teaching. *American educational research journal*, *27*(1), 29-63. https://doi.org/https://doi.org/10.2307/1163068

Lerdal, A. (2009). Metodekapitlet. *Sykepleien Forskning*, *4*(3), 239-241.

Lester Jr, F. K., & Cai, J. (2016). Can mathematical problem solving be taught? Preliminary answers from 30 years of research. *Posing and solving mathematical problems: Advances and new perspectives*, 117-135.

Liljedahl, P., Santos-Trigo, M., Malaspina, U., & Bruder, R. (2016). *Problem solving in mathematics education*. Springer Cham.

Mason, J. (2016). *When is a problem…?"When" is actually the problem!* (P. Felmer, E. Pehkonen, & J. Kilpatrick, Eds.). Springer.

Mason, J., & Davis, J. (1991). *Fostering and sustaining mathematics thinking through problem solving*. Deakin University.

Meld. St. nr. 28 (2015-2016). *Fag – Fordypning – Forståelse — En fornyelse av Kunnskapsløftet*. Retrieved from https://www.regjeringen.no/no/dokumenter/meld.-st.-28-20152016/id2483955/

Mladenović, M., Krpan, D., & Mladenović, S. (2016, 04.-06.07.2016). *Introducing programming to elementary students novices by using game development in Python and Scratch* International Conference on Education and New Learning Technologies, Barcelona, Spain. 10.21125/edulearn.2016.1323

NESH, T. (2022). Guidelines for Research Ethics in the Social Sciences and the Humanities. The Norwegian National Research Ethics Committees.

Nosrati, M. (2011). Python: An appropriate language for real world programming. *World Applied Programming*, *1*(2), 110-117. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a7c1dad0e2abcb5c54a37f235e83c3f2e227c0f3

NOU 2015:8. (2015). *Fremtidens skole: Fornyelse av fag og kompetanser*. Regjeringen Retrieved from https://www.regjeringen.no/contentassets/da148fec8c4a4ab88daa8b677a700292/no/pdfs/nou201520150008000dddpdfs.pdf

NOU 2020:2. (2020). *Fremtidige kompetansebehov III: Læring og kompetanse i alle ledd*. Regjeringen Retrieved from https://www.regjeringen.no/contentassets/053481d65fb845be9a2b1674c35d6d14/no/pdfs/nou202020200002000dddpdfs.pdf

Nowell, L. S., Norris, J. M., White, D. E., & Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria. *International journal of qualitative methods*, *16*(1), 1-13. https://doi.org/https://doi.org/10.1177/1609406917733847

Lov om grunnskolen og den vidaregåande opplæringa, (1998). https://lovdata.no/dokument/NL/lov/1998-07-17-61?q=oppl%C3%A6ringsloven

Perens, B. (1999). The open source definition. In DiBona, Chris, S. Ockman, & M. Stone (Eds.), *Open sources: voices from the open source revolution* (pp. 171-188). O`Reilly Media.

Polya, G. (2004). *How to solve it: A new aspect of mathematical method* (2 ed.). Princeton university press.

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

Pratt, V. R. (1973). Top down operator precedence. Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages,

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional science*, *45*(5), 583-602. https://doi.org/ 10.1007/s11251-017-9421-5

Resnick, L. B. (1988). Treating mathematics as an ill-structured discipline.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., & Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67.

Schoenfeld, A. H. (2014). *Mathematical problem solving*. Elsevier.

Schoenfeld, A. H. (2016). Learning to think mathematically: Problem solving, metacognition, and sense making in mathematics (Reprint). *Journal of education*, *196*(2), 1-38. https://doi.org/https://doi.org/10.1177/002205741619600202

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational researcher*, *15*(2), 4-14.

Sikt. (u.å.). *Barnehage- og skoleforskning*. Hentet 20. mai 2023 fra https://sikt.no/personvernhandbok-forskning/barnehage-og-skoleforskning

Solomon, C., Harvey, B., Kahn, K., Lieberman, H., Miller, M. L., Minsky, M., Papert, A., & Silverman, B. (2020). History of logo. *Proceedings of the ACM on Programming Languages*, *4*(HOPL), 1-66. https://doi.org/https://doi.org/10.1145/3386329

Stenseth, B., Kaufmann, O. T., & Forsström, S. (2019). Programmering og matematikk. *Tangenten–tidsskrift for matematikkundervisning*, *30*(2), 7-12. http://tangenten.no/wp-content/uploads/2021/12/tangenten-2-2019-Stenseth-et-al.pdf

Stigler, J. W., & Hiebert, J. (2009). *The teaching gap: Best ideas from the world's teachers for improving education in the classroom*. Simon and Schuster.

Tjora, A. H., & Tjora, A. (2021). Kvalitative forskningsmetoder i praksis (4. utgave. utg.). *Oslo: Gyldendal*.

