Universitetet i Agder

Master

8.trinn elevers første møte med tekstprogrammering

En kvalitativ flerkasusstudie om ungdomsskoleelevers første med møte med Python, deres utfordringer og hvordan de bruker sine tidligere erfaringer fra blokkprogrammet Scratch

CHRIS OWE GUTTORMSEN
MAGNUS REPSTAD

VEILEDER
Anders Skarpeteig Fidje

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

Forord

Når vi sitter sammen her i kantina på UiA, er vi både lettet og rørt. Vi har nå studert sammen i fem år, fra førsteåret med Kalkulus frem til innlevering av denne masteren. Det har vært mye latter, samarbeid og hjelp gjennom årene, og til tross for noen diskusjoner i arbeidet med denne studien står vi igjen som gode venner og ferdigutdannede lærere. Selv om lektorstudiet har krevd mye av oss, har det også gitt oss mye og vi går nå en spennende karriere i møte. Før vi takker for oss, er det også noen andre som fortjener vår takknemlighet.

Først og fremst gis en stor takk til Anders Skarpeteig Fidje for veiledning dette semesteret. En mer morsom og kunnskapsrik veileder i dette temaet kunne vi ikke forestilt oss. Vi må også takke Universitetet i Agder og alle forelesere vi har hatt, som har gitt oss muligheten til å ta en utdanning som kan forme oss selv og samfunnet rundt oss.

Videre utrettes også en stor takknemlighet ovenfor skolen og elevene på 8.trinn som lot oss gjennomføre undervisningsopplegg og intervjuer hos dem. Bidragene deres har gitt oss mange nyttige erfaringer, og et godt grunnlag for å kunne gjennomføre denne studien.

En stor takk utrettes i tillegg Dina Rosenberg og Marie Fasmer Karlsen for korrekturlesing og god støtte under prosessen.

Til sist ønsker vi å takke kullet vårt for fem flotte år ved lektorutdanning 8 – 13 trinn.

Kristiansand, mai 2023.

Chris Owe Guttormsen & Magnus Repstad.

Sammendrag

Programmering er fortsatt nytt i skolen, men inntil nå har hovedsakelig blokkprogrammering blitt brukt. Likevel er blokkprogrammering stort sett bare brukt i grunnskolen, og fra videregående og utover i arbeidslivet er det tekstprogrammering som foretrekkes. Denne studien undersøker 8.trinn elevers første møte med tekstprogrammering, og forsøker å svare på disse forskningsspørsmålene;

1. **Hvilke utfordringer har elever i sitt første møte med Python?**
2. **Hvordan bruker 8.trinn elever sine erfaringer i Scratch til å løse oppgaver i Python?**

Teorien som danner grunnlaget for å besvare disse spørsmålene er teorien om instrumentell skapelse, her representert av Buteau et al. (2020) og en studie av Qian og Lehman (2017). Den sistnevnte er basert på et rammeverk av Bayman og Mayer (1988) hvor nybegynneres utfordringer og misoppfatninger innenfor syntaks, begreper og strategier blir identifisert. Denne studien er designet som en kvalitativ flerkasus studie hvor tre ulike 8.klasser fikk oppgaver hvor de for første gang møtte programmeringsspråket Python. Seks grupper fra disse klasses deltok videre i et reflekterende intervju. Her ble deres utfordringer innenfor syntaks, begreper og strategi identifisert, sammen med deres forsøk på å bruke deres tidligere erfaring fra Scratch til å løse oppgavene i Python. Funnene tyder på at elever har utfordringer med syntaks i form av å skrive kode, definisjoner av datatyper og forståelse av indikatorer og feilmeldinger. Det virker også som at elevene har en del utfordringer knyttet til forståelsen av input og variabler, og sliter med å identifisere variabler i tekstprogrammer. Flere av elevene viste også en feilaktig forståelse av hvordan programmet gjør utregninger, og mangler gode strategier i arbeid med tekstprogrammering.

Resultatene er derimot ikke overraskende siden dette var deres første møte med tekstprogrammering, og elevenes ufullstendige forståelse av input og variabler stammer nok fra manglende erfaringer med dem. Studien har videre avdekket en del ulikheter mellom blokkprogrammering og tekstprogrammering og noen egenskaper i Scratch kan skape utfordringer for elever. Deler av elevenes kunnskap i blokkprogrammering er dermed ikke direkte overførbar til tekstprogrammering. Dette kan tyde på at grep burde gjøres innenfor arbeid med blokkprogrammering, slik at elevene blir bedre forberedt på overgangen, og eventuelt blir introdusert i tekstprogrammering tidligere.

Abstract

Programming is still a relatively new topic in the Norwegian school, but until now block-programming has mainly been used in secondary school. Block-programming is nevertheless mostly just used in primary and secondary school, but from high school and onwards text-programming is being used. This study aims to research 8. Grade pupils first meeting with text-programming, and tries to answer these research questions;

1. **Which challenges do 8. grade pupils have, in their first meeting with Python?**
2. **How do 8. grade pupils use their experiences in Scratch to solve exercises in Python?**

The theory which forms the foundation to answer these questions is the theory of instrumental genesis, hereby represented by Buteau et al. (2020), and another study by (Qian & Lehman, 2017). The latter is based on a framework developed by Bayman og Mayer (1988) where novices challenges and misconceptions regarding syntax, concepts and strategy are identified. This study is designed as a qualitative multiple-case study where three 8. grade classes got exercises where they met the programming language Python for their first time. Six groups from these classes then participated in a reflective interview. Their challenges regarding syntax, concepts, and strategy were then identified, as well as their attempts at using their prior experiences in Scratch to solve the exercises in Python. The findings suggest that pupils have challenges with syntax in the form of writing codes, defining data types, and understanding indicators and error messages. It also seems as if the pupils have some challenges connected to the comprehension of input and variables, and struggle to identify variables in text-programs. Several of the pupils also showed misconceptions regarding how the program makes calculations and lack good strategies in work with text-programming.

The results, however, are not surprising since this was their first encounter with text-programming, and the pupils incomplete understanding of input and variables probably stems from a lack of experience with it. The study has also uncovered some differences between block-programming and text-programming, and some features in Scratch can create challenges for pupils. Parts of the pupil's knowledge in block-programming cannot therefore not be directly transferred to text-programming. This could imply that steps should be taken within block-programming, so that the pupils are better prepared for the transition, and possibly introduced to text-programming earlier.

Innholdsfortegnelse

Forord i

Sammendrag ii

Abstract iii

1. Innledning 1

1.1 Bakgrunn for oppgaven 1

1.2 Tidligere funn og forskningsspørsmål 2

1.3 Oppgavens struktur 3

2. Teori 5

2.1 Programmering i skolen. 5

2.2 Algoritmisk tenkning 6

2.3 Instrumentell skapelse 7

2.3.1 Instrumentell skapelse gjennom instrumentering og instrumentalisering 7

2.3.2 Instrumentell integrasjon og modell for instrumentell skapelse 8

2.4 Studenters utfordringer og misoppfatninger i programmering 10

2.4.1 Kunnskap om syntaks 10

2.4.2 Kunnskap om begreper 11

2.4.3 Kunnskap om strategi 11

2.4.4 Faktorer som bidrar til misoppfatninger og utfordringer 11

2.4.5 Strategier og verktøy for å bearbeide utfordringer i programmering 13

2.5 Algoritmisk tenking og Instrumentell skapelse opp mot elevers misoppfatninger. 14

3. Metode 15

3.1 Syn på forskning 15

3.2 Forskningsdesign 15

3.3 Programmeringsspråk 16

3.4 Forskningsmetode 17

3.5 Utvalg av elever og grupper 18

3.6 Oppgavedesign 19

3.6.1 Oppgave 1 20

3.6.2 Oppgave 2 21

3.6.3 Oppgave 3 22

3.6.4 Oppgave 4 22

3.7 Transkripsjon og analysemetode 23

3.7.1 Transkripsjon 23

3.7.2 Analyseverktøy og koding 24

3.8 Forskningsetikk 25

3.9 Oppgavens reliabilitet og validitet 26

3.9.1 Reliabilitet 26

3.9.2 Validitet 26

4. Resultater og analyse **29**

4.1 Gruppe 1 ..... *29*

4.2 Gruppe 2 ..... *32*

4.3 Gruppe 3 ..... *36*

4.4 Gruppe 4 ..... *43*

4.5 Gruppe 5 ..... *45*

4.6 Gruppe 6 ..... *48*

5. Diskusjon **53**

5.1 Syntaksforståelse *53*

5.1.1 Forståelse av datatyper 53

5.1.2 Forståelse av fargeindikatorer og feilmeldinger i Python 54

5.1.3 Forståelse av Turtle 54

5.1.4 Rene syntaks/operator feil 55

5.2 Konseptuell forståelse *55*

5.2.1 Forståelse av print og programutregninger 55

5.2.2 Forståelse av input og variabler 56

5.2.3 Forståelse av løkker 56

5.3 Strategi *57*

5.4 Overgangen fra Scratch til Python *58*

6. Avslutning **61**

6.1 Konklusjon *61*

6.1.1 Konklusjon forskningsspørsmål 1 61

6.1.2 Konklusjon forskningsspørsmål 2 62

6.2 Studiens styrker og begrensninger *62*

6.3 Videre forskning og implikasjoner *63*

6.4 Utbytte av studien *64*

7. Litteraturliste **65**

8. Vedlegg **68**

8.1 Samtykkeskjema *68*

8.2 Søknad til NSD *70*

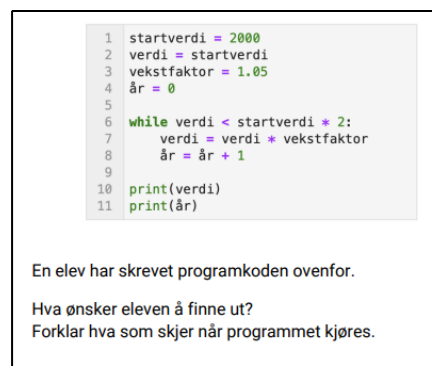
8.3 Oppgavesett ungdomsskole *72*

1. Innledning

1.1 Bakgrunn for oppgaven

Det er nå tre år siden fagfornyelsen, og med det er programmering inkludert i den norske skolen. I fagfornyelsen beskrives algoritmisk tenkning som en viktig del innenfor problemløsning i matematikk, og er nyttig for å lære seg å jobbe systematisk (Utdanningsdirektoratet, 2019a). Utdanningsdirektoratet (2020a) skriver også at: "Når elevene bruker programmering til å utforske og løse problemer, kan det være et godt verktøy for å utvikle matematisk forståelse". For å undervise i programmering på en relevant og læringsrik måte, bør lærere ha god kompetanse i programmering. Det kan tenkes at gode programmeringsferdigheter hos læreren ikke er nok, og at kjennskap til elevers utfordringer også er viktig.

I læreplanen LK20 for 1-10 trinn (Utdanningsdirektoratet, 2020b) nevnes det ikke hvilket programmeringsspråk elever skal programmere i, bare at elevene skal lære å tenke algoritmisk (Utdanningsdirektoratet, 2020a). Programmet Scratch er blitt den vanligste måten å programmere på i grunnskolen, til tross for at blokkprogrammering ikke brukes i videregående skole eller i yrkeslivet (Dolonen et al., 2019). Mange lærere foretrekker nok å undervise i Scratch siden dette blir sett på som et enkelt programmeringsspråk for nybegynnere, og kan sammenlignes med å bygge med legoklosser. Dette eliminerer mye av den vanskelige grammatikken og syntaksen som følger med tekstbaserte programmeringsspråk (Resnick et al., 2009). Elever kan derimot allerede i eksamensoppgaver i 1P (figur 1.1.1) møte oppgaver skrevet i tekstprogrammet Python. Det kan dermed tenkes at elever burde introduseres i tekstprogrammering allerede på ungdomstrinnet, slik at de er kjent med dette når de begynner på videregående. Videre har Python flere bruksområder enn Scratch, både innenfor matematikkundervisningen og i yrkeslivet (Mészárosová, 2015). Bakgrunnen for denne studien er derfor å avdekke hvilke utfordringer elever har i deres første møte med tekstprogrammering, og finne ut hvordan elevenes erfaring i Scratch hjelper dem i overgangen til Python. Dette kan være verdifull kunnskap for lærere, siden det støtter dem i å hjelpe elever. Samtidig kan studien bidra i utviklingen av bedre undervisningsopplegg som tilrettelegger for en enklere og tidligere overgang mellom programmeringsspråkene.



```
1  startverdi = 2000
2  verdi = startverdi
3  vekstfaktor = 1.05
4  år = 0
5
6  while verdi < startverdi * 2:
7      verdi = verdi * vekstfaktor
8      år = år + 1
9
10 print(verdi)
11 print(år)
```

En elev har skrevet programkoden ovenfor.

Hva ønsker eleven å finne ut?
Forklar hva som skjer når programmet kjøres.

Figur 1.1.1: Eksamensoppgave i matematikk 1P. Hentet fra eksamen, våren 2022 (Utdanningsdirektoratet, 2022, s. 15).

1.2 Tidligere funn og forskningsspørsmål

Programmering ble først innført i klasserommet på 80-tallet, da Papert (1993) presenterte sin LOGO-arena. Der kunne elever blant annet programmere en skilpadde til å utføre bevegelser og andre kommandoer. Målet med å programmere i matematikk, var å utvikle elevenes problemløsningskompetanse (Dolonen et al., 2019; Papert, 1993). Utover 80- og 90-tallet kom det derimot undersøkelser av blant andre Pea og Kurland (1984) og Mayer et al. (1986), som ikke fant en betydelig sammenheng mellom programmering og problemløsningskompetanse, og programmering forsvant derfor gradvis ut av skolen (Dolonen et al., 2019, s. 6). En annen bidragsyter til dette kan være at lærere er tradisjonelle, og ikke setter seg inn i eller ser nytten med ny teknologi umiddelbart (Vail, 2003).

Etter den nye innføringen av programmering i skolen har forskning om programmering igjen begynt å ta seg opp. Samtlige av disse er positivt innstilte til programmering, og vektlegger verdien innenfor problemløsning, algoritmisk tenkning og fremtiden (Sevik, 2016; Utdanningsdirektoratet, 2019a). Fokuset ligger derimot på algoritmisk tenkning, og det nevnes ikke nødvendigvis hvilke programmeringsspråk som skal brukes. Siden blokkbaserte programmeringsspråk som Scratch til nå har blitt mye brukt, har dette muligvis skygget for tekstbaserte programmeringsspråk. Gjentatte litteratursøk har ikke avdekket mye om norske elevers erfaringer og utfordringer innen tekstprogrammering, men det finnes derimot en del internasjonal forskning. Qian og Lehman (2017) har undersøkt elevers utfordringer innenfor tekstprogrammering, og mulige faktorer som skaper dem.

I en studie av Powers et al. (2007) ble det observert at elever hadde problemer med syntaks når de gikk fra blokkprogrammering til tekstprogrammering. I tillegg avdekket de at elever ble demotiverte da de innså at blokkprogrammering hovedsakelig brukes i grunnskolen, og ikke brukes i høyere utdanning eller arbeidslivet. Grover et al. (2015) fant ut at elever i 7.- og 8. klasse i USA klarte å overføre kunnskap fra Scratch til tekstbasert programmering. De hadde derimot flere utfordringer, særlig knyttet til vanskelige konsepter som løkker og betingelser[1]. De mente likevel at denne overgangen er gunstig, og kan bedre elevenes forståelse for digitale verktøy. Armoni et al. (2015) avdekket at elever som tidligere hadde lært Scratch trengte mindre tid til å lære seg nye emner innenfor programmering, og møtte færre utfordringer.

---

[1] Betingelser eller vilkår i programmering avgjør at et program skal utføre noe avhengig av om betingelsen er innfridd eller ikke.

Kölling et al. (2015) har identifisert mange av de samme utfordringene, og presenterte en ny tilnærming til overgangen kalt «frame-based editing». Veilederen i denne studien har tidligere veiledet Ore (2022) i sin masteroppgave om elevers arbeid med Scratch. I denne ble blant annet programmering undersøkt i lys av instrumentell skapelse (Buteau et al., 2020), og i prosessen ble i tillegg noen begrensninger ved Scratch avdekket. Inspirert av Ore sitt perspektiv på instrumentell skapelse, og med disse begrensingene i bakhode er målet med denne studien å undersøke elevers første møte med tekstprogrammering.

Selv om det ikke kan forventes at elever skal klare å lage egne tekstbaserte programmer, er det likevel ønsket at de begynner overgangen til tekstprogrammering så tidlig som mulig. Denne studien ønsker videre å undersøke hvorvidt utfordringene funnet av tidligere forskning også gjelder for norske elever, og om deres tidligere erfaringer i Scratch hjelper dem å løse oppgavene i Python.

I bakgrunn av målene over stilles derfor følgende forskningsspørsmål:

1. **Hvilke utfordringer har elever i sitt første møte med Python?**
2. **Hvordan bruker 8.trinn elever sine erfaringer i Scratch til å løse oppgaver i Python?**

1.3 Oppgavens struktur

Oppgaven består av seks kapitler;

1. Kapittel består av bakgrunn for oppgaven, tidligere forskning og forskningsspørsmål.
2. Kapittel presenterer det teoretiske rammeverket som danner grunnlaget for analysen og diskusjonen.
3. Kapittel legger frem metodikken i studien og valgene som er gjort angående forskningsdesignet, datainnsamling og analyse begrunnes. Etiske betraktninger og oppgavens reliabilitet og validitet blir også drøftet her.
4. Kapittel presenterer resultatene som brukes til å besvare forskningsspørsmålene.
5. Kapittel drøftes resultatene og diskuteres i lys av teori og tidligere forsking.
6. Kapittel fremstiller avslutningsvis konklusjoner rundt forskningsspørsmålene, sammen med mulige implikasjoner for matematikkundervisning og videre forskning.

2. Teori

Det teoretiske kapittelet innebærer å gjøre rede for tidligere forskning rundt programmering og algoritmisk tenking. Videre baserer teorien seg på instrumentell skapelse og på elevenes utfordringer innenfor programmering. Til slutt oppsummeres teoriene sammen slik at betydningen og sammenhengen i rammeverket blir tydelig for analysen og diskusjonen.

2.1 Programmering i skolen.

Programmering kan defineres som et dataprogram som avgjør hvordan en datamaskin skal operere, mens programmet kjører (Store norske leksikon, 2022). Det er viktig å kunne skille mellom programmering og koding ifølge Gjørvik og Torkildsen (2019, s. 34). De definerer programmering som kommunikasjon gjennom logikk og utvikling av programmer, mens koding derimot er selve aktiviteten hvor elever skriver kode eller bygger med kodeblokker.

I 2013 kom Norges Offentlige Utredninger (NOU) med sin utredning «Hindre for digital verdiskapning», og påsto at dagens barn og unge er storkonsumenter av teknologi, men ikke innehar kompetanse om hva som faktisk skjer bak skjermen. Det pekes også på at manglende digitale ferdigheter blant barn og unge kan føre til mindre tjenesteutvikling og verdiskapning i fremtiden (NOU, 2013). Sevik (2016) nevner i sin rapport at programmering kan oppleves som mer relevant og yrkesrettet for elever, og kan motivere dem til økt innsats i skolen. Videre peker Sevik (2016) på tre økende trender i Europa;

1. Programmering blir i økende grad sett på som en tverrfaglig kompetanse som integreres i eksisterende fag
2. Programmering innføres på stadig lavere trinn, helt ned i barneskole
3. Økt oppmerksomhet på algoritmisk tenkning (computational thinking) når man velger å ta programmering inn i skolen.

I tillegg ble tre metoder for å innføre programmering presentert; programmering som et eget IKT-fag, programmering som et tverrfaglig emne, eller som en del av matematikkfaget (Sevik, 2016). Norge gikk for en kombinasjon av de to siste, og ifølge læreplanen skal elever lære å programmere i naturfag, musikk og kunst og håndverk, men matematikken har hovedansvaret for programmeringsopplæringen (Flø, 2021). Det er også tydelig at programmering er innført på lavere trinn, slik vi ser i dette kompetansemålet for 5.trinn (Utdanningsdirektoratet, 2020b):

- Lage og programmere algoritmer med bruk av variabler, vilkår og løkker.

Bakgrunnen for å inkludere programmering i matematikken er at elever skal lære å tenke algoritmisk, og bruke dette i problemløsning (Flø, 2021).

2.2 Algoritmisk tenkning

Utdanningsdirektoratet (2019a) definerer algoritmisk tenkning som en problemløsningsmetode hvor en løser problemer på en systematisk måte, og vurderer hvilke steg som må tas for å løse problemet. Det at elevene gjør feil og oppdager dem er en viktig del av prosessen, og det er essensielt at de ikke gir opp underveis (Utdanningsdirektoratet, 2019a). Et av kjerneelementene til matematikk 1 – 10. trinn er «Utforsking og problemløsing». Dette kjerneelementet nevner algoritmisk tenkning som en del av prosessen for å kunne løse matematiske problemer og utvikle strategier og framgangsmåter (Utdanningsdirektoratet, 2019b).

For å opparbeide algoritmisk tenkning kan en prøve å koble den opp mot hverdagslige hendelser (algoritmer). Eksempler kan være å følge matoppskrifter eller bruksanvisninger for å lage et produkt. Dette kalles for analog programmering (Mannila, 2017). Wing (2006) har som Utdanningsdirektoratet (2019a) en lignende og dypere definisjon av algoritmisk tenkning (computational thinking). Den baserer seg på å løse problemer, utvikle systemer og logaritmer, og deretter bruke dette i programmeringen. Wing (2006) forklarer videre at algoritmisk tenkning handler om å kunne avgjøre hva mennesker kan gjøre bedre enn datamaskiner og hva datamaskiner kan gjøre bedre enn mennesker.

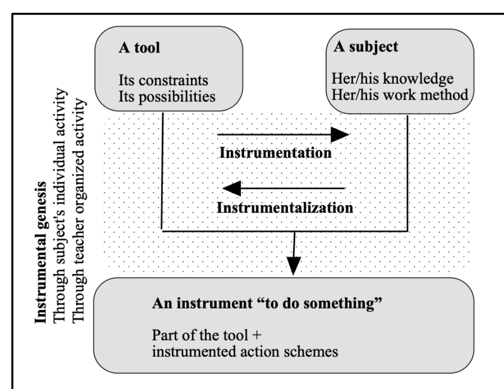
Gjørvik og Torkildsen (2019) viser også frem begrepsbruken av algoritmisk tenkning på norsk og engelsk, da mange av begrepene kan høres like ut. Den direkte oversettelsen av algorithmic thinking er algoritmebehandling, som er en undergruppe av computational thinking (Gjørvik & Torkildsen, 2019, s. 32). Dermed er det viktig å poengtere at når begrepet algoritmisk tenkning blir brukt på norsk, så er det computational thinking (Wing, 2006) det dreier seg om, og ikke algorithmic thinking. Videre handler programmering ikke bare om å kunne bruke eksisterende algoritmer, men at en selv utvikler algoritmer til å løse problemer (Gjørvik & Torkildsen, 2019, s. 36). Dette går igjen i Utdanningsdirektoratet (2019a) og (Wing, 2006) sine definisjoner av algoritmiske tenkning.