UiS. (2022, 21. november 2022). *Om NVivo*. UiS. https://www.uis.no/nb/bibliotek/nvivo

Utdanningsdirektoratet. (2019, 27. mars 2019). *Algoritmisk tenkning*. Utdanningsdirektoratet. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Van Overschelde, J. P. (2008). Metacognition: Knowing about knowing. In J. Dunlosky & R. A. Bjork (Eds.), *Handbook of metamemory and memory* (pp. 47-72). Taylor & Francis.

Van Rossum, G. (2007). Python Programming Language. *USENIX annual technical conference*, *41*(1), 1-36. http://kelas-karyawan-bali.kurikulum.org/IT/en/2420-2301/Python_3721_kelas-karyawan-bali-kurikulumngetesumum.html

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

VEDLEGG

Vedlegg A – Informasjonsskriv + samtykkeerklæring for foreldre/foresatte

Vil du delta i forskningsprosjektet

«Programmering i matematikk»?

Dette er et spørsmål til om deltakelse i et forskningsprosjekt hvor formålet er å bedre forstå hvordan elever løser problemer de møter gjennom programmering i matematikkundervisning i grunnskolen. Du får dette informasjonsskrivet på vegne av ditt barn. I dette skrivet gir jeg informasjon om målene for prosjektet og hva deltakelse vil innebære for ditt barn.

Formål

Programmering er et nylig lagt til i læreplanen i matematikk. Dette er et tema som både lærere og elever ikke nødvendigvis sitter med forhåndskunnskaper i. Programmering består av å lese, tolke, designe, teste og feilsøke programmer som skal kjøres av en datamaskin. Elevene vil gjennom å møte programmering få muligheten til å se matematikken i et nytt lys. Det vil derfor være interessant gjennom denne studien å få et innblikk i prosessene som skjer hos elevene i møte med programmering i matematikkundervisning.

Prosjektet vil ledes av masterstudent ved Universitetet i Stavanger, Lars Midttun Vestbøstad.

Hvem er ansvarlig for forskningsprosjektet?

Universitetet i Stavanger er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?

Du får denne henvendelsen om å delta fordi du er forelder/foresatt til en elev ved en av skolene som er invitert til å delta i prosjektet.

Hva innebærer det å delta?

Prosjektet som helhet har en varighet på ett år, og jeg vil i løpet av dette året besøke én skole. For ditt barn innebærer deltakelse i prosjektet først og fremst at Jeg vil observere (samt gjøre lydopptak) fra gitte gruppeoppgaver én gang. Dersom du ikke ønsker at ditt barn skal delta, kan du skrive dette i samtykkeskrivet. Jeg vil da sørge for at ditt barn ikke blir tatt ut for å delta i gruppeoppgavene. Opptakene vil kun danne utgangspunkt for en skriftliggjøring (transkripsjon) av det som skjer og blir sagt i undervisningen, og det er de anonymiserte transkripsjonene som vil bli analysert og eventuelt gjengitt.

I tillegg til gruppeobservasjoner vil jeg invitere noen elever til å være med på et gruppeintervju (ca. 15–20 minutter) sammen med 1–2 andre elever fra klassen.

Foreldre/foresatte kan få se spørreskjema og intervjuguide (for de som har barn som har sagt seg villige til å delta i intervju) på forhånd. Dette kan ordnes ved å ta kontakt med prosjektleder: Lars Midttun Vestbøstad.

I elevintervjuet vil elevene bli bedt om å svare på/diskutere noen utvalgte matematikkoppgaver. Når jeg senere intervjuer lærer, vil jeg be lærer om å forklare hvordan de tolker slike typer svar (elevsvarene vil da anonymiseres).

Det er frivillig å delta

Det er frivillig å delta i prosjektet. Hvis ditt barn velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle personopplysninger om ditt barn vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg eller ditt barn hvis de ikke vil delta

eller senere velger å trekke seg. Dersom det blir for mange elever i klassen som ikke ønsker å delta, vil jeg finne en annen klasse å observere.

Ditt personvern – hvordan jeg oppbevarer og bruker dine opplysninger

Jeg vil bare bruke opplysningene om ditt barn til formålene jeg har fortalt om i dette skrivet. Jeg behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

- Lydopptak vil kun være tilgjengelig for meg så lenge prosjektet varer.
- Opptakene vil lagres sikkert på krypterte lagringsløsninger, og opptakene vil transkriberes og anonymiseres. Alle navn vil erstattes med fiktive navn, og jeg vil sørge for at kontaktopplysninger lagres sikkert adskilt fra øvrige data.

I publikasjoner fra prosjektet vil alle opplysninger anonymiseres, og jeg vil sørge for at det ikke blir gitt opplysninger som gjør at deltakerne kan gjenkjennes.