2.3 Instrumentell skapelse

Instrumentell skapelse defineres som en prosess der et subjekt samhandler med et artefakt, og gjennom prosessen utvikler et instrument (Buteau et al., 2020, s. 372). Utviklingen av et instrument innebærer blant annet om elevers utvikling av skjemaer (Vergnaud, 1998). Vergnaud (2009, s. 88) definerer skjemaer som det å kunne bruke tidligere erfaringer til å løse nye problemer. Utviklingen av skjemaer er grunnleggende for den instrumentelle skapelsen ifølge Buteau et al. (2020). Den instrumentelle skapelsen er et produkt av disse skjemaene og lærerens instrumentelle integrasjon, som fokuserer på lærerens rolle i arbeid med teknologi og matematikk i undervisningen (Assude, 2007). Buteau et al. (2020) har undersøkt hvordan universitetsstudenter arbeider med programmering. Studentens utvikling av skjemaer underveis i programmering har gitt grunnlaget for Buteau et al. (2020) sitt teoretiske perspektiv, presentert som instrumentell skapelse.

2.3.1 Instrumentell skapelse gjennom instrumentering og instrumentalisering

Instrumentell skapelse bygger blant annet på Vygotsky (1978) sin aktivitetsteori innen menneskets utviklingsprosess. Aktivitetsteorien innebærer at et subjekt (eleven) arbeider og samhandler målrettet med ulike medierende artefakter (produkter produsert av mennesker) og dermed utvikler et instrument, med tilhørende skjemaer i hvordan artefaktet kan brukes (Vergnaud, 1998). Instrumentell skapelse kan deles inn i to parallelle prosesser: Instrumentering der artefaktet påvirker subjektet, og instrumentalisering der subjektet påvirker artefaktet (Buteau et al., 2020, s. 372). Instrumentalisering handler om å kunne bruke et artefakt, og kunnskap om hvilke egenskaper og begrensninger det har. Et eksempel er å bruke en datamaskin og åpne dataprogrammet som skal brukes. Instrumenteringen handler om hvordan artefaktet (programmet) påvirker elevene i deres arbeid med matematiske oppgaver. Artefaktet er derimot ikke meningsfullt for elevene før de har utviklet skjemaer, og snur artefaktet om til et matematisk instrument. Videre Utviklingen av instrumentet vil og kun være hensiktsmessig dersom forholdet mellom artefaktet og subjektet er opparbeidet og at instrumentet har en hensikt for studenten (Buteau et al., 2020, s. 373).



Figur 2.3.1: Instrumentell genesis. Hentet fra Trouche (2004, s. 289)

Instrumentering og instrumentalisering henger sammen ved at de begge innebærer selve subjektet, og artefaktet som subjektet skal bruke for å løse matematiske problemer. Skjemaene for instrumentering og instrumentalisering er assosiert med situasjonen eleven er i, og danner grunnlaget for en teoretisk modell for observasjon om hvordan eleven lærer (Vergnaud, 1998). Instrumentering og instrumentalisering er resultater av både konseptuelle og tekniske deler (Buteau et al., 2020). Den tekniske delen er det observerbare når eleven arbeider, mens den konseptuelle delen er derimot ikke direkte observerbar, men kan observeres gjennom mønstre som elevene gjør (Buteau et al., 2020, s. 373). Siden elevenes instrumentelle skapelse ikke kan observeres direkte, fokuseres det i denne studien på de tekniske og konseptuelle delene når elevene observeres under undervisningstimene og intervjuene.

Instrumentell skapelse består av skjemaer innen programmering og matematikk. Disse skjemaene betegnes som p og m skjemaer (Buteau et al., 2020, s. 374). Skjemaer knyttet opp mot selve programmeringen kalles p-skjemaer, mens m-skjemaer er knyttet opp mot matematikken. Disse skjemaene kan også samles til p+m skjemaer, hvor det er en sammenheng mellom programmeringen og matematikken. Instrumentet består dermed av artefaktet og de tilhørende p-, m- og p+m-skjemaene (Buteau et al., 2020). I neste delkapittel danner Buteau et al. (2020) en oversikt over de ulike stegene av instrumentell skapelse ut ifra lærerens instrumentelle integrasjon (Assude, 2007) og p- og m-skjemaene.

2.3.2 Instrumentell integrasjon og modell for instrumentell skapelse

Instrumentell integrasjon er relasjonen mellom den instrumentelle og matematiske dimensjonen til læreren, og hvordan undervisningen er lagt opp (Assude, 2007, s. 1341). Buteau et al. (2020) baserer modellen for instrumentell skapelse opp mot instrumentell integrasjon, elevenes instrumentelle kunnskap, matematiske kunnskap og sammenhengen mellom dem og elevens utvikling av skjemaer. Instrumentell kunnskap vil si å kunne bruke et program og programmeringsspråket til problemløsing uten matematikk. Den instrumentelle kunnskapen opparbeides gjennom instrumentet som elevene utvikler gjennom sin instrumentelle skapelse. Matematisk kunnskap er kunnskapen elevene har opparbeidet seg i matematikk.
De ulike stegene av lærerens utvikling baseres på om oppgavene elevene arbeider med er rettet mot programmeringsbaserte verktøy eller ren matematikk. Instrumentell integrasjon handler altså om hvordan læreren tilrettelegger for elevenes instrumentelle skapelse, av p- og m-skjemaer og utvikling av instrumentell kunnskap opp mot deres matematiske kunnskap.

Buteau et al. (2020, s. 376-377) har utviklet en modell, som kan måle studentenes utvikling innenfor instrumentell skapelse. Denne er basert på instrumentell integrasjon (Assude, 2007) og elevenes bruk av p+m skjemaer, og er delt i fire steg:

I) Ved instrumentell initiering har elever ikke noe kjennskap til programmering fra før av. Læreren sitt mål er da hovedsakelig at elevene blir kjent med programmet og lærer det konseptuelle med programmet og programmeringsspråket. Her blir elevene guidet gjennom programmeringsoppgaver, og utvikler hovedsakelig p-skjemaer. Her er da sammenhengen mellom det instrumentelle og matematiske lav (Assude, 2007, s. 1342) og elevenes matematiske kunnskap er høyere enn deres instrumentelle kunnskap.

II) I instrumentell utforskning har eleven noe kunnskap om programmering, men skal blir videre kjent med programmet gjennom å arbeide med matematiske oppgaver. Her blir dermed både den matematiske og instrumentelle kunnskapen utviklet, og det er da en større sammenheng mellom dem. Sammenhengen er avhengig av hvordan de matematiske oppgavene er opparbeidet, og hvilke instrumentelle kunnskaper elevene har. Her opparbeider elevene p+m skjemaer.

III) I den instrumentelle forsterkningen har eleven opparbeidet seg kunnskap om programmet, men stoppes opp av instrumentelle problemer mens de arbeider med den matematiske oppgaven.

IV) I det siste steget, instrumentell symbiose, har eleven kjennskap til programmet og arbeider med oppgaver som videreutvikler deres instrumentelle og matematiske kompetanse. Sammenhengen mellom p- og m-skjema er stor.

Denne studien er rett mot elevenes instrumentelle initiering og utforskning, siden elevene ikke har noe erfaring med tekstprogrammering tidligere, og har lite erfaring om annen type instrumentell forståelse innen programmering generelt. Derfor er disse utdypet mer, framfor instrumentell forsterkning og instrumentell symbiose.

2.4 Studenters utfordringer og misoppfatninger i programmering

For å identifisere og kategorisere elevenes utfordringer i denne studien vil rammeverket brukt av Qian og Lehman (2017), bli anvendt. Dette rammeverket er basert på et tidligere arbeid av blant annet Bayman og Mayer (1988) og McGill og Volet (1997). Her identifiseres tre ulike typer kunnskap i programmering; Syntaks, begreper og strategi. I figur 2.4.1 presenterer Mayer og Fay (1987) en tidlig utgave av dette rammeverket med eksempler fra LOGO. Disse eksemplene er fortsatt aktuelle i Python gjennom Turtle-biblioteket. Elevers utfordringer i programmering er vanskelige å definere og Qian og Lehman (2017, s. 3) bruker dermed feil, misoppfatninger, misforståelser og utfordringer om hverandre. Selv om misoppfatninger som regel er koblet til konseptuell kunnskap, kan det også omhandle syntaksfeil eller ufullstendige forståelser. Når misoppfatninger brukes i denne studien, er det Qian & Lehman sin definisjon som anvendes.

Table 1
Chain of Cognitive Changes Promoted by Learning to Program

	Change	Logo example
1.	Learning language features (syntax)	Child recognizes that RT, LT, FD, and BK are command keywords for "right," "left," "forward," and "back."
2.	Learning to think within the domain of programming (semantics)	Child recognizes that "right" refers to the Turtle's right side rather than the right side of the screen.
3.	Learning to think in domains outside of programming (transfer)	Given a map and a list of instructions, child can navigate.

Figur 2.4.1: Oversikt av et tidligere rammeverk over de tre ulike typene av kunnskap innenfor programmering med eksempler i LOGO. Hentet fra Mayer og Fay (1987, s. 269)

2.4.1 Kunnskap om syntaks

Kunnskap om syntaks omhandler programmeringsspråkets egenskaper, fakta og regler (Bayman & Mayer, 1988). Denne kunnskapen inkluderer blant annet bruk av kolon, eller hvordan en løkke skal oppføres. Selv om det er nødvendig med riktig syntaks for å få et program til å fungere, er det ikke nok til å løse problemer eller utvikle programmer (McGill & Volet, 1997). I en studie av Wong et al. (2019) identifiserte dem de vanligste syntaksfeilene som; ugyldig syntaks, manglende innrykk eller parenteser, og uventet EOF (feil/manglende endelse i kodelinjer). Ugyldig syntaks kan være feil i tegnsetting og rettskriving, eksempelvis feilplasserte punktum eller kolon, manglende anførselstegn eller feilstavelser. Innrykkfeil er aktuelle i løkker og betingelser, og dersom kodelinjer under disse ikke er innrykket vil den ikke bli kjørt. Måten koden avsluttes på er også kritisk, og manglende kolon etter betingelser vil føre til syntaksfeil (Wong et al., 2019). Syntaksfeil er hyppigst blant nybegynnere, men samtidig er slike feil som regel enkle å oppdage og fikse (Qian & Lehman, 2017). Et eksempel av denne typen kunnskap i programmering er kunnskap om *turtle*-biblioteket i Python. *Turtle*-biblioteket importerer en skilpadde i Python som blant annet kan bevege seg og tegne. Å huske kommandoer som *turtle.right()* eller *forward()*, og skrive dem med riktig tegnsetting kan regnes som kunnskap om syntaks.

2.4.2 Kunnskap om begreper

Kunnskap om begreper, eller konseptuell kunnskap refererer til en forståelse for hvordan et program utfører handlinger og de ulike begrepene som blir brukt i programmet. Det innebærer også kunnskap om hvordan syntaks, tegn og begreper kan settes sammen til å lage funksjonerende programmer (McGill & Volet, 1997, s. 278). Manglende konseptuell kunnskap kan føre til dypere og mer betydelige syntaksfeil. Misoppfatninger av hvordan variabler fungerer eller hvilke betingelser som er gjeldende er vanlige konseptuelle feil. Hvordan et program fungerer, og rekkefølgen det kjører i kan også være utfordrende for nybegynnere. Dersom elever ikke forstår begrepene eller prinsippene i programmering vil de slite med å finne og rette feil (Qian & Lehman, 2017). I programmering av en skilpadde i Python, kan konseptuell forståelse kjennetegnes av at elevene forstår at kommandoen *turtle.right(90)* snur skilpadden 90° grader mot skilpadden sin høyre side, og ikke at den går et antall steg mot høyre.

2.4.3 Kunnskap om strategi

Strategisk kunnskap handler om å kunne planlegge, skrive og feilsøke programmer på en effektiv måte. Strategisk kunnskap er også nødvendig for å kunne identifisere hvilke oppgaver som kan løses med programmering, og for å finne logikkfeil eller begrensninger i programmene (McGill & Volet, 1997, s. 278). Elevers utfordringer innenfor strategi er korrelert til deres kunnskap i syntaks og begreper, derfor vil elever som mangler kunnskap innenfor syntaks og begreper slite med å utvikle gode strategier og programmer. Det trekkes likevel også frem at kunnskap om syntaks og begreper ikke nødvendigvis tilsvarer gode programmeringsstrategier (Qian & Lehman, 2017). Eksempelvis kan elever forstå variabel som begrep, men de har problemer med å inkludere dem riktig i programmet sitt. Dersom elever klarer å utvikle et program til å løse matematiske oppgaver ville dette vært et eksempel på strategisk kunnskap.

2.4.4 Faktorer som bidrar til misoppfatninger og utfordringer

Gjennom de siste tiårene er det blitt identifisert mange faktorer som kan bidra til utfordringer i programmering. Qian og Lehman (2017) har i sin artikkel en grundig gjennomgang av tidligere forskning, hvor de identifiserer disse ulike faktorene. Disse funnene kommer opprinnelig fra tidligere forskning, men refereres og oppsummeres på en oversiktlig måte av Qian og Lehman (2017, s. 7-10). Relevante faktorer for denne studien presenteres videre i dette delkapittelet.

Den første faktoren som kan bidra til utfordringer for elevene, er om oppgavene blir for vanskelige og komplekse. Dette fører til at elevene gjør flere feil rundt syntaks og begreper, siden oppgavene krever mer tenkning av elevene. Tekstbaserte programmeringsspråk og kommandoer er i tillegg basert på naturlige språk som engelsk, og elever kan dermed møte kolliderende oppfatninger av ord og uttrykk. Det engelske ordet "and" er et bindeord i engelsk, men brukes som en Boolsk operator i programmering. I tillegg er det funnet at gode engelsk kunnskaper hjelper elever i programmeringen. Elevenes tidligere matematiske kunnskap kan også hindre deres evne til å programmere. Dersom deler av programmets koder strider med konvensjonell matematikk, kan dette skape utfordringer hos elever. Et eksempel på dette kan være x=x+1, hvor variabelen x blir tilegnet en ny verdi som er lik seg selv pluss en, noe som ikke gir mening i konvensjonell algebra (Qian & Lehman, 2017).

Elever kan også ha en ufullstendig eller unøyaktig forståelse av hvordan datamaskiner og koder fungerer. Dette kan føre til at elever misoppfatter hvordan en kode blir utført og hvordan datamaskiner bearbeider koder. Elever kan også danne seg en feilaktig forståelse om at programmer er intelligente, og tenker på samme måte som mennesker. Videre kan elever danne seg en fragmentert forståelse av syntaks og begreper. Det kan gjøre at elevene ikke ser mønstrene og strategiene i programmering helhetlig (Qian & Lehman, 2017).

Selv om en elev har riktig syntaks- og begrepsforståelse, er det derfor ikke sikkert at eleven klarer å evaluere eller forklare et program ordentlig. Miljøfaktorer og egenskaper i programmeringsspråk kan også skape utfordringer for elever. For eksempel kan addisjonsoperatoren «+» brukes både i utregning og til å sette samme tekststrenger[2]. Denne fleksibiliteten er en fordel for de som kjenner til det, men kan være forvirrende for elever. Feilmeldingene fra et program er ikke alltid til hjelp for nybegynnere i programmering, og kan fremstå som kryptiske og vanskelige å forstå. I tillegg er feilsøking-strategier ofte ikke undervist tidlig i programmeringskurs (Qian & Lehman, 2017).

[2] En tekststreng er en streng av bokstaver eller tall, men behandles uansett som tekst i programmet. En tekststreng indikeres i Python ved anførselstegn ("").

Lærerens instruksjoner og kunnskap kan også skape utfordringer for elever. Dersom læreren selv ikke har en god forståelse for syntaksen og begrepene i programmering kan misoppfatninger overføres til elevene. I tillegg fokuserer gjerne lærere som er uerfarne i programmering mer på regler og prosedyrer fremfor konseptuell forståelse. Dette kan gjøre at elever danner ufullstendige forståelser og strategier innenfor programmering (Qian & Lehman, 2017). Dette er særlig en bekymring ettersom programmering nå er tatt inn i skolen uten at alle lærere selv har fått tilstrekkelig opplæring og erfaring innenfor programmering (Johansen, 2020).

2.4.5 Strategier og verktøy for å bearbeide utfordringer i programmering

Qian og Lehman (2017, s. 11-15) har også samlet opp noen strategier og verktøy som kan hjelpe elever gjennom disse utfordringene. En strategi er å bruke gode eksempler og elever kan lære mye av programmeringseksempler dersom programmene er oversiktlige og designet for elevenes forståelse. Nybegynnere i programmering har i tillegg en vane for å lese programmer linje for linje, og forstår ikke helheten av et program. Å la elever lese og tolke ferdiglagde programmer kan hjelpe elevene i å forstå programmer mer helhetlig, og se sammenhengene mellom kodelinjene.

I noen programmeringsspråk blir syntaksfeil som manglende parenteser, anførselstegn og semikolon indikert. Dette gjør at elevene enklere kan finne feilene, og deretter rette dem (Qian & Lehman, 2017). En tilnærming for å unngå syntaksfeil er å programmere i grafiske programmer som Scratch. I grafiske programmer kobles bare de grafiske blokkene i koden sammen hvis de kompatible, og elevene unngår syntaksfeil i kodeskrivingen. Løkker kan være utfordrende å lære seg i tekstbaserte programmeringsspråk, men er enklere i Scratch (Qian & Lehman, 2017). Siden løkke-blokken i Scratch er C-formet, legger opp til at andre blokker skal plasseres inni dem, og elevene kan enkelt utforske ved å dra blokker rundt i programmet (Resnick et al., 2009).

2.5 Algoritmisk tenking og Instrumentell skapelse opp mot elevers misoppfatninger.

Elevenes algoritmiske tenkning, instrumentelle skapelse og deres utfordringer i programmering samles herunder til et utfyllende teoretisk rammeverk. Gjennom algoritmisk tenkning opparbeider elever seg kunnskap i systematisk problemløsning, lærer å avgjøre hvilke problemer som kan løses av datamaskiner og hva mennesker må løse selv (Utdanningsdirektoratet, 2019a; Wing, 2006).

For å utvikle sin algoritmiske tenkning må elevene opparbeide seg skjemaer som består av handlingsregler, som er utviklet fra problemer de har mestret tidligere (Vergnaud, 1998; Vergnaud, 2009). Gjennom disse skjemaene, og prosessene instrumentering og instrumentalisering, danner elevene seg et instrument. Når elevene anvender instrumentet i matematiske problemer, vil sammenhengen mellom deres instrumentelle kunnskap og matematiske kunnskap styrkes (Buteau et al., 2020). For å skape denne sammenhengen, er det viktig at læreren integrerer bruken av instrumenter i matematikkundervisningen. Lærerens bruk av instrumenter og programmering i undervisningen gir da grunnlag for elevenes utvikling av p-, m- og p+m-skjemaer, som deretter hjelper elevene i å løse matematiske problemer gjennom programmering (Assude, 2007).

Instrumentell integrasjon er dermed grunnlaget for elevenes instrumentelle skapelse (Buteau et al., 2020). For å integrere instrumenter som programmering i undervisningen, burde læreren kjenne til og forberede seg på utfordringene elevene kan møte i programmeringen. (Assude, 2007). Disse utfordringene klassifiseres i denne studien i de tre typene av kunnskap i programmering; syntaks, begreper og strategier (Bayman & Mayer, 1988; Qian & Lehman, 2017). Ved å identifisere elevenes (subjektet) og programmets (artefaktet) utfordringer og begrensinger, gis det et bedre grunnlag for lærerens instrumentelle integrasjon og elevenes instrumentelle skapelse.

3. Metode

I dette kapittelet presenteres utviklingen av oppgavene, planleggingen og gjennomføringen av datainnsamlingen, samt valg av analysemetode. Perspektivene på forskningsdesign og metode forklares først. Deretter begrunnes valget av programmeringsspråk og oppgavene, og hvordan utvalget av elever og grupper ble gjort. Avslutningsvis drøftes de etiske sidene av studien og dens reliabilitet og validitet.

3.1 Syn på forskning

Ved gjennomlesing av forskningsoppgaver, er det nyttig å forstå hvilket syn på forskning forskerne bruker. Synene som ligger til grunn i tolkningene og analysen presenteres her.
Denne studien bærer et interpretivistisk syn på forskning. Det betyr at forskere i sosiale situasjoner ikke kan forstå og tolke menneskers handlinger objektivt. Siden handlingene er subjektive, forsøkes det å forstå hva som ligger bak dem. Forskere må derfor tolke resultatene i konteksten de er innhentet i, og konteksten er viktige for å kunne forstå funnene (Bryman, 2016, s.26-28). Ved å vise transkripsjonene kan lesere se hvilke av elevenes ord og handlinger som tolkes, og hvilken kontekst de er tolket i. Forskerne i denne studien erkjenner at det som blir lagt frem ikke er fullstendig og objektivt, men at det er deres tolkninger av den gitte sosiale forskningen (Bryman, 2016, s. 29). Siden dette synet på forskning anerkjenner de sosiale rammene, kan det tenkes at det er godt egnet for forskning i skolen, spesielt i lys av den sosiokulturelle tradisjonen. Elever, lærere og forskere opererer alle innenfor disse sosiale rammene, hvor alle påvirker hverandre. Gjennom undervisning og spørsmål som gis til elevene i denne studien vil påvirkes dem av forskerne, og forskernes tolkninger av deres handlinger og ytringer vil igjen påvirkes av elevene.

3.2 Forskningsdesign

I denne studien er det brukt et kvalitativt perspektiv. Kvalitativ forskning bruker generelt ikke numerisk datamateriale, men studerer hovedsakelig individers oppfatninger og handlinger (Bryman, 2016). Fordelen med dette perspektivet er at den tillater forskere å undersøke menneskers tanker og følelser, og analysere dem i konteksten de opptrer i. Kvalitative prinsipper tillater også forskere å starte mer åpent, for å deretter spisse inn eller endre fokuset sitt etter hvert som dataene undersøkes.

Begrensningene ved denne tilnærmingen er at resultatene er uforutsigbare og subjektive, og ikke nødvendigvis kan generaliseres utover denne studien. Siden denne tilnærmingen også er tidkrevende og komplisert, er studien ofte begrenset til et mindre utvalg (Bell & Waters, 2018, s. 25-26). Videre kan denne studien klassifiseres som en flerkasusstudie. Det betyr at flere kasus blir studert, og forskeren forsøker eksempelvis å utforske og sette seg inn i en person, gruppe, skole eller et samfunn (Bryman, 2016, s. 60-61). En kasusstudie tillater også forskere å studere noe i dybden, og kan hjelpe å identifisere viktige aspekter som kan undersøkes videre (Bell & Waters, 2018, s.28-29). I denne studien er seks grupper på 8.trinn utvalgt, og gruppene studeres som seks ulike kasus analysert etter samme teoretisk rammeverk. Dette forskningsdesignet åpner opp for å undersøke og identifisere hvilke utfordringer disse elevgruppene har. Det er flere fordeler ved å studere de ulike gruppene som seks ulike kasus. Det åpner blant annet opp for muligheten til å sammenhenger og trekke konklusjoner på tvers av kasusene, og hjelper i byggingen av en mulig teori (Bryman, 2016, s. 67). I tillegg er det en ryddig måte å gjennomføre transkripsjonene på, som kan tydeliggjøre eventuelle likheter eller ulikheter mellom de seks kasusene.

3.3 Programmeringsspråk

Når det kommer til valg av programmeringsspråk i skolen er det de blokkbaserte språkene Scratch og Makecode, og tekstbaserte Python og Javascript som dominerer (Lær Kidsa Koding, 2023). Scratch er et program som baseres på kodeblokker, der blokkene settes sammen for å lage programmer. Syntaksen og grammatikken som finnes i tekstbaserte programmeringsspråk er dermed ikke et problem i Scratch, og blokkene kan bidra til utforskning av programmers oppbygning og til lek (Resnick et al., 2009). Når elever ikke trenger å lære syntaks, kan det fokuset rettes mot problemløsning og algoritmisk tenkning (Mladenović et al., 2016).

Python er det tekstbaserte programmeringsspråket som er mest utbredt i skolen (Lær Kidsa Koding, 2023). Det er kjent for å være enkelt og oversiktlig med enkel syntaks og innebygde biblioteker som kan legges til ved behov. Et eksempel er biblioteket Turtle som kan tegne figurer. I motsetning til Scratch brukes Python av profesjonelle, og kompetanse og ferdigheter i Python har dermed en verdi utenfor skolen (Mészárosová, 2015).

Siden gruppene som deltar i forskningsprosjektet kun har arbeidet i Scratch, ble dette grunnlaget for oppgavene. Det andre forskningsspørsmålet i denne studien går ut på å undersøke hvorvidt instrumentell kunnskap og erfaringer i Scratch er overførbare til Python. Noen av oppgavene skrives derfor i Trinket, en nettside hvor en kan lage programmer skrevet med programmeringsspråket Python uten å laste ned noe program/editor. Dette gjør det mulig å studere elevenes første møte med Python, og hvordan de tolker og forstår Python uten noe tidligere erfaring i programmeringsspråket.

3.4 Forskningsmetode

Ettersom elevene i denne studien ikke hadde arbeidet i Python tidligere, får elevene først en undervisningstime hvor de arbeider med oppgavearket (vedlegg 7.3). De mottar ingen innføring i Python slik at kun deres tidligere erfaringer i Scratch vil kunne hjelpe dem. Alle elevene i klassene deltar i undervisningen, og gjennomfører oppgavene i klasserommet slik at de arbeider i trygge og kjente rammer. Elevene arbeider parvis med den de sitter sammen med. Samarbeid og par-programmering har flere fordeler ifølge Sevik (2016). I par-programmering kan elevene innta ulike roller som "sjåfør" og "kartleser", og sammen dra nytte av hverandres erfaringer og kunnskap. På denne måten kan elevene sammen diskutere og reflektere over utfordringene de møter. Elevene ble observeres i undervisningstimen og får hjelp dersom det oppstår tekniske problemer. De får derimot ikke hjelp til å løse oppgavene, og anbefales heller til å begynne på neste oppgave om de sitter fast. Gjennom observasjonen vil det dannes et inntrykk av elevenes nivå og hvilke spørsmål som kan stilles under intervjuene.

Umiddelbart etter undervisningstimene vil det avholdes semi-strukturerte intervjuer. I et semi-strukturert gruppeintervju følger intervjuerne en intervjuguide med spørsmål, der spørsmålene kan endres på underveis, og intervjudeltakernes sine svar kan endre fokuset i intervjuet. Denne intervjuformen er fleksibel og godt egnet til sosiale studier (Bryman, 2016, s. 468). For å være mer til stede under intervjuet, vil lyd- og videoopptak benyttes under intervjuene. Dette gjør det mulig å lytte mer aktivt til deltakerne og skaper en mer naturlig dialog, siden notering ikke er nødvendig. En ulempe ved bruk av videokamera og lydopptaker er at det kan forstyrre eller gjøre deltakerne ukomfortable, da de vet at deres besvarelser blir spilt inn (Bryman, 2016, s. 469). Til intervjuene vil derfor lydopptaker og videokamera plasseres litt unna bordet slik at elevene blir minst mulig distrahert. De vil også få en tydelig innføring i hvordan dataen oppbevares, brukes og anonymiseres.

Under intervjuene vil oppgavene fra undervisningstimen bli gjennomgått en etter en, der elevene deler sine løsninger, tankemåter og refleksjoner. Denne metoden er inspirert av stimulert hukommelse (stimulated recall). Dette er en metode hvor forskere studerer de kognitive prosessene til deltakerne, ved å la dem tenke tilbake på en tidligere situasjon ved hjelp av et visuelt hjelpemiddel (Lyle, 2003). Når elevene ser på oppgavebesvarelsene sine under intervjuet, kan de huske tilbake til undervisningen og reflektere rundt deres besvarelser.

3.5 Utvalg av elever og grupper

Siden målet er å undersøke elevers første møte med tekstprogrammering, blir tre klasser fra 8.trinn på en ungdomsskole studert. Læreren og skolen tar sikte på å introdusere elevene gradvis for tekstprogrammering gjennom skoleløpet, og ønsker at elevene møter dette så tidlig som mulig. Om forskningen ble gjennomført på et høyere skoletrinn kunne elevene allerede møtt tekstprogrammering, noe som ville hindret denne studien. Dette utvalget gjør det også mulig å undersøke hvorvidt elevenes kunnskap i Scratch fra barnetrinnet, er nok til å gjøre overgangen til Python allerede i 8.trinn.

Alle elevene som er til stede de tre dagene dataen blir innsamlet, vil kunne delta i undervisningen og besvare oppgavene. Et målrettet utvalg (Purposive sampling) vil gjøres sammen med faglæreren basert på observasjonene av elevenes løsninger, utfordringer og spørsmål i undervisningstimen. «Purposive sampling» innebærer å gjøre et utvalg som har data som kan besvare de gitte forskningsspørsmålene (Bryman, 2016, s. 408). Elevparene som oppfyller kriteriene ovenfor vil mot slutten av undervisningstimen bli spurt om de ønsker å delta på et intervju. I hver klasse vil to grupper bli utvalgt, slik at seks grupper intervjues totalt. Hver av disse gruppene vil bestå av to par elever fra samme klasse, hvor parene kjenner hverandre og løser oppgavene sammen. Gruppene vil om mulig bestå av et par med gutter og et par med jenter. For å få til et intervju med gode muligheter for datainnsamling, vil elevenes kjennskap og samarbeid vektlegges. Elevenes kjennskap til hverandre er viktig for deres trygghet og gir grunnlag for naturlige samtaler siden elevene tidligere har arbeidet sammen (Bryman, 2016, s. 508-510). Parene i hver gruppe kan i tillegg spille på hverandre, og sammenligne løsningene sine, noe som kan bidra til en mer naturlig samtale i gruppen. Bryman (2016, s. 416-417) vektlegger også betydningen av utvalgets størrelse i henhold til datainnsamling, hvor et større utvalg gir større validitet i studien. På den andre siden er dette en kvalitativ studie, og ved bruk av intervjuer er det viktig at forskere analyserer grundig.

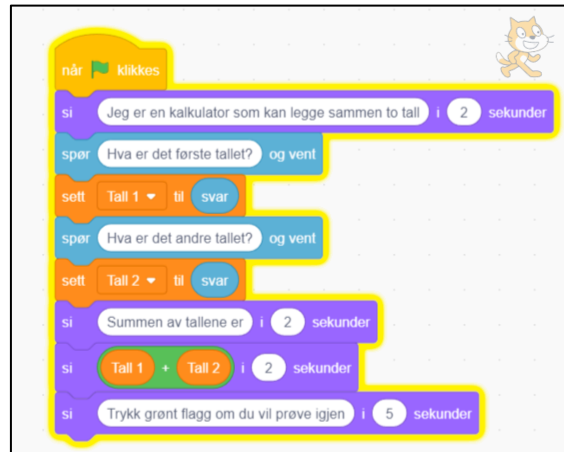
Bryman (2016, s. 481) trekker videre frem at intervju og transkripsjon er en tidkrevende prosess, og en time med intervju kan tilsvare opp mot seks timer med transkripsjonsarbeid. De seks utvalgte gruppene vil derfor bli intervju i opptil 20 minutter hver, og danner dermed et tilstrekkelig datagrunnlag, samtidig som at arbeidet rundt transkripsjonene og analysen er overkommelig.

3.6 Oppgavedesign

Under utarbeidelsen av oppgavearket, ble det sammen med faglærer gjennomgått hvilke erfaringer elevene hadde og hvilket nivå de var på i programmering. Elevene hadde aldri jobbet med tekstprogrammering, men hadde hatt flere undervisningsøkter med Scratch. Fokuset til faglærer hadde tidligere vært på tolkning og forståelse av koder, og programmering var ikke særlig brukt til å løse matematiske problemer. Siden elevene ikke hadde hatt om tekstprogrammering er de sannsynligvis i den instrumentelle initieringsfasen, hvor de hovedsakelig utvikler p-skjemaer (Buteau et al., 2020). Oppgavesettet er dermed ment som en utforskning av Python, og selv om oppgavene er satt i matematisk kontekst er det ikke forventet noe særlig matematisk utbytte av oppgavene.

Oppgavesettet som blir brukt i denne studien består av oppgaver som elevene har hatt tidligere i Scratch, hvor flere av oppgavene delvis er oversatt til Python. Oppgavene tilsvarer deres kunnskap i Scratch, og det matematiske innholdet er satt etter læringsmålene for 7.trinn, siden disse bør være kjent for elevene. Oppgavesettet består av fire oppgaver, hvor elevene vil få femten minutter til å gjennomføre hver oppgave. Den siste oppgaven er ment som en ekstraoppgave i tilfelle noen av elevene er effektive, eller om de står helt fast. For å belyse eventuelle svakheter i oppgavene eller metoden kan en pilottest være nyttig (Bell & Waters, 2018, s. 252). Gjennomføringen av pilottesten i den første gruppen viste ingen betydelige svakheter, og nivået virket passende. Ingen utbedringer ble derfor gjort i ettertid, og resultatene fra denne testen er også inkludert i resultatene.

3.6.1 Oppgave 1



Oppgave 1 er delt i fem deloppgaver a-e, og er designet ut ifra den første oppgaven (presentert til høyre) elevene hadde hatt tidligere i Scratch med faglæreren deres. I denne oppgaven skulle de opprinnelig tolke og forstå programmet. Oppgave 1 i oppgavesettet er stort sett en ren oversettelse av denne oppgaven til Python.

a) Hva gjør denne koden? ____

```
a = int(input("Velg et heltall: "))
b = int(input("Velg et annet heltall: "))
summen = a+b
print("Summen av tallene er", summen)
```

b) Hva tror dere disse kodedelene gjør?

```
a =     ____________________________________________________
input  _____________________________________________________
print  _____________________________________________________
int    _____________________________________________________
```

I de første deloppgavene skal elevene forsøke å identifisere hva de ulike kodedelene[3] gjorde. Det spørres om "a=" for å sjekke om elevene forstår at dette er en variabel. Videre blir det spurt om *input* og *print*. *Input* ber brukeren om å sette inn et tall, mens *print* skriver ut det som er gitt i koden. Dette spørres om for å sjekke om elevenes engelskkunnskaper kan hjelpe dem å til å forstå de ulike kommandoene. Det er også interessant om elevene ser koblingen mellom *input* og *print* i Python, opp imot «spørre»-blokken og «si»-blokken i Scratch. *Int*[4] er nok ukjent for elevene, men svarene deres kan likevel gi innsikt i hva de tror denne delen gjør. *Int* definerer det som settes inn som et heltall. Dersom det som settes inn er tekst gir programmet en feilmelding, og dersom det er et desimaltall blir det avrundet til nærmeste heltall.

c) Hvilke variabler finnes i den følgende koden?

```
a = int(input("Velg et heltall: "))
b = int(input("Velg et annet heltall: "))
differanse = a-b
print("Summen av tallene er", differanse)
```

Videre blir elevene bedt om å finne de ulike variablene i koden.

---

[3] Med kodedel menes en vilkårlig del av koden. Dette kan være enhver generell del som koden er bygd opp av.
[4] Int (Integer/heltall) er en av flere datatyper som kan tilegnes variabler i Python. Andre muligheter er float (floating number/flyttall eller desimaltall) og str (string/tekststreng).

d) Denne koden skal egentlig multiplisere sammen to tall, men om vi setter inn tallene 2 og 3 får vi 8, og ikke 6 som det burde være. Hva er feil, og hvordan bør koden endres for å fikse den? Hva tror du feilen gjorde?

```
a = int(input("Velg et heltall: "))
b = int(input("Velg et annet heltall: "))
produkt = a**b
print("Summen av tallene er", produkt)
```

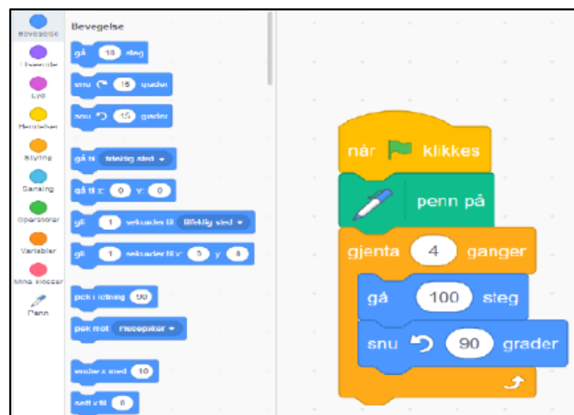
I koden over bes elevene om å finne feilen. Feilen er at det i linje 3 er to multiplikasjonstegn mellom variablene a og b, noe som medfører at a og b ikke multipliseres, men heller at a opphøyes i b.

e) Hvordan tror dere koden i **d)** kan endres til å finne produktet av tre eller flere faktorer?

Til slutt spør vi hvordan koden over kan endres til å finne produktet av flere tall. Her må elevene definere en ny variabel på lik måte som i linje 1 eller 2, samt legge til den nye variabelen i produktet.

3.6.2 Oppgave 2

Oppgave 2 er delt inn i deloppgave a-d. Oppgaven er igjen basert på et lignende program elevene har møtt tidligere i Scratch. Her skulle elevene opprinnelig tolke hvilken figur som tegnes, og deretter endre den til å tegne trekanter og femkanter. Koden under er oversatt til Python.



Denne koden tegner en geometrisk figur i Trinket.

```
import turtle as tegner #Importer turtle som fungerer som en penn i python
tegner.forward(150)
tegner.left(90)
tegner.forward(150)
tegner.left(90)
tegner.forward(150)
tegner.left(90)
tegner.forward(150)
```

a) Tegn figuren for hånd:

Målet med denne oppgaven er å sjekke om elevene forstår de ulike kommandoene *forward* og *left.* Siden blokkene i Scratch både har ordforklaringer, piler og benevninger, blir det interessant å se hvor hjelpsomme eller nødvendige disse er for elevenes forståelse. Videre må elevene tolke tallene 150 og 90 (antall steg og grader), siden disse ikke forklares i Python.

b) Lag et program i Scratch som tegner den samme geometriske figuren

c) Hvordan kan dere endre koden i Trinket a) til å tegne en trekant? Skriv den også ned her:

Deretter skal elevene tegne figurer i Python og Scratch. Siden elevene må legge inn penn-biblioteket i Scratch, kan det studeres om elevene ser sammenhengen mellom å importere penn i Scratch og tegner (Turtle) i Python. Når elevene tegner sin egen figur, må de huske å importere Turtle for å kunne tegne, og bruke riktig syntaks i form av punktum og parenteser. For å tegne en trekant må elevene fjerne de to siste kodelinjene i programmet fra deloppgave a), og samtidig endre gradene til 120 for å danne trekanten.

> **d)** Dere har tidligere brukt løkker (gjentakelser) for å gjøre koder i Scratch kortere. Hvordan kan dere gjøre Trinket-koden i a) kortere ved bruk av en løkke?
> *Tips: Dere kan søke opp dette på internett*

Denne oppgaven tester elevenes bruk av løkker. Løkker er et krevende konsept, og det er ingen forventninger om at elevene klarer dette på egenhånd. Derfor kan internett benyttes som hjelp i denne oppgaven. De som gjør oppgaven, kan derimot gi innsikt i hvilke utfordringer de møter når lager en løkke-struktur i Python.

3.6.3 Oppgave 3

I Oppgave 3 møter elevene igjen begrepene *int, input* og *print.* Her skal elevene vise hvordan programmet kan utvides, og hvilke hensyn de må ta i arbeid med syntaksen av programmet. Under intervjuene kan det igjen spørres om hvilke variabler elevene her identifiserer siden de nå har fått tydelige navn.

```
lengden = int(input("Hvor lang er bredden? " ))
bredden = int(input("Hvor lang er bredden? " ))
omkrets = lengden + bredden + lengden + bredden
print("Omkretsen er", omkrets)
```

> **a)** Hva vil dere utvide koden slik at den også regner ut areal?

> **b)** Dersom målene er i centimeter, hvordan tror dere dette kan legges til i koden?

> **c)** Lag den samme koden i Scratch

3.6.4 Oppgave 4

Avslutningsvis inkluderes oppgave 4 som en ekstraoppgave, og hver deloppgave utdypes derfor ikke. Tanken bak denne oppgaven å observere elevenes evne til å overføre programmet fra Python til Scratch. I denne overgangen møter de på if/else, og betingelsen om at alderen må være mindre enn 18. Elevgruppene som eventuelt løser denne oppgaven gir innsikt i hvordan de forstår disse konseptene, og hva som eventuelt er annerledes i Scratch og Python. I de siste deloppgavene blir de testet på å skrive programmet selv med riktig syntaks og strukturer.

```
alder = int(input("Hvor gammel er du? "))
if alder < 18:
  print("Du er ikke myndig")
else:
  print("Du er myndig")
```

a) Lag den samme koden i Scratch

b) Lag en egen kode i **Trinket** basert på den over for å sjekke om en person er i pensjonistalder (67år eller eldre). Skriv også koden ned her:

c) Ekstra utfordring: Hvordan kunne dere sjekket både myndighetsalder og pensjonistalder i samme kode? *Tips: Dere kan søke opp dette på internett*

3.7 Transkripsjon og analysemetode

3.7.1 Transkripsjon

Etter datainnsamlingen skal intervjuet transkriberes. Transkripsjon med lyd- og videoopptak er et nyttig verktøy for forskere, ettersom det utfyller det som ikke blir oppfattet av forskerne under selve intervjuet. Videre kan forskerne være mer til stede under selve intervjuet, siden videoopptaket og transkripsjonen kan gjennomgås flere ganger i ettertid. Transkripsjon krever derimot mye tid og bra utstyr for å ta opp lyd og video (Bryman, 2016, s. 479).

Verktøyet som skal brukes til transkripsjonen er programmet NVivo. NVivo er et nyttig program hvor lyd og video importeres inn i programmet, og gjør det enkelt å endre hastigheten eller pause filmen samtidig som transkripsjonen skrives rett inn i programmet. For å skape et rikere bilde av situasjonen vil tydelige gestikulasjoner eller kroppsspråk fra videoopptakene inkluderes i transkripsjonen. Det tas likevel forbehold om at elevenes kroppsspråk kan være vanskelig å overføre til tekst, og noe kan bli oversett i videoopptakene. Tabellen under forklarer de ulike symbolene i transkripsjonen. Transkripsjonene er sortert alfabetisk slik at gruppe 1 har navn som starter på forbokstavene A-D, gruppe 2 E-H og intervjuere er presentert i transkripsjonene som intervjuer 1 og intervjuer 2.

(…)	Pause fra elev/intervjuer i transkripsjonen
[…]	Start/slutt av utklipp fra transkripsjonen
«...»	Elev blir avbrutt i transkripsjonen
*	Henvisning eller peking av elevene

3.7.2 Analyseverktøy og koding

Å kode datamaterialet innebærer å identifisere ulike deler av dataen med unike merkelapper, slik at sammenhenger og konklusjoner enklere kan bli trukket (Bell & Waters, 2018, s. 271). NVivo som ble brukt til transkripsjonen har flere funksjoner som hjelper til med koding og analyse. Det er blant annet mulig å markere ulike sekvenser av transkripsjonen med spesifikke koder, slik at det i ettertid er enkelt å navigere datamateriale etter det er transkribert. I analysen vil en teoridrevet, deduktiv koding brukes. En slik tilnærming innebærer blant annet at tidligere teoretisk rammeverk brukes til å utvikle kodene, og kodene er gjerne festet til teksten (Crabtree & Miller, 1999).

For å kode transkripsjonene, anvendes rammeverket fra Qian og Lehman (2017), utviklet av Bayman & Mayer (1988). Kunnskap rettet mot det konseptuelle og strategiske, samt kunnskap om syntaks danner grunnlaget for kodingen. Ved å gjennomgå transkripsjonen og de innsamlede oppgavearkene hvor elevene møter utfordringer i oppgavene, kategoriseres og kodes disse basert på de tre kunnskapstypene nevnt ovenfor. Det vil deretter opprettes under-koder som videre kategoriserer unike og like utfordringer under de tre hovedkodene. Resultatene fra analysen kan også sammenlignes med funnene i Qian og Lehman (2017), noe som gjør det enklere å kategorisere utfordringene riktig. I analysen og diskusjonen vil også faktorene som presentert i 2.4.4 brukes til å tolke elevenes utfordringer.