Hva skjer med opplysningene dine når jeg avslutter forskningsprosjektet?

Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er *31. juni 2023.* Da vil alle lydopptak slettes, og jeg vil kunne oppbevare anonymiserte transkripsjoner og anonyme svar på spørreskjema.

Dine rettigheter

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

- innsyn i hvilke personopplysninger som er registrert om ditt barn, og å få utlevert en kopi av opplysningene,
- å få rettet personopplysninger om ditt barn,
- å få slettet personopplysninger om ditt barn, og

- å sende klage til Datatilsynet om behandlingen av ditt barns personopplysninger.

Hva gir meg rett til å behandle personopplysninger om ditt barn?

Jeg behandler opplysninger om ditt barn basert på ditt samtykke.

På oppdrag fra *Universitetet i Stavanger* har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?

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

- Masterstudent, Lars Midttun Vestbøstad (tlf.: 95 42 23 71, e-post: 250575@uis.no)
- Prosjektansvarlig, Sean Peter Kåss Martin (tlf.: 91 12 22 79, e-post: sean.martin@uis.no).

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

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

Med vennlig hilsen

Lars Midttun Vestbøstad

(Masterstudent)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *Programmering i matematikk*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ at mitt barn blir observert (ved hjelp av lydopptak) i én gruppeoppgave.
- ☐ at mitt barn kan delta i *gruppeintervju*

Jeg samtykker til at opplysninger om mitt barn behandles frem til prosjektet er avsluttet

---

(Signert av foreldre/foresatte på vegne av elev, dato)

Vedlegg B – Informasjonsskriv + samtykkeerklæring for lærer

Vil du delta i forskningsprosjektet

«Studere matematikkundervisning»?

Dette er et spørsmål til om deltakelse i et forskningsprosjekt hvor formålet er å bedre forstå hvordan elever løser problemer de møter gjennom programmering i matematikkundervisning i grunnskolen. Du får dette informasjonsskrivet på vegne av ditt barn. I dette skrivet gir jeg informasjon om målene for prosjektet og hva deltakelse vil innebære for ditt barn.

Formål

Programmering er et nylig lagt til i læreplanen i matematikk. Dette er et tema som både lærere og elever ikke nødvendigvis sitter med forhåndskunnskaper i. Programmering består av å lese, tolke, designe, teste og feilsøke programmer som skal kjøres av en datamaskin. Elevene vil gjennom å møte programmering få muligheten til å se matematikken i et nytt lys. Det vil derfor være interessant gjennom denne studien å få et innblikk i prosessene som skjer hos elevene i møte med programmering i matematikkundervisning.

Prosjektet vil ledes av masterstudent ved Universitetet i Stavanger, Lars Midttun Vestbøstad.

Hvem er ansvarlig for forskningsprosjektet?

Universitetet i Stavanger er ansvarlig for prosjektet..

Hvorfor får du spørsmål om å delta?

Du får spørsmål om å delta fordi du underviser i matematikk ved en av grunnskolene i distriktet.

Hva innebærer det for deg å delta?

Prosjektet som helhet har en varighet på ett år, og jeg vil i løpet av dette året besøke én skole. For din del innebærer deltakelse i prosjektet først og fremst at jeg vil gjennomføre 1–2 intervjuer med deg (hvert intervju vil ha en varighet på maksimalt en time). I tillegg vil jeg invitere noen elever fra klassen din til å være med på et gruppeintervju (ca. 15–20 minutter) sammen med 1–2 andre elever fra klassen. Jeg håper at du kan være behjelpelig med å velge ut elever til gruppeintervju og gruppeoppgaver.

Jeg vil sende ut informasjonsskriv med samtykkeskjema til foreldrene i forkant, og foreldre kan også få se spørreskjema og intervjuguide (for de som har barn som har sagt seg villige til å delta i intervju) på forhånd. Dette kan ordnes ved å ta kontakt med prosjektleder: Lars Midttun Vestbøstad.

Det er frivillig å delta

Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan jeg oppbevarer og bruker dine opplysninger

Jeg vil bare bruke opplysningene om deg til formålene jeg har fortalt om i dette skrivet. Jeg behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

- Lydopptak vil kun være tilgjengelig for meg så lenge prosjektet varer.

- Opptakene vil lagres sikkert på krypterte lagringsløsninger, og opptakene vil transkriberes og anonymiseres. Alle navn vil erstattes med fiktive navn, og jeg vil sørge for at kontaktopplysninger lagres sikkert adskilt fra øvrige data.

I publikasjoner fra prosjektet vil alle opplysninger anonymiseres, og jeg vil sørge for at det ikke blir gitt opplysninger som gjør at deltakerne kan gjenkjennes.

Hva skjer med opplysningene dine når jeg avslutter forskningsprosjektet?

Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er *31. juni 2023.* Da vil alle lydopptak slettes, og jeg vil kunne oppbevare anonymiserte transkripsjoner fra intervjuene og anonyme spørreskjema.

Dine rettigheter

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

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

Hva gir meg rett til å behandle personopplysninger om deg?

Jeg behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra *Universitetet i Stavanger* har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?

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

- Masterstudent, Lars Midttun Vestbøstad (tlf.: 95 42 23 71, e-post: 250575@uis.no)
- Prosjektansvarlig, Sean Peter Kåss Martin (tlf.: 91 12 22 79, e-post: sean.martin@uis.no).

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

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

Med vennlig hilsen

Lars Midttun Vestbøstad

(Masterstudent)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *Det komplekse undervisningsarbeidet i matematikk*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta i *intervju*

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

---

(Signert av prosjektdeltaker, dato)

Vedlegg C – Godkjenning fra Sikt

Sikt

Meldeskjema / Programmering og problemløsning i matematikk / Vurdering

Vurdering av behandling av personopplysninger

Referansenummer
122963

Vurderingstype
Standard

Dato
13.01.2023

Prosjekttittel
Programmering og problemløsning i matematikk

Behandlingsansvarlig institusjon
Universitetet i Stavanger / Fakultet for utdanningsvitenskap og humaniora / Institutt for grunnskolelærerutdanning, idrett og spesialpedagogikk

Prosjektansvarlig
Sean Peter Kåss Martin

Student
Lars Midttun Vestbøstad

Prosjektperiode
01.11.2022 - 30.06.2023

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 30.06.2023.

Meldeskjema

Kommentar
OM VURDERINGEN

Sikt 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.

TYPE PERSONOPPLYSNINGER
Prosjektet vil behandle alminnelige personopplysninger samlet inn fra to utvalg. Utvalg 1 er lærere på ungdomstrinnet, mens utvalg 2 er elever på ungdomstrinnet.

UTDYPENDE OM LOVLIG GRUNNLAG
Prosjektet vil innhente samtykke fra foresatte til behandlingen av personopplysninger om elevene fra utvalg 2. 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.

TAUSHETSPLIKT
Utvalg 1 har yrkesmessig taushetsplikt som lærere. De kan ikke dele taushetsbelagte opplysninger med forskningsprosjektet. Vi anbefaler at du minner dem på taushetsplikten.
Merk også at det ofte ikke er nok å kun utelate navn ved omtale av elever. Vær oppmerksom og forsiktig ved bruk av eksempler og bakgrunnsopplysninger som tid, sted, kjønn og alder som sammen kan gjøre enkeltelever gjenkjennelige.

FØLG DIN INSTITUSJONS RETNINGSLINJER
Vi har vurdert at du har lovlig grunnlag til å behandle personopplysningene, men husk at det er institusjonen du er ansatt/student ved som avgjør hvilke databehandlere du kan bruke og hvordan du må lagre og sikre data i ditt prosjekt. Husk å bruke leverandører som din institusjon har avtale med (f.eks. ved skylagring, nettspørreskjema, videosamtale el.)

1/2

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).

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. Se våre nettsider om hvilke endringer du må melde: sikt.no/melde-endringar-i-meldeskjema

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

Lykke til med prosjektet!

Vedlegg D – Oppgavesett

Oppgave 1

```
from turtle import *

setx(45)
sety(100)
setx(-45)
sety(100)
hideturtle()
```

Skriv av koden ovenfor. Plasser kodelinjene i riktig rekkefølge slik at resultatet blir bokstaven T.

Oppgave 2

```
from turtle import *

speed(2)
shape("turtle")

for i in range(3):
    forward(100)
    left(120)
```

Skriv av koden ovenfor. Kjør koden og se hvilke figur den lager.

Endre tallene i for-løkken slik at koden lager en:

1. *Firkant*
2. *Femkant*
3. *Sekskant*

Oppgave 3

```
from turtle import *

setx(200)
sety(50)
setx(0)
sety(100)
setx(200)
```

1. Les av koden ovenfor, og gjett på resultatet.
2. Skriv av koden, og kjør programmet.
3. Endre programmet slik at resultatet blir tre parallelle linjer. Du får bruk for kommandoene **penup()** og **pendown()**.

Oppskrift:

1. Penn opp
2. Sett y til -150
3. Gjenta følgende 10 ganger:

(a) Penn ned
(b) Sett x til 200
(c) Sett x til 0
(d) Penn opp
(e) Øk y-koordinaten med 30

Endre programmet slik at du tegner 10 slike parallelle linjer. Se oppskriften ovenfor. Her er noen kodebiter du kan få bruk for:

```
penup()
pendown()
for i in range(10)
sety(ycor() + 30)
```