Videre forsøkes det å analysere elevenes utfordringer i perspektiv av deres instrumentelle kunnskap (Buteau et al., 2020). At dette er et perspektiv tilsier at dette ikke kodes direkte, men heller fungerer som et bakteppe for å forstå elevenes tenkning og arbeid. Å kjennetegne elevenes instrumentelle kunnskap er en omfattende og stor prosess, men elevenes utfordringer kan derimot gi indikasjoner på hvilken kunnskap de innehar, og hvorvidt deres instrumentelle kunnskap hjelper dem i overgangen fra Scratch til Python. I tillegg til kodene ovenfor, vil også elevenes forsøk på å bruke erfaringer fra Scratch til å forstå Python kodes. Et eksempel kan være å se at *input* i Python kan kobles opp mot spørre-blokken i Scratch. I tillegg vil alle refleksjoner og sammenligninger elevene gjør mellom programmeringsspråkene kodes under dette. Kodingen av elevenes utfordringer vil også danne et grunnlag for å tolke hvilke av utfordringene som kan være forårsaket av overgangen i seg selv.

3.8 Forskningsetikk

Å handle etisk i forskning handler ifølge Bell & Waters hovedsakelig om å beskytte deltakere mot både fysisk og psykisk skade. For å beskytte deltakerne bør det derfor bes om godkjennelse til å gjennomføre forskningen. Deltakerne bør også informeres om forskningens mål og deres rettigheter, før de samtykker til å delta. I tillegg bør dataen som samles inn, behandles konfidensielt og sikkert. Dersom deltakerne blir lovet anonymitet skal de ikke kunne identifiseres (Bell & Waters, 2018, s. 64-71). Før forskningsprosjektet sendes derfor en søknad til Norsk senter for forskningsdata (NSD, vedlegg 7.1). I søknaden blir det gjort rede for hvilke data som skal samles inn, og hvilke metoder for datainnsamling som brukes. Når denne er godkjent, sendes et informasjonsskriv til elevene. Både elevene og deres foresatte må lese og samtykke og signere i informasjonsskrivet, og levere det før elevene kan delta. Kun elever som leverer et signert samtykkeskjema, og selv ønsker å delta vil intervjues.

En annen etisk problemstilling er forskning på elever i skolen. Selv om elevene ikke blir utsatt for fysisk skade, kan de oppleve psykisk eller sosial skade dersom undersøkelsen ikke tar hensyn til elevenes behov. Elever som utvelges kan være usikre på hvorfor de tas ut, og kan være bekymret for at andre skal tenke dårlig om dem og deres bidrag. Det er likevel en betydelig verdi i forskning på elevers arbeid, ettersom innsyn i deres tankemåter og utfordringer kan gjøre det mulig for lærere å tilrettelegge for bedre læring for fellesskapet (Felzmann, 2009). Elevenes anonymitet skal sikres, og sensitiv informasjon skal ikke bes om i intervjuet. Spørsmålene som stilles i intervjuet, skal hovedsakelig finne ut hvordan elevene tenker og arbeider. Det vil derfor ikke fokuseres på om elevene svarer rett eller galt, men elevenes svar kan bekreftes under intervjuet dersom de etterspør det. Disse tiltakene bør beskytte elevene og minimere sjansene for psykisk skade hos dem.

Til intervjuet vil video- og lydopptaker lånes av Universitetet i Agder (UiA). Etter datainnsamlingen vil disse leveres tilbake med slettet innhold. Før filene slettes, lastes de opp i en lukket OneDrive beskyttet av UiA. Dataen som samles inn, blir bare sett av forskerne i undersøkelsen. Alle elevene blir gitt pseudonymer i transkripsjonen, slik at de ikke kan identifiseres. I tillegg vil skolen, klassene og faglærer anonymiseres. Forskerne i studien har tidligere i studie signert en taushetserklæring ved UiA, og vil opprettholde denne ved å ikke dele sensitiv informasjon som dukker opp gjennom datainnsamlingen.

3.9 Oppgavens reliabilitet og validitet

3.9.1 Reliabilitet

Reliabilitet handler om en forskning kan gjennomføres på en lignende måte, og skape lignende resultater (Bryman, 2016, s. 41). Innenfor kvalitativ forskning, og særlig innenfor sosial forskning er det derimot vanskelig å gjenskape de samme sosiale kontekstene. For å styrke reliabiliteten er det viktig å være tydelig på hvilke metoder som blir brukes, og hvilken rolle forskeren har i studien (Bryman, 2016, s. 383).

Metodekapittelet er derfor godt utbrodert slik at synene, oppgavene og fremgangsmåten som brukes er så oversiktlig som mulig, og kan brukes videre i senere forskning. Inter-rater reliabilitet handler blant annet om at forskere burde være konsekvente og samstemte når de vurderer eller tolker subjektive data (Bryman, 2016, s. 157). For å skape en felles forståelse av hvordan transkripsjonene skulle bli analysert, er kodene basert på det teoretiske rammeverket og alle bidrag og forskerne har sammen drøftet tolkningene. På denne måten er en gjensidig forståelse for analyseverktøyet skapt, og tolkningene i studien blir mer konsekvente.

3.9.2 Validitet

Med validitet innenfor forskning vektlegges det om metodene som skal undersøke eller måle et konsept faktisk måler konseptet (Bryman, 2016, s. 158). Videre handler det også om konklusjonene i forskningen faktisk underbygges av resultatene (Bell & Waters, 2018, s. 141). Innenfor kvalitativ forskning trekker Bryman (2016) blant annet frem intern og ekstern validitet som viktige. Intern validitet går ut på hvorvidt det er en korrelasjon mellom årsakene og virkningene i forskningen, og om konklusjonene som trekkes underbygges av resultatene (Bryman, 2016, s. 41). For å identifisere de riktige virkningene brukes studien fra Qian og Lehman (2017), hvor blant annet det anerkjente og etablerte rammeverket til Bayman og Mayer (1988) ligger til grunn. Disse teoriene hjelper i identifiseringen av de ulike typene av kunnskap, og de tidligere funnene om utfordringer og faktorer som presentert i Qian & Lehman kan overføres til denne studien. Teorien om instrumentell skapelse (Buteau et al., 2020) er et omfattende perspektiv på hvordan elevenes utvikling skjer, basert på lærerens rolle og utvikling av instrumentell kunnskap. Videre kan dette antyde hvordan elevenes utfordringer innenfor disse kunnskapstypene oppstår. Ved å bruke slike utprøvde, og godt etablerte teorier styrkes validiteten i konklusjonene fra denne studien.

Bruken av stimulert hukommelse (stimulated recall) og intervju kan også styrke validiteten i oppgaven, ved å gi et bedre innsyn i elevenes tankeganger enn bare observasjoner. Det tas likevel høyde for at tolkningene og oppfattelsene av elevene kan være ufullstendige og unøyaktige. Transkripsjonene som tolkes er presentert i oppgaven, slik at andre kan skape egne tolkninger og sammenligne dem opp mot denne studien. Selv om noen mulige implikasjoner og årsaker for elevenes utfordringer vil gis i denne studien, er det derimot ikke nok kapasitet for å sjekke validiteten i disse. En egen studie ville derfor vært nødvendig for å undersøke hvilke årsaker som faktisk skaper utfordringene hos elevene.

Ekstern validitet refererer til om funnene i en studie kan generaliseres. Dette kan være en utfordring i kvalitativ forskning ettersom det ofte brukes kasusstudier og små utvalg (Bryman, 2016, s. 384). Målrettede utvalg er heller ikke godt egnet til å generaliseres, siden denne metoden prioriterer funn som svarer på forskningsspørsmålene (Bryman, 2016, s. 408). Resultatene som ble funnet i denne studien kan dermed ikke direkte generaliseres opp mot alle ungdomsskoler og elever. Derimot kan bruken av seks ulike kasus fra tre ulike klasser skape et grunnlag for å sammenligne de ulike gruppene opp mot hverandre. Om resultatene er like i flere av gruppene og ligner det andre forskere (Qian & Lehman, 2017) tidligere har funnet, kan dette styrke validiteten i denne studien. Teoriene, metodene og oppgavene er grundig presentert slik at andre kan gjenskape forskningen, og studere hvorvidt deres funn sammenfaller med funnene i denne studien.

En annen strategi for å styrke validiteten til konklusjonene og resultatene i forskning er ved triangulering. Triangulering innebærer å bruke flere observatører, teoretiske perspektiver, datakilder og metoder, og kan styrke validiteten i forskningen (Bryman, 2016, s. 386). I denne studien har to forskere samarbeidet om observasjonene og analysen, og brukt flere teorier til å forstå og analysere funnene. Dette kan skape mer nøyaktige og presise tolkninger og drøftinger, og reduserer de subjektive meningene forskerne kan ha.

4. Resultater og analyse

Her presenteres resultatene og analysen basert på det teoretiske rammeverket og metoden for analyse som er presentert tidligere. Resultatene presenteres gruppevis med forklaringer av funn innenfor syntaks, begreper og strategi, hentet fra utklipp av intervjutranskripsjonene.

4.1 Gruppe 1

I resultatene fra gruppe 1 oppdages det i allerede i transkripsjon 1 utfordringer knyttet til syntaks og begreper. Benjamin har programmering som valgfag, og sa at de andre heller burde bli spurt om de hadde utfordringer. Likevel viste Benjamin flere utfordringer, og det kan antydes at flere av konseptene i Python var nye for han. Alle elevene fortalte at de forstod hva programmet i oppgave 1 gjorde, men viser utfordringer knyttet til datatyper.

Int var ukjent for alle elevene unntatt Benjamin, men alle visste utfordringer knyttet til syntaksen. Dina hadde søkt dette opp på internett, og fant da ut at det var en forkortelse for integer (heltall). Da Benjamin ble spurt om hva som skjedde dersom *int* fjernes, svarte han at det da kunne vært hvilket som helst tall. Uten *int* vil *input* behandle det som settes inn som en tekststreng, ikke som tall slik Benjamin foreslo. Dette fremstår som en ufullstendig forståelse hos Benjamin.

[...]

Intervjuer 1: Var det noen av kodelinjene, eller kodedelene i oppgave b som var vanskelige å forstå? *Input*, *print* og *int*?

Benjamin: Det må du egentlig spør de andre om siden jeg har programmeringsvalgfag.

Albert: Nei, ehm (...), jeg skjønte det egentlig, men ikke den *int.* *Jentene nikker*

Intervjuer 1: Fant dere ut hva *int* var da? Søkte dere det opp eller?

Albert: Ja.

Dina: Ja, vi søkte det opp.

Intervjuer 1: Hva fant dere ut?

Camilla: Du må skrive inn heltallsverdi.

Intervjuer 1: Ok. Hvis vi hadde fjernet *int* fra koden, slik at det bare sto a=*input* osv., hva ville da skjedd tror dere?

Albert: Det kunne være hvilket som helst tall.

[...]

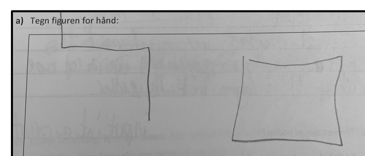
Transkripsjon 1, oppgave 1b

I transkripsjon 2 skulle elevene tegne en figur basert på tekstprogrammet de fikk presentert i oppgaven. Camilla og Dina hadde to tegninger, en trapp og et kvadrat. Elevene hadde oversett syntaksen, og fokuserte heller på kodedelene *forward* and *left*. Analysen viser at elevene oversatte disse engelske ordene, og deretter lagde en figur. Figuren ble likevel ikke riktig, ettersom de oversatte *left* til å gå mot venstre fremfor å snu mot venstre. Selv om de klarte å oversette riktig, hjalp det ikke elevene i å forstå hva selve koden gjør. At syntaksen og tallene her ikke hadde noen måleenheter eller benevninger, gjorde muligvis koden utfordrende å tolke. I Scratch spesifiseres det om tallene er «antall steg» eller «grader» (figur 4.1.1), og elevene trakk frem at det stod snu fremfor *left.* Snu-blokken gjorde det tydeligere for dem at pennen skulle snu seg, men til tross for pilen, var to av elevene usikre på hvilken retning pennen ville snu seg mot. Analysen antyder her at elevene hadde problemer med å finne en blokk i Scratch som tilsvarte kodelinjen i Python.

Figur 4.1.1: Snu-blokken i Scratch

[...]

Intervjuer 1: Da kan vi hoppe videre til oppgave 2. Den handlet om tegning, og dere fikk en kode som skulle tegne en geometrisk figur. Jeg ser at dere (jentene) har tegnet to tegninger. Om vi først ser på den trappen, hva tenkte dere når dere lagde den?



Figur 4.1.2: Tegning til Camilla og Dina

Camilla: Vi så egentlig bare på det som sto, og ikke på tallene.

[...]

Intervjuer 1: Etterpå skulle dere lage det samme programmet i Scratch, opplevdes det annerledes sammenlignet med koden her? Var noe annerledes i Scratch?

Camilla: Det sto jo ikke *left*, bare snu.

Dina: Også var det en sånn runding pil.

Intervjuer 1: På selve blokken? Var det til hjelp?

Camilla: Nei, vi skjønte ikke hvilken vei det gikk, men vi skjønte det etter hvert da.

Intervjuer 1: Kan dere se noen fordeler med å jobbe med slike oppgaver i Scratch fremfor Trinket?

Benjamin: Ja, altså Scratch er jo lettere å forstå.

Intervjuer 1: Hvordan er det lettere?

Benjamin: Fordi når du skal skrive i Trinket blir det veldig rart. I Trinket så er det jo tekst, og da er det mange rare kommandoer, mens i Scratch så står det akkurat hva som kommer til å skje.

[...]

Transkripsjon 2, oppgave 2a

I transkripsjon 3 blir elevene spurt om hva som skjer, dersom bare utregningen av areal skrives i koden uten å printe svaret. Camilla sa da at det ikke ville vært noe svar. Camilla viser her en ufullstendig forståelse av funksjonen *print* og hvordan programmet gjorde utregningene. Camilla viser her en konseptuell utfordring, og forstod ikke at programmet kjører uavhengig av om svaret ikke blir skrevet ut eller ikke. Benjamin tror det blir regnet ut, men er litt tvilende i svaret sitt.

[...]
Intervjuer 1: Hvis man ikke hadde tatt med den *printen* der, hva ville da skjedd for noe?
Camilla: Ehm (…), omkretsen er, også areal ...
Intervjuer 1: Ja, hvis vi hadde tatt vekk hele den siste linja, altså bare skrevet areal = lengde ganger bredde også sluttet koden. Hva ville skjedd da?
Camilla: Det ville ikke vært noe svar.
Intervjuer 1: Det hadde ikke vært noe svar?
Benjamin: Da hadde ikke arealet kommet, det hadde ikke stått.
Intervjuer 1: Hadde arealet fortsatt blitt regnet ut?
Benjamin: Ja, ehm, jeg tror det.
[...]

Transkripsjon 3, oppgave 3

I transkripsjon 4 blir elevene spurt om hvordan centimeter kunne legges inn i koden. Analysen viser her at Benjamin har kunnskap om syntaks i hvor cm skal legges inn i koden, og at det skal legges til som tekst (inni anførselstegn) i *print*-linjen.
Han har likevel ikke en god konseptuell forståelse av hvorfor plasseringen var viktig, ettersom han sa at det «blir rart» og ikke kunne svare direkte på hva som ellers ville skjedd.

[...]
Intervjuer 1: Jeg tror vi var litt innpå dette i timen, hvis vi hadde en figur som var for eksempel 5cm i bredde og 3cm i lengde, hvordan kunne man fått med det i koden?
Benjamin: Du kan legge til cm som tekst etter omkrets.
Intervjuer 1: Med omkrets, mener du i linje tre eller fire?
Benjamin: I linje 4, slik at det står omkrets i centimeter.
Intervjuer 1: Er det noe grunn til at du satt inn cm der (i linje 4) og ikke i linje 1 eller 3?

Benjamin: Ja, for hvis ikke ville jo bare alt blitt rart.

Intervjuer 1: Rart, ok. Hvis en kjører denne koden her så får en opp at du må skrive inn et tall, hvis man hadde skrevet inn 5cm fremfor bare 5, hva ville da skjedd?

Intervjuer 1: Det er et godt spørsmål.

Intervjuer 2: Hvis vi ser på selve oppgaven så står det igjen *int*, så hvis man begynner å legge inn bokstaver, har du noen tanker da hva som hadde skjedd om du både skrev inn tall og bokstaver, når den spør etter tall?

Benjamin: Kanskje den bare fjerner bokstavene?

Intervjuer 1: Okei, fint. Vi kan gå videre.

[...]

Transkripsjon 4, oppgave 3b

Etter at intervjuer 2 nevnte *int* for Benjamin, og spurte hva som ville skjedd om både tall og bokstaver ble satt inn foreslo Benjamin at programmet kanskje fjerner bokstavene. Benjamin forsto altså syntaksen rundt *int*, men ikke begrepet i seg selv. Denne utfordringen rundt *int* underbygges av transkripsjon 1 hvor Benjamin foreslo at hvilket som helst tall kan settes inn i en *input* dersom *int* ikke er med i koden.

4.2 Gruppe 2

I gruppe 2 blir det igjen funnet utfordringer innen syntaks og begreper, men også sekvenser der det vises noe forståelse av Python. Da elevene i transkripsjon 5 arbeidet med oppgave 1 skulle de i deloppgave a) finne feilen i en kode som multipliserte sammen to tall. Koden hadde nemlig to multiplikasjonstegn fremfor ett, noe som førte til at a ble opphøyet i b.
Analysen viser at dette var ukjent for alle de andre gruppene utenom for denne gruppen. Det virket som at elevene uten erfaring med Python, klarte resonnere frem matematisk hva de to multiplikasjonstegnene gjorde.

[...]

Intervjuer 1: Er det noe som gjør at noe annet skjer her?

Fredrikke: a gange b, er det det?

Intervjuer 1: To gangetegn?

Fredrikke: Ja.

Intervjuer 1: Kan dere tenke dere hvorfor vi får 8 da? I stedet for 6?

Henriette: Å, kanskje hvis den blir sånn 2 i tredje eller noe sånt? Så da blir det 2 gange 2 gange 2.

Intervjuer 1: Ok. Hvordan kunne vi rettet denne her da hvis vi ønsker å multiplisere heller?

Fredrikke: Skrive a gange b?

[...]

Transkripsjon 5, oppgave 1d

I transkripsjon 6 presenteres elevenes forsøk på å legge inn centimeter i koden. Analysen viser at det oppstod konseptuelle utfordringer, ved at elevene ikke forstod betydningen av *int*. Videre viser analysen at problemer med det konseptuelle ga problemer med syntaksen som en konsekvens. Elevene viste usikkerhet rundt hvor de skulle plassere centimeter og hva som skjer dersom vi skriver centimeter direkte inn i *input*. Elevene fikk da en feilmelding, og hadde ikke nok instrumentell kunnskap til å forklare syntaksen i programmet eller forstå betydningen av begrepet *int* i programmet.

[...]

Intervjuer 1: Var det noen som prøvde å bare legge det inn i *input*en når man skrev programmet? Altså at man satt inn da for eksempel 5 centimeter som lengde?

Fredrikke: Ja, men det kom så error eller noe.

Intervjuer 1: Det kom error ja. Kan dere tenke dere om hvorfor det kom error når dere prøvde å sette inn 5 centimeter?

Fredrikke: Fordi det står i centimeter enda? Er det noe med det?

Intervjuer 1: Ja, husker dere hva *int* stod for?

Fredrikke: Åja, ja. Så du måtte bytte *int* med centimeter?

Intervjuer: Ja, på grunn av den *int*-en der, så kreves et eller annet av svarene du legger inn.

Fredrikke: Mhm.

Intervjuer 1: Hva tror dere den krever for noe?

Fredrikke: Centimeter eller noe som for at det ikke blir heltall, men at du kan skrive bokstaver også. Jeg vet ikke.

[...]

Transkripsjon 6, oppgave 3b

Videre i transkripsjon 7 viser analysen at elevene i gruppen hadde konseptuelle utfordringer med å definere hva som kunne være en variabel i programmet. Elevene fikk hint fra intervjuer om at omkrets muligens kunne være en variabel, men elevene var usikre rundt variabelbegrepet, selv om de uttrykker at det innebærer noe som er «ukjent». Elevene reflekterte også om hvorvidt omkretsen er en variabel i oppgave 3, men manglende konseptuell og instrumentell kunnskap skaper utfordringer rundt variabelbegrepet.

[...]

Intervjuer 1: Hva er det som gjør at omkretsen her er annerledes sammenlignet med differansen i 1c? Der var differanse lik a minus b.

Fredrikke: Ja?

Intervjuer 1: Men i oppgave 3 så har vi egentlig omkrets er lik lengde pluss bredde pluss lengde pluss bredde.

Fredrikke: Her står det jo a minus b.

Intervjuer 1: Er det det som gjør at det blir en variabel?

Fredrikke: Jeg vet ikke, nei?

Intervjuer 1: Det er ikke noe riktig eller feil svar her.

Henriette: Omkretsen er jo også på en måte ukjent da, siden man må gange (...) ukjent eller man må plusse ukjent med ukjent med ukjent med ukjent på en måte «...»

Fredrikke: Ja.

Henriette: Og det «...»

Fredrikke: Også her må du ta ukjent minus ukjent.

Intervjuer 1: Så siden både lengden og bredden er ukjent, så er omkretsen også på ukjent?

Fredrikke og Henriette: Ja.

[...]

Transkripsjon 7, oppgave 3

Mot slutten av intervjuet ble også elevene spurt om oppgave 4, der de skulle bruke forgreininger *if/else* til å avgjøre om en var myndig eller ikke. I oppgave 4 skulle elevene endre koden slik at programmet spurte om en person var pensjonist eller ikke. Elevene hadde gjort en lignende oppgave tidligere i Scratch, og det virker som at elevene brukte sin instrumentelle kunnskap fra Scratch og matematiske kunnskap om ulikheter til å endre koden.

[...]
Intervjuer 1: Dere hadde gjort oppgave 4 i Scratch før altså?
Fredrikke og Henriette: Ja.
Intervjuer 1: Ja. Hvis dere tenker litt, hvordan kan dere endre koden i c) til å gjøre slik oppgave b, altså sjekke om en person er pensjonist eller ikke?
Fredrikke: If alder 67 ikke 18.
Henriette: Og kanskje endre du er ikke myndig til du er «...»
Fredrikke: Ja, bytte det til at du er ikke pensjonist eller du er det.
Intervjuer 1: Hvis vi hadde skrevet if alder er lik 67, hva hadde da skjedd hvis vi skrev inn 70?
Henriette: Hmm (…) Du er pensjonist? I stedet for du er myndig så står det at du er pensjonist.
[...]

Transkripsjon 8, oppgave 4a

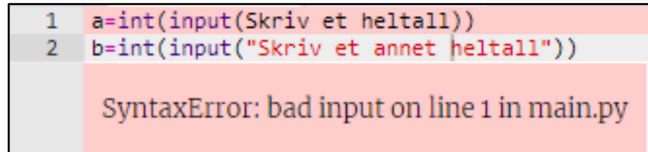
Etter elevene hadde diskutert rundt hvordan programmet i oppgave 4 kunne endres, spurte intervjuer 1 mer om betingelsen i transkripsjon 9. Her viser analysen at de ikke helt forsto hva selve betingelsen gjør, dersom den settes lik en verdi. De forsto ikke hva programmet utfører når ulikheten blir til et likhetstegn, nettopp at betingelsen og likhetstegnet påvirker programmet ut ifra verdiene som settes inn.. Elevene har ikke kunnskap om at operatoren for likhetstegnet i Python er «==», men dette ble ikke vektlagt i intervjuet.

[...]
Intervjuer 1: Om en hadde byttet ut den der som går sånn *viser ulikhetstegn*, det krokodilletegnet til et likhetstegn heller, altså hvis alder er lik 18 og du fremdeles hadde skrevet inn 17. Hva hadde da blitt skrevet tror dere?
Fredrikke: Har ikke peiling.
Intervjuer 1: Eller ville koden ha stoppet kanskje da?
Henriette: Ja det kan godt være.
Fredrikke: Ja, tror det, eller så har det stått noe feil.
Henriette: Men 17 er lik 18 gir jo ikke helt mening «...»
Fredrikke: Ja, kanskje det.
Henriette: De er jo helt forskjellige tall?
Intervjuer 1: Så da hadde ikke den if-en fungert da? Men tror dere else-en hadde blitt kjørt da?
Henriette: Kanskje.
[...]

Transkripsjon 9, oppgave 4

4.3 Gruppe 3

Gruppe 3 var en av de mest aktive gruppene under intervjuene. Allerede før gjennomgangen av oppgavene var de tydelige på at syntaksen i Python var utfordrende. Katrine nevnte at de ikke fikk fargen på strengene i oppgave 1 da de forsøkte å skrive dem inn. Dersom teksten som skrives i *input*en-koden ikke settes i anførselstegn, vil den ikke få en farge. Dersom programmet kjøres, vil det stoppe opp og sende ut en SyntaxError[5]. Disse fargene og feilmeldingene er lagt inn i programmeringsspråket for å tydeliggjøre feil i koden, men er ikke hjelpsomme for elevene uten øvelse eller erfaring i tekstprogrammering. Line sa at det sikkert var et mellomrom for mye, og forstod ikke at det var manglende anførselstegn som var feilen.



Figur 4.3.1: Slik indikeres fargene i Trinket med tilhørende feilkode dersom anførselstegn i input glemmes.

[...]

Intervjuer 1: Var dette helt nytt, eller har dere hatt dette før?

Julian: Det der har vi ikke hatt før.

Katrine: Vi har jo hatt Scratch før.

Ivar: Jeg følte vi hadde dette på barneskolen.

Katrine: Men det var litt av det som var kjent.

Line: Jeg tror det var litt meningen.

Katrine: Men det var noen ganger de ikke fikk sånn farge.

Intervjuer 1: Når man skrev koden selv?

Katrine: Ja.

Line: Ja i den der liksom. *Peker på Python-koden i figur 4.3.1 (Oppgave 1a) *

Ivar: Ja, når man skrev i Trinket.

Katrine: Ja, når man skrev den inn så funka det ikke.

Line: Det var nok for mye mellomrom eller noe.

[...]

Transkripsjon 10, oppgave 1a

[5] En SyntaxError indikerer at det er en skrivefeil i koden som programmet ikke klarer å lese. I figur 4.3.1 mangler det anførselstegn i linje 1, og programmet stopper opp da den ikke klarer å lese teksten som en streng.

Alle elevene forsto hva programmet i deloppgave a) gjorde ved å se på linjen *a+b=summen*, og deres manglende kunnskap om syntaks hindret dem dermed ikke i deres forståelse av programmet. Når de ble spurt om de ulike kodedelene i oppgave 1b), hadde de flere interessante forslag.

[...]
Intervjuer 1: Så var det å tolke disse ulike kodedelene.
Line: Ja, det var litt vanskelig å gjøre.
Intervjuer 1: Hvilken var vanskeligst?
Katrine: Interact, er det ikke det?
Line: *Int,* og egentlig *input* også.
Line: *Print* er det liksom det som sies.
Katrine: Ja, det som blir "printet", eller det blir ikke printet, men «...»
Line: Det som vises «...»
Katrine: Vi tenkte *input* er det som blir med, men vises ikke eller
Intervjuer 1: Hva var det dere tenkte *int* skulle stå for?
Ivar: *Input*
Line: Hjelpelæreren sa det var interact
Ivar: Åja, interact.
Intervjuer 1: Interact, okei.
Katrine: Okei, det er ikke interact *elevene ler*
[...]

Transkripsjon 11, oppgave 1b

Definering av datatype (*int*) var nytt for alle elevene, og var det elevene syntes var vanskeligst. Ivar trodde først at det var en forkortelse for *input*, mens Line og Katrine mente det sto for interact. Ivar hadde nok konkludert med at det var en forkortelse på egenhånd, men jentene sa at en hjelpelærer fortalte dem at det var interact. Det er usikkert hvor hjelpelæreren hadde fått dette fra, men mest sannsynlig var dette gjetting. Hadde dette ikke blitt oppdaget, kunne elevene dannet seg en misoppfatning rundt betydningen av *int.*

Elevene nevnte også *input* som litt vanskelig, noe analysen også tilsier. Elevene ble blant annet utfordret på hva som ville skjedd dersom *int* ble fjernet fra koden. Om *int* fjernes adderes ikke lengre tallene, og de behandles som to tekststrenger som bare settes sammen.

Dermed konkluderte elevene i transkripsjon 12 at *input* setter eller "putter" tallene sammen. Selv om elevene opprinnelig brukte *a+b=summen* til å forstå hva koden gjorde, så de da bort fra det, og ga heller *input* en funksjon den ikke har. Det antydes at elevene oversatte det engelske ordet *input* til "putt inn". Det virker som at oversettelsen og deres oppfatning av ordet førte dem i feil retning.

[...]
Intervjuer 2: Så *int* er egentlig integer. Det betyr at det må settes inn et heltall. Men hvis vi hadde fjernet *int*, og bare hatt *input*. Kan dere tenke dere hva som hadde kommet ut dersom dere kjørte koden da?
Line: Komma.
Ivar: Et halvtall. *Elevene ler*
Julian: Desimaltall.
Intervjuer 1: Hvis vi hadde satt a som 1 og b som 2 ville vi fått 12.
Katrine: Åja «…»
Ivar: Åja, så det blir pluss.
Line: Nei, det blir (...)
Ivar: Nei, det blir innsetting, setter de sammen.
Katrine: Ja, fordi inputting bare putter dem sammen.
Line: Åja fordi det setter inn 1 og 2, også (...)
Katrine: Ja, det settes sammen. *Klapper hendene sammen*
Ivar: Ja, det settes sammen.
[...]

Transkripsjon 12, oppgave 1b

Analysen viser også gruppen hadde utfordringer med variabler. Da elevene i transkripsjon 13 ble de spurt om hvilke variabler de fant i oppgave 1c), trakk de frem at variabler er oransje i Scratch, og at de i Python er røde Elevene forsøker her å bruke tidligere erfaring fra Scratch til å forstå Python, men hvor den instrumentelle kunnskapen om variabler fra Scratch ikke er direkte overførbar. Siden tekststrengene blir fargelagt, tenkte elevene at de måtte være variabler, og utelukket *a* og *b* som egentlig er variablene.

Denne fargekodingen er opprinnelig lagt til for å gjøre koden mer oversiktlig, men skapte i dette tilfelle forvirring blant elevene. Siden jentene tenkte at bare de røde delene var variabler, valgte dem derfor bort *differanse* som variabel. Det kan virke som jentene dannet seg en misoppfatning om at alle røde kodedeler i Python er variabler, en oppfatning som ikke er koblet til elevenes matematiske kunnskap om variabler. Guttene gjettet at *differanse* var en variabel, men hadde ikke noe forklaring på hvorfor det måtte være det.

[...]
Intervjuer 1: Da tror jeg vi går videre til oppgave c. Der spurte vi dere dere om dere kunne finne noen variabler. Vet dere hva en variabel er for noe?
Alle elevene: Ja.
Ivar: Det er de oransje i Scratch.
Line: Ja de oransje.
Katrine: *Peker på Python-koden* Her er de røde.
Intervjuer 1: Her er de røde?
Katrine/Line: Ja, det trodde vi i hvert fall.
Line: For eksempel disse to. *Peker på "Velg et heltall: " og "Velg et annet heltall: " i rødt i de to første *input* linjene (Se figur 4.3.1) *
Katrine: Ja, de med heltall.
Line: Ja, der du skal skrive inn heltall.
Intervjuer 1: Var det noe som fikk dere til å tenke at disse var variabler?
Line: At liksom her skal du skrive inn tall, og de tallene er variablene, for det er to variabler som blir regnet sammen. Så det er liksom de røde.
[...]
Intervjuer 1: Hva med differanse da? Differanse=a-b, tror dere differanse er en variabel?
Julian: Vi skrev det, men jeg vet ikke.
Intervjuer 1: Var det en tanke bak det gjettet?
Julian: Jeg trodde bare det var det.
Ivar: Det var en tanke bak det, men jeg husker den ikke.
[...]

Transkripsjon 13, oppgave 1c

Denne gruppen klarte ikke å finne ut hva de to multiplikasjonstegnene i oppgave 1d) representerte, men alle antok at dette var feilen. Dette tyder på at elevene klarer å finne en syntaksfeil, men deres manglende forståelse av syntaksen og operatoren hindret deres matematiske forståelse. Katrine tenkte at de doble stjernetegnene kunne medføre noe lignende som at minus og pluss blir minus. Det virker som eleven forsøkte å anvende tidligere tilegnet matematisk kunnskap, inn i dette instrumentet (Python) som de ikke forstår, uten at det gir noe mening.

[...]

Intervjuer 1: Dere (Line og Katrine) hadde gjort noe interessant her. (Figur 4.3.2)

Line: Vi tror feilen er to gangetegn, men vi vet ikke helt (...)

Katrine: Vi vet ikke helt hvordan det henger sammen, men det er en mulighet.

Figur 4.3.2: Line og Kathrines forklaring

Julian: Vi trodde også det var fordi det var to gangetegn.

Intervjuer 2: Ja, men dere vet ikke hva de to gangetegnene sammen gjør?

Alle elevene: Nei.

Katrine: Nei, vi lurte på fordi hvis du har pluss og pluss så blir jo det pluss, og minus og pluss blir minus ikke sant, så vi tenkte det var noe sånt.

Line: Ja, eller du tenkte.

Katrine: Ja, jeg tenkte. Men det var ikke det.

[...]

Transkripsjon 14, oppgave 1d

I oppgave 2 viser elevene at de forsto begrepet løkker, og lagde på eget initiativ en kode i Scratch med en løkke. De trekker frem at de fikk ideen til å gjøre dette siden de hadde gjort det før, og brukte dermed tidligere instrumentell kunnskap. Når de ble spurt om fordeler og ulemper ved de to programmeringsspråkene nevnte de at gjentakelse-blokken i Scratch var en fordel. Videre trodde de at det sikkert er mulig i Python også, men at de ikke klarte det. Jentene forsøkte faktisk å lage en løkke i Python etter å ha søkt dette opp på internett. De endte opp med en ufullstendig while-løkke (figur 4.3.3), men forsøket deres tydeliggjør utfordringene rundt å lage løkker i Python.

Figur 4.3.3 - Katrine og Line sin while-løkke

Elevenes mangler på instrumentell kunnskap om while-løkker, betingelser og innrykk gjorde denne løkken mangelfull. Dette kan derimot tolkes som et tegn på strategisk kunnskap, da elevene erkjente verdien av løkker i Scratch, og aktivt ønsket å bruke det i både Scratch og Python.

[...]

Intervjuer 2: Brukte dere Trinket-koden som hjelp til å tegne i Scratch eller var det noe dere kunne fra før?

Katrine: Vi brukte sånn gjenta, også tok vi sånn inni fordi vi hadde gjort det før.

Line: Ja vi brukte gjenta, også sånn penn, også gå frem, også snu og gjenta

Intervjuer 1: Ser dere noen fordeler eller ulemper med de to forskjellige måtene å skrive på?

Line: *Peker på Python-koden i oppgave 2* Du må skrive dette mange ganger

Katrine: Det gikk, men du må (...)

Julian: I Scratch kunne man bare bruke sånn gjenta til å gjøre det.

Katrine: Men det er nok en måte (I Python), men vi klarte det ikke.

Intervjuer 1: Vet dere hva det heter for noe, denne gjenta funksjonen?

Julian: Løkker.

[...]

Intervjuer 1: Fikk dere jobbet noe med d) også?

Line: Vi var inne på det, men fikk for lite tid så da (...)

Katrine: Veldig vanskelig da.

Intervjuer 2: Ja, så dere har brukt en while løkke (…)

Intervjuer 1: Hva fant dere når dere søkte opp løkke?

Line: Vi fant egentlig ikke så mye.

Katrine: Det sto liksom veldig mye, om løkker, men det var ikke så lett.

Line: Ja, det er lettere i Scratch.

Katrine: Ja, for når jeg trodde det var løkke så sto det sånn parentes, også, nei, jeg vet ikke, ikke lett.

[...]

Transkripsjon 15, oppgave 2b

I transkripsjon 16 viser analysen igjen at elevene har problemer med *int*. Selv om de fikk forklart hva *int* var og hvilken funksjon den hadde tidligere, hadde de flere utfordringer i oppgave 3b). Line og Katrine klarte oppgaven, men var usikre på hva som ville blitt skrevet ut dersom tekst ble skrevet i *input*-linjene. Ivar og Julian forsto ikke helt hvordan programmet fungerte. Julian nevnte at dersom en hadde skrevet inn cm bak et tall når en regner ut arealet, ville programmet skrevet ut cm i andre. Eleven tenkte her at programmet vil regne ut i kvadratcentimeter på egenhånd. Julian viser dermed en mangel på konseptuell kunnskap om programmets muligheter og begrensninger, og tilegnet programmet en funksjon det ikke egentlig har.

[...]
Intervjuer 1: Hvis vi hadde lagt til cm her *peker på linjene med *input**, ville det endret noe?
Ivar: Ja, jeg tror det.
Intervjuer 1: Hva ville det endret, tror dere?
Julian: Nei, det går ikke.
Ivar: Nei, for da blir det for mye.
Line: Men man skriver det vel der *peker på slutten av print-linjen*
Katrine: Ja, også inni parentes, bak der, bak omkretsen.
Intervjuer 1: Bak omkretsen?
Line og Katrine: Ja.
Julian: Ja, så bare cm.
Line: Ja, så omkretsen er omkrets, også centimeter.
Katrine: I sånn parentes, nei vent, sånn. *lager anførselstegn med fingrene*
Line: Sånne anførselstegn.
Intervjuer 1: Hva tror dere ville skjedd dersom en satt inn cm i *input*en?
Julian: Da hadde det stått centimeter.
Katrine: Da hadde det stått centimeter pluss.
Julian: Centimeter i andre hvis det var areal, hadde det ikke? For det blir sånn bredde ganger lengde.
Katrine: Kanskje det ville stått centimeter flere ganger.
Intervjuer 1: Tror dere programmet ville tillatt at oss å skrive inn centimeter?
Katrine: Nei.

Line: Nei, det er sant, fordi programmet var veldig sånn at hvis du skrev litt feil så ville det ikke gjøre noen ting.

Julian: Så hva skal man ta i stedet for da?

Katrine: Du skal bare ikke ta det inn.

Line: Ja, du må bare ikke skrive det inn her *peker på *input*-linjene*

[...]

Transkripsjon 16, oppgave 3b

4.4 Gruppe 4

Analysen viser at elevene i gruppe 4 også møtte på utfordringer om syntaks og ulike begreper i programmeringen. Ved gjennomgang av oppgave 1 under intervjuet ble elevene spurt om deres kunnskap rundt variabler, der Martin definerer det som en «spørsmålsboble» i transkripsjon 17. Analysen viser her at eleven har instrumentell kunnskap fra Scratch om variabler, men ikke konseptuell forståelse om variabler. Martin gjettet derfor at alle kodedeler med spørsmål er variablene, og ikke a og b som egentlig er variablene. Dette forsøket på å overføre instrumentell kunnskap i Scratch til Python blir dermed feil.

[...]

Intervjuer 1: Kunne dere forklart litt hva dere tenker når dere hører ordet variabel?

Martin: Jeg føler det er noe sånn spørsmål boble eller noe.. Jeg vet ikke

Intervjuer 1: Spørsmål boble?

Nils: I Scratch så kan du jo legge inn sånn variabel på en måte i programmet.

Intervjuer 1: Hvilke av disse kunne vært variabler, altså i den koden vi ser?

Martin: Velg et tall tror jeg.

Intervjuer 1: Velg et tall?

Martin: Eller et heltall.

[...]

Transkripsjon 17, oppgave 1

I transkripsjon 18 viser elevene utfordringer med syntaksen i programmet. Analysen viser her at elevene virket usikre når de ble spurt om hva ville skjedd dersom *int* ble fjernet fra koden. Intervjuer 1 ga noen hint for at elevene å teste elevene i forståelsen av int. Martin foreslo deretter at det ikke handlet om addisjon av variablene, men at tallene ble lagt sammen. Siden dette var et ledende spørsmål er det vanskelig å analysere dette riktig, men stillheten kan tyde på at elevene sliter med å forstå datatyper og betydningen dette har i programmet.

[...]
Intervjuer 1: Om vi fjernet integer her, og kuttet den helt ut ifra koden slik at det bare står *input*.
Pauline: Ja.
Intervjuer 1: Og deretter setter inn tallene 1 og 2, så hadde vi fått 12. Kan dere tenke dere til hvorfor vi hadde fått det? (...)
Intervjuer 1: Dere kan tenke litt på den, det kan være litt vanskelig.
Pauline: Fordi da er det ikke et heltall, da er det. Eller ...?
Intervjuer 1: Hva tror dere koden gjorde, siden vi fikk 12 i stedet for 3?
Martin: Du kan få lagt sammen tallene slik at det blir sånn 1 og 2, sånn 12.
[...]

Transkripsjon 18, oppgave 1

I oppgave 2 (transkripsjon 19) skulle elevene tegne en figur ut ifra koden de var gitt. Analysen viser at Oda og Pauline opprinnelig ikke forsto syntaksen av programmet, siden de tegnet en trapp. Oda og Pauline tolket programmet som at pennen først går 150 frem, og deretter går 90 til venstre. Det fremstår som at elevene ser på skjermen og retningene som statiske, og ser bort ifra at pilen er den som flytter og snur seg. Elevene viste dermed konseptuelle utfordringer i forståelsen av skilpaddens bevegelser. Denne gruppen brukte i tillegg tallene som størrelser til å tegne opp trappen, slik at den gikk litt lenger frem enn den gikk til venstre.

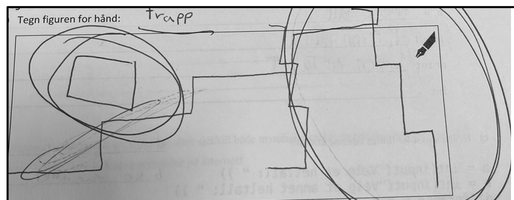
[...]

Oda: Vi (...) *Oda og Pauline ler litt av sin tegning*

Pauline: Først trodde vi det var en trapp *Viser tegningen sin*

Intervjuer 1: En trapp.

Pauline: Og så trodde vi gikk at den sånn *peker på trappen* men så fant vi ut at den ble sånn *peker på firkanten*



Figur 4.4.1 – Oda og Pauline sin tegning fra oppgave 2

Intervjuer 1: Hvordan tenkte dere når dere tegnet den?

Pauline: Vi tenkte liksom 150 opp fordi fram liksom ... Også *left* også opp igjen sånn *viser igjen på tegningen sin*

Intervjuer 1: Hvorfor går den kortere til venstre enn fremover?

Pauline: Fordi tallet er mindre liksom

[...]

Transkripsjon 19, oppgave 2

4.5 Gruppe 5

I analysen av gruppe 5 er det igjen utfordringer knyttet til syntaks og begreper. Elevene klarte å forklare hva programmet i oppgave 1 gjorde, og kjente spesielt igjen $a+b$ som addisjon fra matematikk. Ragnar trakk frem at de ikke visste hva *int* var, og at de måtte søke dette opp på internett. Når de ble spurt om hva de trodde ville skjedd om *int* ble fjernet fra koden hadde de ingen forslag. Selv om de klarte å finne betydningen av ordet integer (heltall), hadde de ikke noen forståelse av begrepet, og hvorfor *int* var nødvendig.

[...]

Intervjuer 1: Vi kan begynne på oppgave 1. Var det noe som var spesielt vanskelig der?

Ragnar: Den var fin egentlig, men det var noen ting som vi slet med. Vi visste ikke at *int* var et heltall, og at det betydde heltall.

Intervjuer 1: Hvordan fant dere ut av det da?

Ragnar: Det ble google da, så brukte vi translate og fant ut at det het integer på engelsk.

Intervjuer 1: Var det noe som avslørte hva koden gjorde?

Stine: Det var vel $a+b$ egentlig.

Intervjuer 1: Hva tror dere ville skjedd om man fjernet *int*? *Ingen respons fra elevene*

[...]

Transkripsjon 20, oppgave 1a-c

Elevene oppdaget at feilen i oppgave 1d) var to multiplikasjonstegn, og klarte å finne og rette en syntaksfeil. De hadde likevel ingen idé om hvorfor disse to multiplikasjonstegnene ga et annet svar enn ønsket, og selv om de bekreftet at de hadde hørt om potenser før, gjenkjente de ikke denne matematiske operasjonen i koden. Det kan dermed virke som at elevenes deres matematiske kunnskap er høyere enn deres instrumentelle kunnskap om syntaksen og denne måten å skrive eksponenter på, noe som hindret deres forståelse av programmet.

[...]
Intervjuer 1: I oppgave d), så spurte vi om denne koden som skulle multiplisere to tall, men som skrev ut 8 i stedet for 6, når den multipliserte 2 og 3. Fant dere feilen her?
Stine: Nei, vi kom ikke så langt.
Quang: Nei, vi kom ikke til den vi heller.
Stine: Det er to gangetegn.
Intervjuer 1: Ok, to gangetegn. Kan dere da tenke dere hvorfor vi da får 8, og ikke 6?
Elevene tenker
Quang: Jeg vet ikke.
Elevene rister på hodet
Intervjuer 1: Nei? Har dere hørt om potenser før?
Quang: Ja!
Intervjuer 1: Ja, okei.
[...]

Transkripsjon 21, oppgave 1d

Alle elevene i denne gruppen hadde også opprinnelig tegnet en trapp i oppgave 2. Ved å tegne linjer frem og venstre lagde de dermed en trapp, og tolket *left* som å gå mot venstre fremfor å snu seg mot venstre. Stine og Tiril byttet derimot til et kvadrat etter de prøvde seg i Scratch. Selv om koden sto i Python, valgte de heller å gjenskape koden i Scratch. Elevene brukte dermed heller et instrument de var kjent med fra før, og unngikk å arbeide i tekst-programmet, til tross for at kodelinjene ser ulike ut i Scratch. Ragnar og Quang nevner videre i transkripsjon 22 at de ikke tenkte på tallene som steg eller grader. De virket til å oppfatte tallene som lengder, og skapte dermed en trapp med høyde på 150, og lengder på 90. Siden Python ikke spesifiserer hva disse tallene er, i motsetning til Scratch, fremstår dette som en utfordring for denne gruppen.

[...]

Intervjuer 1: Jeg ser at dere (Stine og Tiril) begynte på en trapp. Hva tenkte dere her?

Stine: Det var at vi først gikk litt frem, også snudde den litt til venstre, også frem igjen.

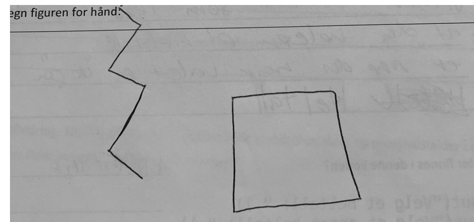
Intervjuer 1: Okei, men så endret dere til et kvadrat?

Tiril: Mhm.

Stine: Ja, det var etter vi prøvde i Scratch.

Intervjuer 1: Okei, dere prøvde i Scratch ja (...)

Intervjuer 1: Jeg ser at dere (Ragnar og Quang) også tegnet en trapp? Hva tenkte dere når dere lagde den?



Figur 4.5.1: Tegningen til Stine og Tiril

Ragnar: Nei, vi tenkte ikke at var liksom 150 steg hvis du skjønner, men vi tenkte at det var, 90, også 150, så det ble liksom frem 150, også 90 *peker til venstre*, også ja.

[...]

Transkripsjon 22, oppgave 2a

I oppgave 2b skulle elevene endre programmet til å tegne en trekant. Ragnar og Quang hadde skrevet riktig kode, men fikk den ikke til å kjøre. De hadde nemlig glemt å importere *turtle* som tegner, og de fikk NameError[6], som forklarte at tegner ikke var definert (se figur 4.5.2). Selv om elevene fikk denne feilmeldingen, og hadde den ferdige koden på et kvadrat fra forrige oppgave klarte de ikke å finne og rette denne feilen på egenhånd. Denne feilmeldingen innebærer at elevene glemte å definere en variabel og importere biblioteket Turtle. Siden disse er konseptuelle feil er det forståelig at de ikke klarte å utrette disse feilene uten hjelp, og feilmeldinger kan fremstå kryptiske og lite hjelpsomme for nybegynnere. Elevene prøvde seg også frem i Scratch, og klarte der å lage denne trekanten, selv om elevene måtte importere datapakken penn i Scratch for å kunne tegne. Kombinasjonen av tidligere tilegnet instrumentell kunnskap og at blokken «penn ned» inkluderer et bilde av en penn, kan ha gjort Scratch enklere for elevene å utforske og programmere.

main.py shapes.py

```
1 tegner.forward(150)
2 tegner.left(120)
3 tegner.forward(150)
4 tegner.left(120)
5 tegner.forward(150)
```

tegner is not defined. Did you mean something else?

NameError: name 'tegner' is not defined on line 1 in main.py

Figur 4.5.2: Gjenskapelse av Ragnar og Quang sitt program med feilmeldingen deres.

[6] NameError indikerer at en variabel eller funksjon som brukes ikke er definert i programmet. I figur 4.5.1 hadde elevene ikke definert *turtle* som tegner, og programmet stoppet derfor da den ikke gjenkjente tegner i koden.

[...]

Intervjuer 1: I neste oppgave skulle dere endre koden i 2a til å lage en trekant. Fikk dere til det?

Stine: Nei.

Ragnar: Ja, vi fikk det til. Men vi glemte den *import turtle as tegning* (Figur 4.5.1).

Intervjuer 1: Hva måtte dere endre for å få den til å tegne en trekant?

Ragnar: Jeg tok vel vekk noe, jeg kan ikke huske hvor mye, men vi tok vekk noe. Også i stedet for 90 så tok vi 120.

Intervjuer 1: Var det tilfeldig eller visste dere det?

Ragnar: Ja, eller vi prøvde oss frem i Scratch.

[...]

Transkripsjon 23, oppgave 2b

4.6 Gruppe 6

Gruppe 6 var den siste gruppen som ble intervjuet, og viste mange av de samme utfordringene med begreper og syntaks. Da elevene ble spurt om å hva de trodde de ulike delene av koden gjorde, var både *input* og *int* ukjent for de aller fleste. Begrepet *input* var umiddelbart ukjent for dem, men de skjønte det etter hvert ved å bruke engelsk, og resonnerte seg da frem til at det var noe som skulle puttes inn. Begrepet *int* forble derimot ukjent for dem.

[...]

Intervjuer 1: Var det noen av de delene i koden som var vanskelig å forstå?

Wanda: *Input.*

Ylva: Og *int.*

Vetle: Nei vi skjønte heller ikke *int*, men vi skjønte de to andre.

Wanda: Vi skjønte *input* etter hvert, men hva er *int*?

[...]

Transkripsjon 24, oppgave 1.

Videre i transkripsjon 25 klarte elevene å argumentere på hvorfor «differansen» er en variabel i programmet. Analysen viser at Ylva og Wanda her brukte sin matematiske kunnskap til å forklare at variabler kan forandre seg. Ylva forklarte her at siden a og b kan forandres, så må jo også differansen endres ut fra de andre variablene.

[...]
Intervjuer 1: Er det noen som kan forklare kort hva en variabel er?
Wanda: Det varierer.
Ylva: Det kan byttes liksom på den.
Intervjuer 1: Hvilke variabler fant dere i denne koden her?
Ulrik: a og b.
Ylva: a, b også tenkte vi differansen også *int*. Jeg vet ikke.
Intervjuer 1: Ja, du sa differansen? Hva var det som fikk dere til å tenke det?
Ylva: Hvis a og b kan forandre seg, så må den og forandre seg.
[...]

Transkripsjon 25, oppgave 1.

I oppgave 2 hadde Ylva og Wanda som flere andre også tegnet en trapp. Analysen viser igjen at elevene har utfordringer rundt tolkningen av *left* i koden. Ulrik og Vetle hadde bare tegnet et kvadrat, men sa at de opprinnelig misforstod tallet 90 som en lengde, og ikke som grader. Wanda tolket alle tallene i programmet som lengder, og forkortet 150 til 15. Analysen viser da at hun bruker sin matematiske kunnskap inn i programmet, selv om programmet ikke forteller noe om måleenheten til kvadratet som tegnes.

At Python ikke viser benevninger som steg og grader, eller beskriver at pennen snur seg slik Scratch gjør, skapte her utfordringer for elevene. Ylva sitt bidrag nederst tyder også på at overgangen mellom programmeringsspråkene kan være utfordrende.

[...]

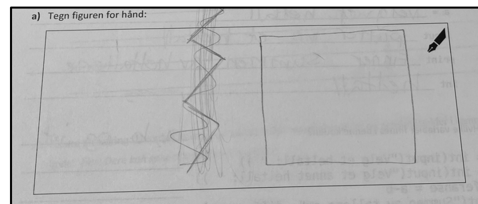
Intervjuer 1: Her i oppgave 2 skulle dere finne ut hvilken geometrisk figur som ble tegnet. Fant dere det ut med en gang eller var dere innom flere figurer?

Jentene ler

Ulrik: Bare en figur. *Elevene viser figurene sine*

Vetle: Den klarte vi.

Wanda: Vi hadde den litt sånn vi *viser figuren av en trapp*



Figur 4.6.1 - Ylva og Wanda sin tegning fra oppgave 2

Wanda: Vi tegnet den litt større enn den egentlig var.

Intervjuer 1: Litt større enn det egentlig var?

Wanda: Ja.

Intervjuer 1: Hva var det som gjorde at du tenkte det?

Wanda: Jeg brukte linjal.

Intervjuer 1: Du brukte linjal. Trodde dere tallene var lengder?

Wanda: Ja.

Intervjuer 1: Ja. Så dere tok først at det skulle gå fram 150 (...)

Ylva: Så har vi forkortet det til 15.

Vetle: Vi misforstod litt i starten fordi vi trodde det var 150 fram, 90 til venstre, men det er at man skal snu seg 90 grader.

Intervjuer 1: Var det noe som var annerledes når dere skrev i Scratch framfor i Trinket?

Ylva: Det stod noe sånn gå også snu, jeg skjønte det ikke helt først men skjønner det nå

[...]

Transkripsjon 26, oppgave 2.

I transkripsjon 27 viser elevene at deres tidligere erfaringer med Scratch, ikke kan overføres direkte til Python. Vetle forklarte at programmet ikke spør om noe i Python, men at den gjør det i Scratch. Analysen viser da at elevene ikke klarer da å se sammenhengen mellom spørreblokken i Scratch og *input* i Python. Elevene hadde i tillegg noen refleksjoner rundt fordelene ved programmeringsspråkene, og sa selv at Scratch er enklere og skaper færre feil. Ulrik sa derimot at tekst også har sine fordeler, ved at koden kan beskrives, og at tekstprogram er enklere å finne frem enn Scratch.

[...]

Intervjuer 1: Var det noe som var veldig annerledes når dere lagde den koden i Scratch framfor i Trinket? Var det noe som skilte seg litt ut når dere skulle programmere i Scratch?

Vetle: Altså i Scratch må du jo.. Jeg vet ikke om det er riktig, men da må du jo spørre først. Det må man vel ikke her, men jeg vet ikke.

Intervjuer 1: Følte dere lengden også var noe annerledes?

Ulrik: Hmm?

Intervjuer 1: Altså så koden lenger ut i Scratch enn i (…)?

Vetle og Ulrik: Ja

Intervjuer: Ser dere noen fordeler å bruke tekst framfor å bruke Scratch?

Ulrik: Du kan jo liksom finne tekst, men ikke alltid Scratch ut av det blå. Du kan liksom skrive hva programmet skal bety.

Intervjuer 1: Hvilke fordeler har Scratch da?

Ulrik: Det er enklere.

Wanda: Ja.

Vetle: Du gjør jo aldri feil.

[...]

Transkripsjon 27, oppgave 3.

5. Diskusjon

I diskusjonen blir det drøftet rundt elevenes utfordringer innen syntaksforståelse, konseptuell og strategisk forståelse, og mulige faktorer som skapte dem drøftet. Målet er å belyse elevenes utfordringer i Python, ved å diskutere disse resultatene opp mot tidligere forskning og teori. Videre diskuteres elevenes forsøk på å bruke tidligere erfaring i Scratch til å forstå oppgavene i Python. Diskusjonen gir et grunnlag for å besvare forskningsspørsmålene.

5.1 Syntaksforståelse

5.1.1 Forståelse av datatyper

Det som gjentar seg mye i resultatene er at elevene ikke satte inn *int* i programmet, og fem av gruppene forstod ikke betydningen av *int*. Dette funnet er ikke overraskende siden de ikke har hatt behov for å definere datatyper før. I Scratch brukes nemlig ikke datatyper, ettersom at *input*-kommandoen står i form av et spørsmål som mottar både tekst og tall. Dersom en ønsker å spesifisere at det kun er tall som ønskes i Scratch, må det legges inn som variabler. For å kunne lage kodene fra Scratch i Python selv, tyder det på at elevene må opparbeide mer instrumentell kunnskap om håndtering av datatyper.

At blokkbaserte programmeringsspråk som Scratch ikke krever definering av datatyper, kan skape utfordringer for elever i Python, siden til og med enkle tekstprogrammer trenger datatyper for å fungere (Kölling et al., 2015). På den andre siden fremstår ikke definering av datatyper som uoverkommelig, og noen av elevene virket til å forstå funksjonen til *int* senere i intervjuene. I praksis er definering av datatyper en syntaksfeil som er relativt lett å oppdage og rette opp når en først er klar over det. Syntaksfeil er generelt lette å rette opp, og er vanlige blant nybegynnere ifølge Qian og Lehman (2017).

Noen av elevene fra gruppe 3 som ble intervjuet hadde spurt en hjelpelærer hva *int* betydde, og fikk i svar at det betydde interact. Dette var nok en gjetning fra hjelpelæreren, men kunne potensielt skapt en misoppfatning av *int* hos elevene. Misoppfatninger fra læreren kan være en faktor i elevers dannelse av misoppfatninger (Qian & Lehman, 2017). Dette viser viktigheten av lærerens kunnskap og erfaring om programmering, før den videreformidles til elevene (Johansen, 2020).

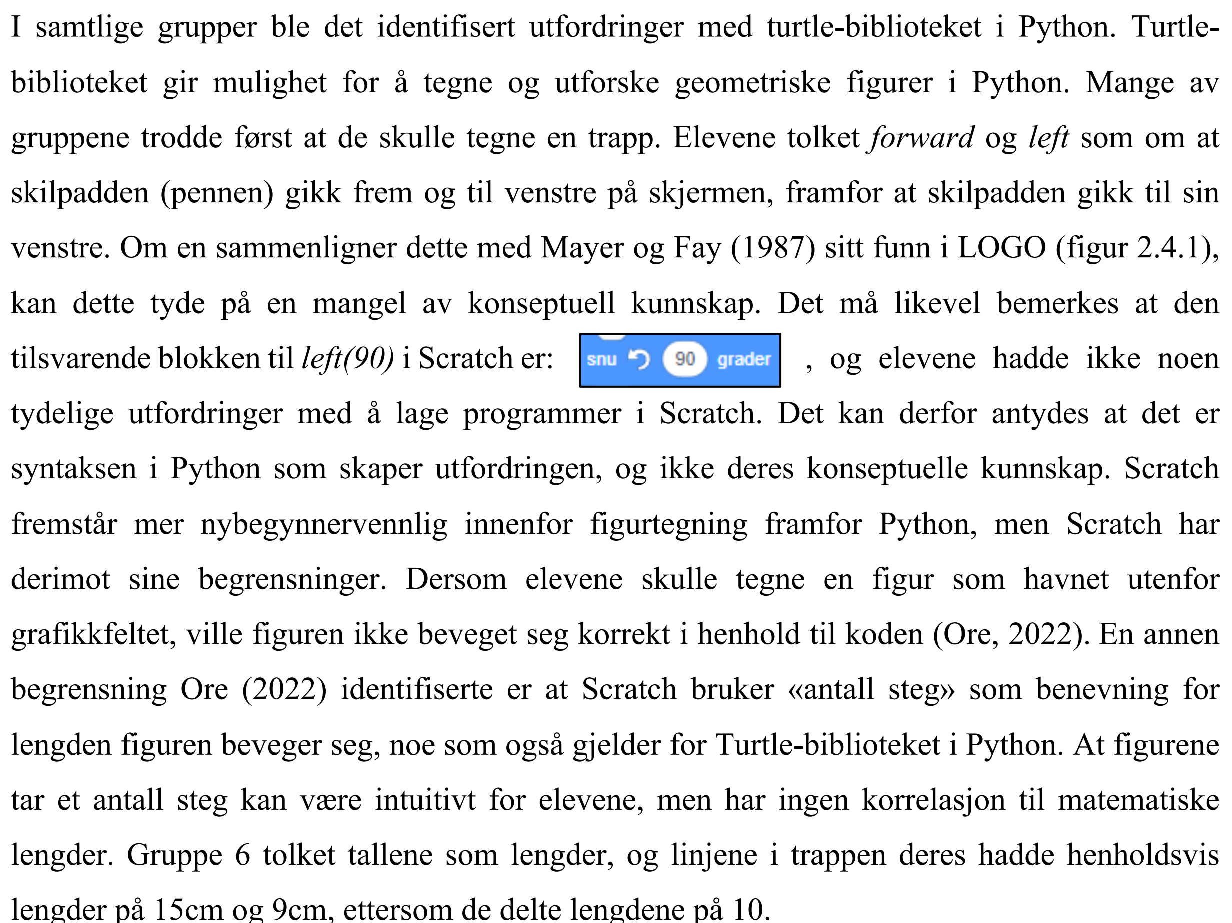
5.1.2 Forståelse av fargeindikatorer og feilmeldinger i Python

Det var særlig en gruppe som hadde problemer med å tolke indikatorene som oppstår i arbeid med Python. I Python blir enkelte kodedeler slik som tekststrenger indikert med farger i programmet. Fargeindikatorene skal gjøre det enklere å lese koden, og viser eksempelvis hvilke kommandoer som blir brukt. Elevene i gruppe 3 forstod at de hadde skrevet noe feil, da de ikke fikk rødfarge i koden deres i oppgave 1. Feilmeldingen hjalp dem likevel ikke i å finne feilen. Dette tyder på at fargeindikatorene og feilmeldingene i Python kan hjelpe elevene å avdekke feil, men ikke nødvendigvis hjelper dem å rette opp feilene, noe også Qian og Lehman (2017, s. 10) forklarer i artikkel.

Disse fargeindikatorene skapte også en annen utfordring i samme gruppe. Elevene i gruppe 3 trodde at alle variabler i Python er markert med farger. Dette gjorde at elevene trodde både de røde tekststrengene, og *int* og *input* som var lilla var variabler. Dette kan ha skapt en konseptuell misforståelse rundt variabler, noe som drøftes videre i neste delkapittel. Det var derimot bare gruppe 3 som tolket indikatorene slik.

5.1.3 Forståelse av Turtle

I samtlige grupper ble det identifisert utfordringer med turtle-biblioteket i Python. Turtle-biblioteket gir mulighet for å tegne og utforske geometriske figurer i Python. Mange av gruppene trodde først at de skulle tegne en trapp. Elevene tolket *forward* og *left* som om at skilpadden (pennen) gikk frem og til venstre på skjermen, framfor at skilpadden gikk til sin venstre. Om en sammenligner dette med Mayer og Fay (1987) sitt funn i LOGO (figur 2.4.1), kan dette tyde på en mangel av konseptuell kunnskap. Det må likevel bemerkes at den tilsvarende blokken til *left(90)* i Scratch er:



, og elevene hadde ikke noen tydelige utfordringer med å lage programmer i Scratch. Det kan derfor antydes at det er syntaksen i Python som skaper utfordringen, og ikke deres konseptuelle kunnskap. Scratch fremstår mer nybegynnervennlig innenfor figurtegning framfor Python, men Scratch har derimot sine begrensninger. Dersom elevene skulle tegne en figur som havnet utenfor grafikkfeltet, ville figuren ikke beveget seg korrekt i henhold til koden (Ore, 2022). En annen begrensning Ore (2022) identifiserte er at Scratch bruker «antall steg» som benevning for lengden figuren beveger seg, noe som også gjelder for Turtle-biblioteket i Python. At figurene tar et antall steg kan være intuitivt for elevene, men har ingen korrelasjon til matematiske lengder. Gruppe 6 tolket tallene som lengder, og linjene i trappen deres hadde henholdsvis lengder på 15cm og 9cm, ettersom de delte lengdene på 10.

Elevene begrunnet at figurer har lengder og bredder, og prøvde da å bruke sin matematiske kompetanse til å forstå tallene i Python, uten at det er en sammenheng mellom disse. Dette skapte muligvis et hinder mellom artefaktet og elevenes matematiske kunnskap om måleenheter. Elevene har opparbeidet seg kunnskap om at lengder har en måleenhet fra konvensjonell matematikk, men når dette plutselig ikke gjelder i programmering (med mindre det legges inn i programmet), kan dette hindre elevenes forståelse (Qian & Lehman, 2017).

5.1.4 Rene syntaks/operator feil

Ved rene syntaks og operatorfeil er det mest interessant å studere elevenes refleksjoner rundt operatoren «**», ettersom denne representasjonen av eksponentiering sannsynligvis var ukjent for elevene. Både gruppe 3 og 5 identifiserte at operatoren «**» i oppgave 1d hadde et multiplikasjonstegn for mye. Elevene klarte dermed å finne en feil i programmets syntaks, uten å ha noe konseptuell kunnskap om syntaksen, noe som tyder på at de klarer å endre og fikse enkle programmer. Selv om de fleste elevene sa de kjente til potenser og eksponenter klarte de ikke å identifisere at operatoren «**» opphøyde a i b i denne oppgaven. Elevene kan tidligere ha møtt andre måter å skrive eksponenter på, noe som igjen skapte et hinder sammenlignet med konvensjonell matematikk (Qian & Lehman, 2017). En annen tolkning er at elevene ble forvirret, ettersom at deres møte med Python og dets syntaks var overveldende (Qian & Lehman, 2017). Dette gjaldt alle gruppene utenom gruppe 2, hvor gruppen klarte å resonnere seg matematisk frem til at operatoren «**» gir en potens i programmet.

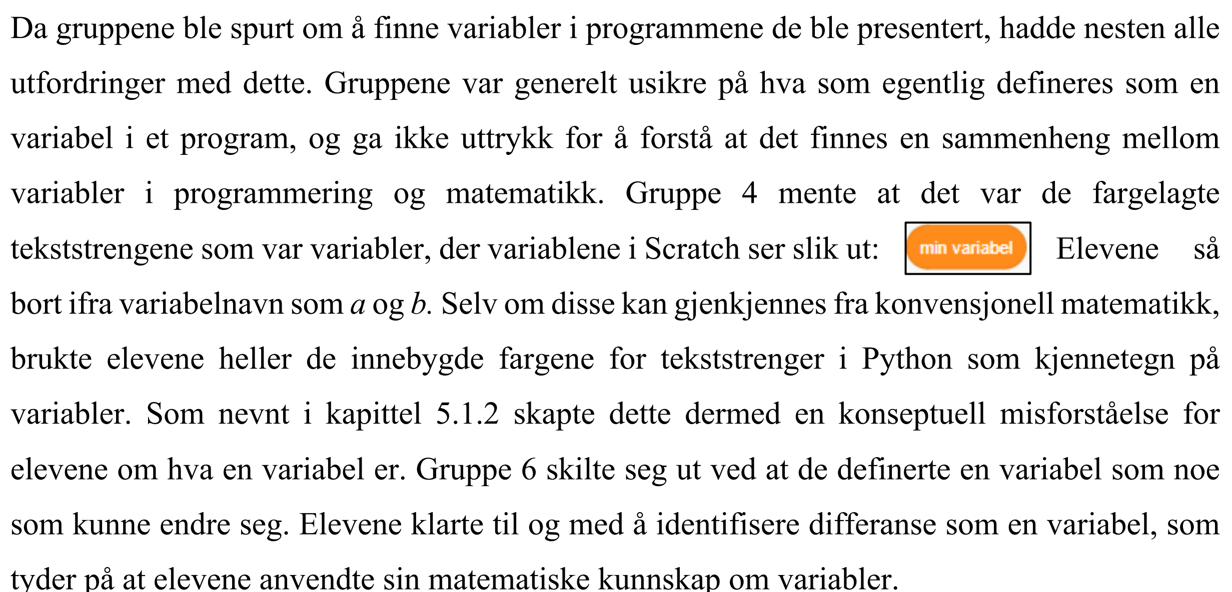
5.2 Konseptuell forståelse

5.2.1 Forståelse av print og programutregninger

Flere av gruppene hadde også problemer med å forstå *print* sin funksjon i programmet. Gruppe 1 og 3 visste at det var noe som skulle skrives ut, men de mente at programmet ikke ville regne ut verdier når *print* ikke var til stede. Elevene viser her en manglende konseptuell forståelse i hvordan programmet kjører, siden de tror at noe må bli skrevet ut for at programmet skal kjøre. Elevene klarte å forstå hva programmet gjorde, men det virker som at elevene brukte den fragmenterte kunnskapen sin innenfor programmering til å forstå programmets funksjon, men algoritmene og helheten bak programmets utførelse var vanskelig å forstå for dem. I følge Qian og Lehman (2017) har elever ofte en fragment kunnskap i programmering som hjelper dem å forstå funksjonen av et program, men som ikke hjelper dem i forstå hvordan programmet fungerer.

5.2.2 Forståelse av input og variabler

Flere av gruppene hadde utfordringer rundt betydningen av *input*, som ber brukeren av programmet til å skrive inn en streng eller en verdi. Et eksempel er gruppe 3 som tror *input* «putter» sammen to tall. Denne misoppfatningen av *input* kan skape videre utfordringer rundt definering av datatyper. Denne tolkningen av *input* kan dermed gjøre at elevene ikke forstår syntaksen i programmet, og deres manglende instrumentelle kunnskap kan dermed gi grunnlag for fremtidige utfordringer med forståelsen av begrepet *int* (Buteau et al., 2020; Qian & Lehman, 2017). I tillegg trekkes det frem at miljøfaktorer i programmeringsspråket kan bidra til utfordringer. Addisjonsoperatoren «+» nevnes spesifikt, og elevene i gruppe 3 var nok ikke klare over at denne operatoren både kan addere sammen tall og strenger. Da de ble fortalt at 1+2 ble 12 uten *int* antok dem at det var *input* som gjorde dette fremfor «+». Denne dobbeltfunksjonen er igjen ulikt fra konvensjonell matematikk, som tyder på en misoppfattelse av *input* hos gruppe 3.



Da gruppene ble spurt om å finne variabler i programmene de ble presentert, hadde nesten alle utfordringer med dette. Gruppene var generelt usikre på hva som egentlig defineres som en variabel i et program, og ga ikke uttrykk for å forstå at det finnes en sammenheng mellom variabler i programmering og matematikk. Gruppe 4 mente at det var de fargelagte tekststrengene som var variabler, der variablene i Scratch ser slik ut: min variabel Elevene så bort ifra variabelnavn som *a* og *b.* Selv om disse kan gjenkjennes fra konvensjonell matematikk, brukte elevene heller de innebygde fargene for tekststrenger i Python som kjennetegn på variabler. Som nevnt i kapittel 5.1.2 skapte dette dermed en konseptuell misforståelse for elevene om hva en variabel er. Gruppe 6 skilte seg ut ved at de definerte en variabel som noe som kunne endre seg. Elevene klarte til og med å identifisere differanse som en variabel, som tyder på at elevene anvendte sin matematiske kunnskap om variabler.

5.2.3 Forståelse av løkker

I gruppe 3 trakk elevene frem at fordelene med Scratch fremfor Python er at Python kan gjenta, eller lage løkker i ulike programmer. Elevene har altså noe forståelse av løkker, og hvilke fordeler de har. Gjentakelse-blokken i Scratch er designet slik at det er tydelig at andre blokker skal plasseres inni den, og blokken forklarer at noe gjentas x antall ganger.

Figur 5.2.1 Gjentakelse-blokk i Scratch

En av fordelene med dette er at utforskning og bruk av løkker blir enklere i Scratch enn i tekstbaserte programmeringsspråk siden syntaksen elimineres, noe Resnick et al. (2009) argumenterer for. Jentene i gruppe 3 forsøkte å lage ei løkke (figur 5.2.2) for å kunne tegne figuren i Python. Elevene fikk ikke hint til å sette opp løkke-strukturen, og mangler blant annet instrumentell kunnskap om at innholdet i løkka må rykkes inn. For at en *while*-løkke skal fungere, må også en egen variabel etableres, og den må øke ved hver gjentakelse slik at betingelsen $n < 4$ gjelder. At elevene ikke klarte dette er ikke overraskende siden de aldri hadde gjort dette før, men etter en innføring i løkker kan det være elevene hadde lært seg dette. At jentene så fordelen med løkker og ønsket å bruke det for å effektivisere koden, kan tolkes som et tegn på strategisk kunnskap hos elevene, selv om deres bruk var ufullstendig.

Skriv koden her:

```
import turtle as tegner
while n<4:
tegner.forward(150)
tegner.left (90)
```

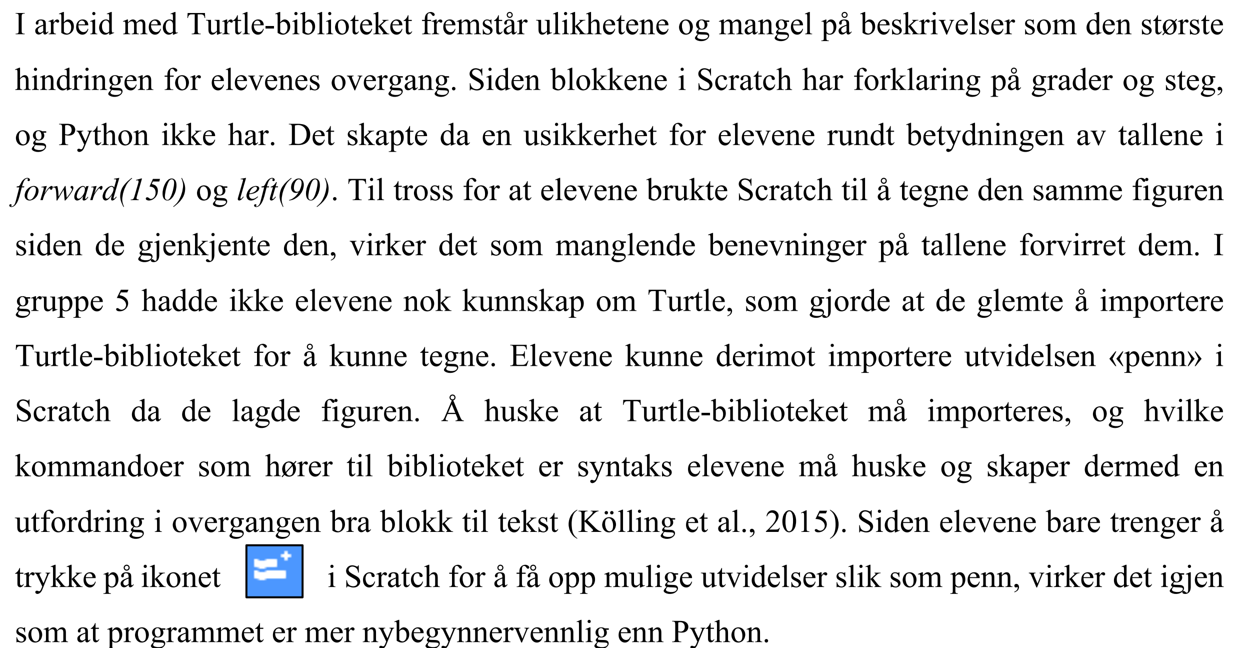
Figur 5.2.2: Katrine og Line sin *while*-løkke. Det er verdt å merke seg at variabelen n ikke er definert og heller ikke øker. Linjene er heller ikke rykket inn i løkken.

5.3 Strategi

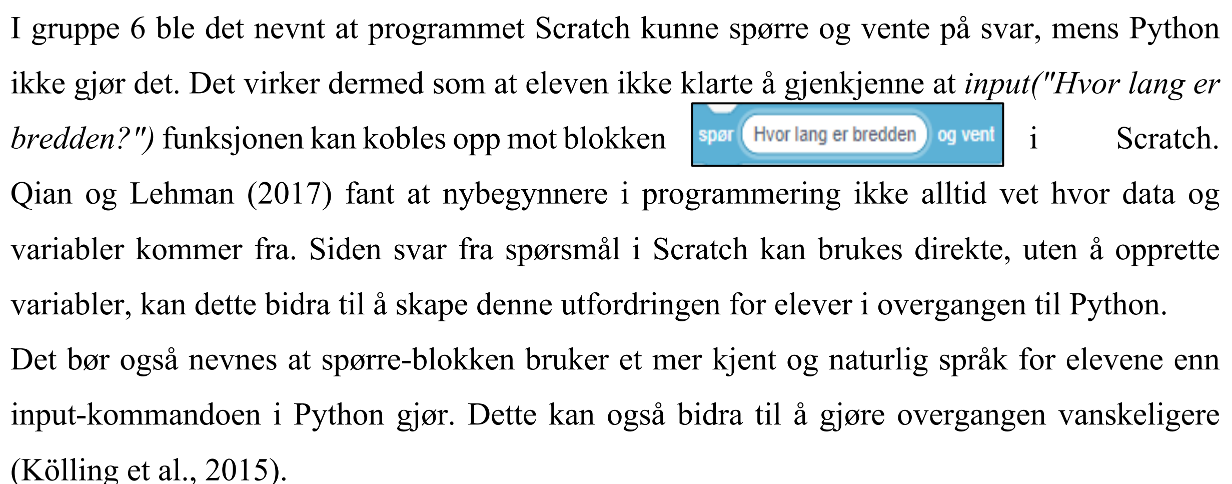
Siden elevene ikke har utviklet instrumentell kunnskap om Python, kan det være vanskelig for dem å løse oppgaver på en strategisk måte. Elevene har derimot kunnskap fra det blokkbaserte programmet Scratch, og de brukte tidligere erfaringer fra Scratch inn i lignende tekstbaserte oppgaver. Basert på den instrumentelle skapelsen til Buteau et al. (2020), kan det antas at elevene er i fasen instrumentell initiering. Dette er tydelig gjennom at elevenes utfordringer var forårsaket av deres manglende instrumentelle kunnskap, og ikke det matematiske. Matematikken virket kjent for alle elevene, og elevenes utfordringer knyttet til artefaktet (Python) ville gjort det vanskelig for elevene å bruke det til å løse matematiske oppgaver. På den andre siden var oppgavedesignet var rettet mot programmering, og de trengte dermed ikke å utvikle sin matematiske kunnskap, men kun sin instrumentelle kunnskap. Elevene utviklet her p-skjemaer som hjalp dem i å forstå artefaktet gjennom instrumentalisering og instrumentering. I initieringsfasen har elevene ikke mulighet til å opparbeide seg noen særlige strategier for å kunne løse matematiske oppgaver. For å skape et instrument som kan hjelpe dem i matematikken, må elevene først komme videre i deres instrumentelle skapelse (Buteau et al., 2020). Dette kommer frem ved operatoren «**», hvor elevenes matematiske kunnskap om eksponenter sannsynligvis var høyere enn deres instrumentelle kunnskap. Jentene fra gruppe 3 som forsøkte å lage en løkke-struktur i Python viste tegn på strategisk tenkning og ønsket å effektivisere koden, men deres manglende instrumentelle kunnskap hindret dem i å tenke strategisk deres.

5.4 Overgangen fra Scratch til Python

Elevenes forsøk på å bruke sine tidligere erfaringer fra Scratch i Python er et annet interessant diskusjonsmoment. Oppgavene i Python var basert på oppgaver elevene hadde gjort i Scratch før, og dette skulle gjøre en overgang mellom programmeringsspråkene mulig.

I arbeid med Turtle-biblioteket fremstår ulikhetene og mangel på beskrivelser som den største hindringen for elevenes overgang. Siden blokkene i Scratch har forklaring på grader og steg, og Python ikke har. Det skapte da en usikkerhet for elevene rundt betydningen av tallene i *forward(150)* og *left(90)*. Til tross for at elevene brukte Scratch til å tegne den samme figuren siden de gjenkjente den, virker det som manglende benevninger på tallene forvirret dem. I gruppe 5 hadde ikke elevene nok kunnskap om Turtle, som gjorde at de glemte å importere Turtle-biblioteket for å kunne tegne. Elevene kunne derimot importere utvidelsen «penn» i Scratch da de lagde figuren. Å huske at Turtle-biblioteket må importeres, og hvilke kommandoer som hører til biblioteket er syntaks elevene må huske og skaper dermed en utfordring i overgangen bra blokk til tekst (Kölling et al., 2015). Siden elevene bare trenger å trykke på ikonet  i Scratch for å få opp mulige utvidelser slik som penn, virker det igjen som at programmet er mer nybegynnervennlig enn Python.

En av fordelene som ofte nevnes om Python, er at det finnes mange biblioteker som kan importeres etter behov, og at det dermed har flere muligheter enn Scratch (Mészárosová, 2015). I motsetning til Scratch må elever enten huske eller finne disse bibliotekene for å importere dem i Python. Dette kan skape utfordringer med syntaksen for elevene.

I gruppe 6 ble det nevnt at programmet Scratch kunne spørre og vente på svar, mens Python ikke gjør det. Det virker dermed som at eleven ikke klarte å gjenkjenne at *input("Hvor lang er bredden?")* funksjonen kan kobles opp mot blokken 

i Scratch. Qian og Lehman (2017) fant at nybegynnere i programmering ikke alltid vet hvor data og variabler kommer fra. Siden svar fra spørsmål i Scratch kan brukes direkte, uten å opprette variabler, kan dette bidra til å skape denne utfordringen for elever i overgangen til Python.

Det bør også nevnes at spørre-blokken bruker et mer kjent og naturlig språk for elevene enn input-kommandoen i Python gjør. Dette kan også bidra til å gjøre overgangen vanskeligere (Kölling et al., 2015).

I gruppe 4 (transkripsjon 19) hadde en elev dannet seg en oppfatning av variabler som en slags spørsmålsboble. Eleven forsøkte da å overføre denne oppfatningen til Python ved å tolke spørsmålet *"Hvor lang er bredden"* som en variabel. Denne misoppfatningen ble kanskje forsterket av at tekststrengen også var indikert med en farge. Dette ble også diskutert i 5.1.2, forståelse av indikatorer, hvor analysen viste at elevene forsøkte å overføre sin kunnskap om farger i Scratch til Python. Dette skapte en misoppfatning hos elevene, om at alle tekststrenger og kommandoer med farge var variabler.

I Scratch er ulike typer av blokker kategorisert i farger for å gjøre det mer oversiktlig. Dette funnet kan derimot tyde på at en som lærer burde være oppmerksom på at elever kan anvende disse fargekategoriene inn i Python, og dermed misoppfatte variabler. Indikatorer og feilmeldinger kan også skape utfordringer for elever i overgangen til Python, ifølge Kölling et al. (2015). Flere av elevene møtte disse feilmeldingene og indikatorene, og hadde utfordringer med å tolke og bruke dem. Siden feilmeldinger og indikatorer ikke fremtrer i Scratch, er det ikke overraskende at elever ikke forstår disse. Videre kan språket i disse feilmeldingene og Python skape utfordringer for elevene. I motsetning til Scratch, må det i Python leses og skrives på engelsk. Ved å la elever bruke Scratch på engelsk fremfor norsk kan denne ulikheten mellom programmeringsspråkene reduseres litt, men elevene må likevel lære seg grammatikken og kodeordene som er unike i Python.
Elevenes forsøk på å skrive løkker i kapittel 5.2.3 gir antydninger om at løkker og betingelser fra Scratch ikke er særlig overførbart til Python. Siden elevene ikke behøver å sette opp egne strukturer for løkker i Scratch, er dette nytt for dem i Python. Flere av elevene valgte derfor heller å jobbe i Scratch under oppgavene, sannsynligvis fordi de kan prøve seg frem, og trenger kun å fylle inn de angitte rutene i blokkene. Dette stemmer godt overens med funnene (Grover et al., 2015, s. 222) hvor elevene klarte å tolke Scratch-programmer i en tekstbasert kontekst, men hadde utfordringer med strukturer som løkker. I Python må hele strukturen settes opp manuelt med riktig syntaks og innrykk, noe som er vanskelig for nybegynnere (Kölling et al., 2015). Siden blokkene i Scratch allerede er ferdiglagde er løkke-strukturer unikt for Python og vanskelig å gjenskape. Elevers arbeid med løkker i Scratch er derfor forbeholdt konseptuell forståelse om løkker, og ikke kunnskap om syntaks.

6. Avslutning

6.1 Konklusjon

Gjennom studien har det blitt utført observasjoner og intervjuer for å undersøke elevers første møte med tekstprogrammering. Studien har lagt vekt på teori og tidligere forskning knyttet opp mot programmering, elevenes instrumentelle skapelse og hvilke utfordringer de kan møte på. Resultatene som er presentert og diskutert er blitt gjort i lys av dette og skal nå konkluderes med utgangspunkt i forskningsspørsmålene:

1. **Hvilke utfordringer har elever i sitt første møte med Python?**
2. **Hvordan bruker 8.trinn elever sine erfaringer i Scratch til å løse oppgaver i Python?**

6.1.1 Konklusjon forskningsspørsmål 1

Studien viser at 8.trinn elever møter flere av de samme utfordringene som Qian og Lehman (2017) også har funnet i sin gjennomgang av tidligere forskning. Resultatene peker på at elever har visse utfordringer rundt syntaksen og begreper i Python.
Definering av datatyper og feilmeldinger er særlig utfordrende for elevene siden de ikke har møtt dem før. Miljøfaktorer i språket slik som engelske kommandoer og manglende benevninger på tall skapte også utfordringer for elevene. Elevene visste også noen konseptuelle utfordringer rundt variabler og hvordan programmer og løkker fungerer. Sannsynligvis stammer disse utfordringene fra elevenes ufullstendige forståelse, og er derfor ikke overraskende. Elevenes forståelse for variabler var til tider bekymringsverdig, og elevene slet med å identifisere dem i oppgavene. Det ble stort sett ikke observert noe bemerkelsesverdig innenfor strategi. Dette er nok fordi elevene ikke har nok instrumentell kunnskap om programmeringsspråket til å kunne utvikle strategier for å løse oppgavene. På den andre siden antydes det at elevene stort sett forstod helheten av de ulike tekstprogrammene de fikk presentert i oppgavesettet, selv om de møtte flere utfordringer rundt den ukjente syntaksen og deres ufullstendige forståelse. Siden algoritmisk tenkning er tydeligst når elevene utvikler sin strategiske kunnskap, virker det som Python ikke er godt egnet for disse elevene, sammenlignet med Scratch. Likevel er mulighetene innenfor matematikk større og flere i Python, og det er ønsket at elevene gradvis introduseres for det. Dersom elevenes instrumentelle kunnskap i Python styrkes kan det muligvis bli et nyttig instrument for elevene, både i grunnskolen og videre i livet.

6.1.2 Konklusjon forskningsspørsmål 2

Funnene i studien tyder på at elevenes tidligere instrumentelle kunnskap innen det blokkbaserte programmet Scratch, delvis hjalp elevene å forstå oppgavene i Python. Elevene nevnte flere ganger selv at de forsøkte å bruke tidligere erfaring i Scratch til å forstå variabler, og antok en sammenheng mellom farger og spørsmål i Python og Scratch. Det er derimot flere egenskaper i Python som ikke finnes eller brukes i Scratch, og som ikke kan overføres direkte mellom programmeringsspråkene. Dette gjelder særlig oppbygningen av løkke-strukturer, definering av datatyper og feilmeldinger. Scratch har i tillegg flere benevninger og forklaringer som hjelper elevene, men som skaper usikkerhet i overgangen siden disse ikke gis i Python. Det ser ut til at elevene kan bruke sine erfaringer med Scratch til å løse tekstbaserte program, så lenge det er en tydelig sammenheng mellom oppgavene fra Scratch til Python. Likevel er mye av deres erfaring ikke direkte overførbar, og overgangen fra Scratch til Python virker til tider utfordrende. Tidligere studier av Armoni et al. (2015) og Kölling et al. (2015) har lignende funn, men understreker likevel at erfaring i Scratch er verdifull, og at overgangen kan gjøres på en god måte. Selv om forskningsspørsmålet i denne studien ikke tar hensyn til om erfaringene hjelper elevene eller ikke, fremstår de som nyttige, spesielt med tanke på elevenes motivasjon og konseptuelle kunnskap om løkker, da Python kan være utilgivelig for nybegynnere.

6.2 Studiens styrker og begrensninger

Denne studien har sine begrensninger, først og fremst når det kommer til tidsrammen som er gitt til prosjektet. Ettersom dette prosjektet har strukket seg over et vårsemester, er tidsrommet for datainnsamlingen for studien avgrenset. Det har likevel vært nok tid til å kunne besvare forskningsspørsmålene, og mye data ble samlet inn på kort tid. Selv om studien har sine begrensninger, gir variasjonen av ulike klasser og grupper et større grunnlag for studiens validitet. Sidene flere av funnene i de ulike gruppene både lignet på hverandre, og på tidligere forskning, kan flere av funnene antas å gjelde generelt. En annen begrensning ved denne studien er studentenes subjektive påvirkning under utførelse av prosjektet. Oppgavene som ble gitt, og intervjuspørsmålene som ble stilt, kunne muligvis vært designet bedre, slik at elevenes utfordringer og tanker hadde kommet tydeligere frem. Det er i tillegg mulig å analysere og tolke transkripsjonene annerledes, og interessante funn kan ha blitt oversett i prosessen. Analysen av funnene er derimot gjort av begge studentene i fellesskap, slik at en forsker-triangulering er dannet. Dette gir da en større sikkerhet for funnene og konklusjonene, ettersom subjektive tolkninger er blitt nøye drøftet og gjennomgått, og godkjent av begge forskerne.

Studiens styrke kommer også frem i dens unike perspektiv i forhold til at elevenes første møte med tekstprogrammering er studert. Siden alle elevene i gruppene unntatt Benjamin aldri hadde møtt Python før gir disse funnene et innblikk i hvordan tekstprogrammering fremstår for nybegynnere, og gir også verdifull innsikt i elevenes utfordringer i programmet og deres overgang fra blokkprogrammering til tekstprogrammering. Det kan nevnes at i den nye læreplanen skal elever fra 5.trinn lære programmering (Utdanningsdirektoratet, 2020b), og fremtidige 8.trinn elever har sannsynligvis mer kunnskap innenfor programmering enn elevene i denne studien.

6.3 Videre forskning og implikasjoner

Denne studien avdekker hvilke utfordringer elever har i deres første møte med tekstprogrammering, og det vil da være en naturlig fortsettelse å studere hvordan overgangen kan forbedres og tilpasses for elever og lærere. Det tyder på at Scratch er nyttig for å lære elever grunnleggende prinsipper i programmering, men denne studien impliserer at Scratch har noen egenskaper som ikke lar seg overføre til Python. Dette gjelder blant annet datatyper, variabler og feilmeldinger. Forskning på hvordan disse konseptene fra Scratch kan overføres inn i Python, kan forhåpentligvis gjøre overgangen mellom blokk og tekst mindre utfordrende. En tidligere studie av Kölling et al. (2015) presenterer «frame-based editing» som en mulighet for å gjøre overgangen enklere. En replikasjon av denne studien i norsk skole kunne vært interessant, og det er viktig at lærere er åpne for nye muligheter, og ikke låser seg til bare Scratch og Python.

Funnene fra denne studien underbygges av tidligere forskning og elevenes utfordringer innen det konseptuelle, syntaksen og deres strategi. Videre viser denne studien nytten av å lære seg programmering før det integreres i matematikkundervisningen. Selv om oppgavene kan baseres på matematikk, må elevene lære seg programmeringsspråket før de kan anvende det i matematikken. Studien viser at elevene kan tolke programmer i Python, men sliter med å skrive og endre koder selv. Siden elever hovedsakelig utvikler sin algoritmiske tenkning når de arbeider strategis, er det derfor nyttig å anvende begge programmeringsspråkene i undervisningen. Elevene kan med fordel utforske programmering på egenhånd i Scratch, og deretter få hjelp av lærer til å gjøre overgangen til Python. Slik kan elevene potensielt utvikle sin algoritmiske tenkning, samtidig som de gradvis gjør overgangen til Python som har mer muligheter for deres matematiske læring.

Elevene i denne studien viste flere utfordringer knyttet til variabelbegrepet. Som lærer kan en være mer oppmerksom på forklaringen av variabler slik at elevene ikke danner seg misoppfatninger. For å forberede elevene på variabelbruk i Python kan de allerede i Scratch tilvennes å alltid opprette variabler. Konseptet løkker er derimot vanskeligere å forberede elever på i Scratch. Elevene kan tilegne seg konseptuell kunnskap rundt løkker, men syntaksen rundt strukturene av for- og while-løkker i Python må læres utelukkende i Python, siden blokkene i Scratch ikke krever kunnskap om syntaks. Siden løkker er essensielt i programmering, viser dette verdien av å jobbe med både Scratch og Python.

6.4 Utbytte av studien

Ettersom studien er gjennomført, står vi igjen med mye relevant kunnskap som kan tas med inn i den framtidige karrieren som lærer. Motivasjonen vår for denne studien var at programmering fortsatt er relativt nytt i læreplanen og matematikken, og for å undervise elever i programmering, er det viktig å studere hvilke utfordringer de har. Gjennom denne studien har vi avdekket flere utfordringer blant elevene som vi selv ikke hadde tenkt på. Nå som vi har kunnskap om disse utfordringene kan vi tilrettelegge fremtidig undervisning rundt disse, og hjelpe elever på en bedre måte. Kunnskapen gir også et godt utgangspunkt vi som lærere kan forholde seg til elevenes frustrasjoner og mestringsfølelse i arbeid med programmering. I tillegg har vi lest mye interessant litteratur, som vi ønsker å ta med oss videre som lærere, og kanskje som fremtidige forskere.

Som lærere ønsker vi å gjøre overgangen mellom blokkprogrammering til tekstprogrammering så enkelt og raskt som mulig for elevene, ettersom vi anser de matematiske mulighetene som større i Python enn i Scratch. Dersom elevene skal forbedres til eksamensoppgaver skrevet i Python, burde de så tidlig som mulig på ungdomstrinnet starte overgangen. Gjennom arbeidet med dette studiet har vi derimot avdekket noen utfordringer som kan oppstå i overgangen til Python, spesielt rundt syntaksen. Det virker som at Scratch har noen fordeler, og vi ønsker derfor å bruke begge programmeringsspråkene i vår fremtidige undervisning, slik at de kan utfylle hverandre.

7. Litteraturliste

Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M. (2015). From Scratch to “Real” Programming. *ACM Trans. Comput. Educ.*, *14*(4), Article 25. https://doi.org/10.1145/2677087

Assude, T. (2007). Teachers’ practices and degree of ICT integration. Proceedings of the fifth congress of the European Society for Research in Mathematics Education,

Bayman, P. & Mayer, R. E. (1988). Using conceptual models to teach BASIC computer programming. *Journal of Educational Psychology*, *80*(3), 291–298.

Bell, J. & Waters, S. (2018). *Doing your research project* (7. utg.). Open University Press.

Bryman, A. (2016). *Sosial Research Methods* (5. utg.). Oxford University Press.

Buteau, C., Muller, E., Mgombelo, J., Sacristán, A. I. & Dreise, K. (2020). Instrumental Genesis Stages of Programming for Mathematical Work. *Digital Experiences in Mathematics Education*, *6*(3), 367-390. https://doi.org/10.1007/s40751-020-00060-w

Crabtree, B. F. & Miller, W. L. (1999). *Doing Qualitative Research* (2. utg.). SAGE Publications, Inc.

Dolonen, J. A., Kluge, A., Litherland, K. & Mørch, A. (2019). Litteraturgjennomgang av programmering i skolen. *Universitetet i Oslo*.

Felzmann, H. (2009, 19 March 2009). *Ethical issues in school-based research*. Vulnerable groups – ethical dimensions and dilemmas, Birmingham.

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

Gjørvik, Ø. & Torkildsen, H. A. (2019). Algoritmisk tekning. *Tangenten: Tidsskrift for matematikkundervisning*, *30*, 31 - 37.

Grover, S., Pea, R. & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*, 199–237.

Johansen, A.-K. (2020). *Programmering vil bli en utfordring for lærere*. https://forskning.no/barn-og-ungdom-hogskolen-i-ostfold-matematikk/programmering-vil-bli-en-utfordring-for-laerere/1711838

Kölling, M., Brown, N. C. C. & Altadmri, A. (2015). *Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming*. The 10th Workshop in Primary and Secondary Computing Education, London, UK.

Lyle, J. (2003). Stimulated Recall: A Report on Its Use in Naturalistic Research. *British Educational Research Journal*, *29*(6), 861-878. https://doi.org/10.1080/0141192032000137349

Lær Kidsa Koding. (2023). *Koding i skolen*. Hentet 20.04 fra https://www.kidsakoder.no/skole/

Mannila, L. (2017). *Att undervisa i programmering i skolan-Varför, vad och hur?* Studentlitteratur AB.

Mayer, R. E., Dyck, J. L. & Vilberg, W. (1986). Learning to program and learning to think: what's the connection? *Commun. ACM*, *29*(7), 605–610. https://doi.org/10.1145/6138.6142

Mayer, R. E. & Fay, A. L. (1987). A chain of cognitive changes with learning to program in Logo. *Journal of Educational Psychology*, *79*, 269-279. https://doi.org/10.1037/0022-0663.79.3.269

McGill, T. J. & Volet, S. E. (1997). A conceptual framework for analyzing students’ knowledge of programming. *Journal of Research on Computing in Education*, *29*(3), 276-297.

Mészárosová, E. (2015). Is Python an appropiate programming language for teaching programming in secondary schools? *International Journal of Information and Communication Technologies in Education*, *Volume 4,*(2015), 5-14.
Mladenović, M., Krpan, D. & Mladenović, S. (2016). Introducing programming to elementary students novices by using game devolopment in Python and Scratch. 8th International Conference on Education and New Learning Technologies, Barcelona, Spain.
NOU. (2013). *Hindre for digital verdiskaping*. a.-o. k. Fornyings-. Departementenes servicesenter. https://www.regjeringen.no/contentassets/e2f0d5676e144305967f21011b715c16/no/pdfs/nou201320130002000dddpdfs.pdf
Ore, C. (2022). *Programmering i matematikkfaget* [Master, Universitetet i Agder].
Papert, S. (1993). Mindstorms: Children, Computers and Powerful Ideas. I *Harvester Studies in Cognitive Science* (2. utg.). Basic Books.
Pea, R. D. & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*, 137-168. https://doi.org/10.1016/0732-118X(84)90018-7
Powers, K., Ecott, S. & Hirshfield, L. M. (2007). Through the looking glass: teaching CS0 with Alice. Proceedings of the 38th SIGCSE technical symposium on Computer science education, Covington, Kentucky, USA.
Qian, Y. & Lehman, J. (2017). Students Misconceptions and Other Difficulties in Introductory Programming A literature review.
Resnick, M., Maloney, J., Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. (2009). Scratch: Programming for Everyone. *Communications of the ACM*, (52(11)), 60-67.
Sevik, K. (2016). Programmering i skolen. *Notat fra Senter for IKT i utdanningen*.
Store norske leksikon. (2022, 5. september 2022). *Programmering*. Store Norske Leksikon.
Trouche, L. (2004). Managing the Complexity of Human/Machine Interactions in Computerized Learning Environments: Guiding Students' Command Process through Instrumental Orchestrations. *International Journal of Computers for Mathematical Learning*, *9*(3), 281-307. https://doi.org/10.1007/s10758-004-3468-5
Utdanningsdirektoratet. (2019a). *Algoritmisk tenkning*. Utdanningsdirektoratet. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/
Utdanningsdirektoratet. (2019b). *Kjerneelement*. Utdanningsdirektoratet. https://www.udir.no/lk20/mat01-05/om-faget/kjerneelementer
Utdanningsdirektoratet. (2020a, 03.09.2020). *Hva er nytt i matematikk?* Utdanningsdirektoratet. https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-matematikk/
Utdanningsdirektoratet. (2020b). *Læreplan i matematikk 1-10 (MAT01-05)*. Utdanningsdirektoratet.
Utdanningsdirektoratet. (2022). Eksamen MAT1019 Matematikk 1P. I(s. 1-24). Utdanningsdirektoratet.
Vail, K. (2003). School Technology Grows Up. *The American school board journal*, *190*(9), 34-37.
Vergnaud, G. (1998). Towards a cognitive theory of practice. *Mathematics Education as a Research Domain: A Search for Identity: An ICMI Study Book 1*, 227-240.
Vergnaud, G. (2009). The Theory of Conceptual Fields. *Human Development*, *52*(2), 83-94. https://doi.org/10.1159/000202727
Vygotsky, L. S. (1978). *Mind in society: Development of higher psychological processes*. Harvard university press.

Wing, J., M. (2006). Computational Thinking. *Communications of the ACM*, *49(3)*, 33-35.
Wong, A. W., Salimi, A., Chowdhury, S. & Hindle, A. (2019). Syntax and Stack Overflow: A methodology for extracting a corpus of syntax errors and fixes.

8. Vedlegg

8.1 Samtykkeskjema

Vil du delta i et forskningsprosjekt

Programmering i matematikk med fokus på elevers tankemåter

Dette er et spørsmål til dere som foresatte om at ditt/deres barn kan delta i et forskningsprosjekt hvor formålet er å se på hvilke undervisningsmetoder som kan fungere best for elever når de arbeider med programmering. Ved å la elevene jobbe med opplegg som er ment til å trigge og skape ulike løsninger, arbeidsmetoder og argumentasjoner rundt programmering ønsker vi å studere hvordan elever håndterer disse oppleggene sammen som en gruppe og dermed få innsyn i deres tankemåter. Forskningsprosjektet inngår som en del av vår mastergradsoppgave som trer i kraft fra januar 2023.

Hvem er ansvarlig for forskningsprosjektet?

Universitetet i Agder er ansvarlig for prosjektet. Forskere involvert i prosjektet er masterstudenter Chris Owe Guttormsen (chrisg18@student.uia.no) og Magnus Repstad (magnr18@student.uia.no) ved universitetet i Agder. Veilederen vår er Anders Skarpeteig Fidje, universitetslektor ved Universitetet i Agder (anders.s.fidje@uia.no)

Hvorfor får du spørsmål om å delta?

Etter ny læreplan i 2020 har vi fått inntrykk at både lærere og elever synes det er krevende å arbeide med programmering, både i ungdomsskole og videregående. På trinnet barnet ditt/deres går på, er et av de nye kompetansemålene i fra den nye læreplanen i matematikk *«å utforske matematiske egenskaper og sammenhenger ved å bruke programmering»* som er et mål vi ønsker å undersøke ut ifra undervisningsopplegget vi har utviklet. Universitetet i Agder ved Anders S. Fidje har et samarbeid med skolen ditt/deres barn går på, og vi ønsker å utføre forskningsoppgaven i klassen som ditt/deres barn er elev i.

Hva innebærer det for ditt/deres barn som deltar?
Selve datainnsamlingen vi foregå i elevenes klasserom hvor vi med videoopptak, skjermopptak og intervju forsøker å få innsikt i elevenes tankemåter og løsninger. Videoopptak vil være nyttig for å dokumentere det elevene sier til hverandre, mens skjermopptak er nyttig for å følge programmeringen til elevene. Intervju er nyttig for å høre elevenes refleksjoner og tankemåter. Opplegget vil gå over en vanlig matematikktime i uke 5/6, og noen vil også bli tilbudt et reflekterende gruppeintervju etterpå. Om dere foresatte ønsker å se på undervisningsoppleggene, og eventuelt om prosjektbeskrivelsen av masteroppgaven er det bare å ta kontakt på forhånd. Elevene vil jobbe sammen i grupper, og matematikklæreren deres vil være til stede.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du lar ditt/deres barn delta, kan dere når som helst trekke samtykket uten å oppgi noen grunn. Det vil ikke ha noen negative konsekvenser for ditt/deres barn hvis samtykke trekkes. Disse undervisningsoppleggene vil ikke ha noe form for påvirkning av elevenes vurdering eller karakter. Dersom du ikke ønsker at barnet ditt/deres skal delta i undersøkelsen, skal det legges til rette for alternativt opplegg (i samråd med lærer).

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om elevene til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket. Det er kun forskerne som vil ha tilgang til informasjonen som samles inn. Video og lydopptakene vil bli lagret og beskyttet i Universitetet i Agders sine låste servere. Når vi skal transkribere videoene og lydopptakene, vil vi bruke pseudonymer framfor ekte navn for å skjule eleven sin identitet.

Hva skjer med personopplysningene?
Alt personidentifiserende som samles inn (lyd, video og eventuelle skriftlige arbeider til elevene) vil bli oppbevart trygt og kryptert på en lukket og låst server hos Universitetet i Agder. Innen 28. februar

slettes opptak fra lydopptaker/videoopptaker. Video- og lydopptak vil bli transkribert, og de trygt lagrede opptakene vil innen 31. desember 2023 bli permanent slettet. I den skriftlige masteroppgaven vil alt som var personidentifiserende være anonymisert.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke. På oppdrag fra Universitetet i Agder, institutt for matematiske fag, har Personverntjenester vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Deres rettigheter
Så lenge barnet ditt/deres kan identifiseres i datamaterialet, har du rett til:

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

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

- Veileder: Anders Skarpeteig Fidje, anders.s.fidje@uia.no
- Forskere: Chris Owe Guttormsen, (chrisg18@uia.no) og Magnus Repstad (magnr18@student.uia.no)
- Universitetet i Agders personvernombud: Trond Hauso, trond.hauso@uia.no
- Personverntjenester på epost (personverntjenester@sikt.no) eller på telefon: 53 21 15 00.

Med vennlig hilsen

Anders Skarpteig Fidje , *Chris Owe Guttormsen & Magnus Repstad*

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet «Programmering i matematikk med fokus på elevers tankemåter» og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ Å delta i forskningsprosjektet (som omfatter video og lydopptak av undervisningsoppleggene)
- ☐ At den informasjonen som blir samlet inn fra undervisningsoppleggene vil bli bearbeidet og lagret under forskningsarbeidet og slettet permanent den 31.Desember 2023
- ☐ Å delta på reflekterende gruppe-intervju i etterkant av opplegget

----
(Signert av foresatt/foresatte, dato)

(Signert av prosjektdeltaker (elev), dato)

8.2 Søknad til NSD

Referansenummer	**Vurderingstype**	**Dato**
401062	Standard	08.12.2022

Prosjekttittel
Undervisningsopplegg rundt programmering i matematikk

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

Prosjektansvarlig
Anders Skarpeteig Fidje

Student
Chris Owe Guttormsen

Prosjektperiode
01.01.2023 - 31.12.2023

Kategorier personopplysninger
Alminnelige

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

Behandlingen av personopplysningene er lovlig så fremt den gjennomføres som oppgitt i meldeskjemaet. Det lovlige grunnlaget gjelder til 31.12.2023.

Meldeskjema

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

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

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

TYPE OPPLYSNINGER OG VARIGHET
Prosjektet vil behandle alminnelige kategorier av personopplysninger om ungdomsskoleelever i alderen 13-14 år (åttende trinn) frem til 31.12.2023.

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

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

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

lovlighet, rettferdighet og åpenhet (art. 5.1 a), ved at foresatte får tilfredsstillende informasjon om og samtykker til behandlingen

formålsbegrensning (art. 5.1 b), ved at personopplysninger samles inn for spesifikke, uttrykkelig angitte og berettigede formål, og ikke viderebehandles til nye uforenlige formål

dataminimering (art. 5.1 c), ved at det kun behandles opplysninger som er adekvate, relevante og nødvendige for formålet med prosjektet

lagringsbegrensning (art. 5.1 e), ved at personopplysningene ikke lagres lengre enn nødvendig for å oppfylle formålet

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

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

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

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

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

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

MELD VESENTLIGE ENDRINGER
Dersom det skjer vesentlige endringer i behandlingen av personopplysninger, kan det være nødvendig å melde dette til oss ved å oppdatere meldeskjemaet. Før du melder inn en endring, oppfordrer vi deg til å lese om hvilke type endringer det er nødvendig å melde:

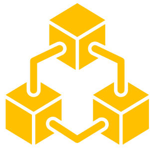
https://www.nsd.no/personverntjenester/fylle-ut-meldeskjema-for-personopplysninger/melde-endringer-i-meldeskjema. Du må vente på svar fra oss før endringen gjennomføres.

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

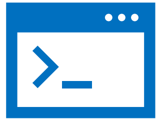
Kontaktperson hos oss: Anne Marie Try Laundal

Lykke til med prosjektet!

8.3 Oppgavesett ungdomsskole



Programmeringsoppgaver til ungdomsskole



Oppgavene er merket med Scratch (Blokkprogrammering) eller Trinket (Python) etter hvor dere skal gjøre dem. Oppgaver med tekstlinjer kan gjøres rett på arket. Bruk maksimalt 15 min på hver oppgave.

Gruppenavn: ______________________________ **Klasse:** ___________________

Oppgave 1

a) Hva gjør denne koden? ____

```
a = int(input("Velg et heltall: "))
b = int(input("Velg et annet heltall: "))
summen = a+b
print("Summen av tallene er", summen)
```

b) Hva tror dere disse kodedelene gjør?

a = ____

input ____

print ____

int ____

c) Hvilke variabler finnes i den følgende koden?

```
a = int(input("Velg et heltall: "))
b = int(input("Velg et annet heltall: "))
differanse = a-b
print("Summen av tallene er", differanse)
```

____

____

____

____

d) Denne koden skal egentlig multiplisere sammen to tall, men om vi setter inn tallene 2 og 3 får vi 8, og ikke 6 som det burde være. Hva er feil, og hvordan bør koden endres for å fikse den? Hva tror du feilen gjorde?

```
a = int(input("Velg et heltall: "))
b = int(input("Velg et annet heltall: "))
produkt = a**b
print("Summen av tallene er", produkt)
```

____

____

____

____

e) Hvordan tror dere koden i **d)** kan endres til å finne produktet av tre eller flere faktorer?

____

____

____

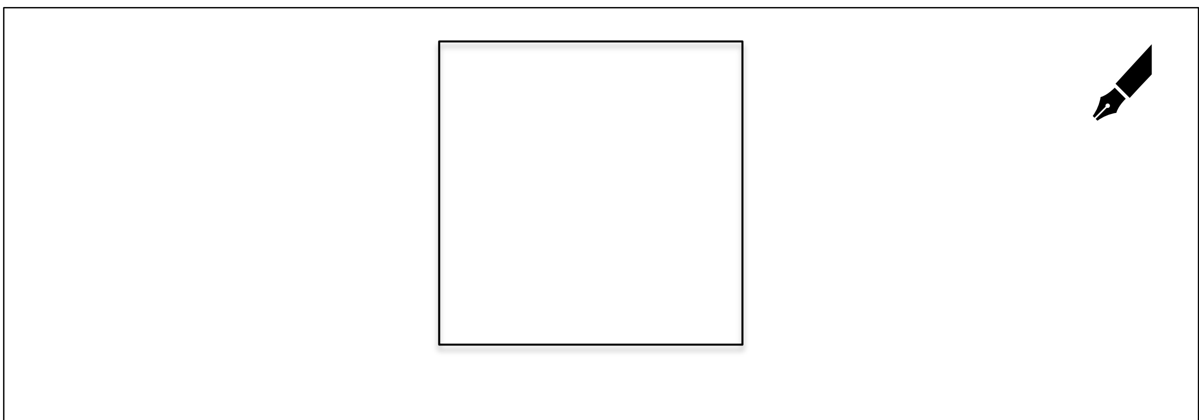
____

Oppgave 2

Denne koden tegner en geometrisk figur i **Trinket**.

```
import turtle as tegner #Importer turtle som fungerer som en penn i python
tegner.forward(150)
tegner.left(90)
tegner.forward(150)
tegner.left(90)
tegner.forward(150)
tegner.left(90)
tegner.forward(150)
```

a) Tegn figuren for hånd:



b) Lag et program i **Scratch** som tegner den samme geometriske figuren

c) Hvordan kan dere endre koden i **Trinket** a) til å tegne en trekant? Skriv den også ned her:

____

____

____

____

____

d) Dere har tidligere brukt løkker (gjentakelser) for å gjøre koder i **Scratch** kortere. Hvordan kan dere gjøre **Trinket**-koden i a) kortere ved bruk av en løkke?

Tips: Dere kan søke opp dette på internett

Skriv koden her:

____

____

____

____

____

Oppgave 3

Denne koden finner omkretsen av et rektangel med en gitt lengde og bredde

```
lengden = int(input("Hvor lang er bredden? " ))
bredden = int(input("Hvor lang er bredden? " ))
omkrets = lengden + bredden + lengden + bredden
print("Omkretsen er", omkrets)
```

a) Hva vil dere utvide koden slik at den også regner ut areal?

____

____

b) Dersom målene er i centimeter, hvordan tror dere dette kan legges til i koden?

____

____

____

c) Lag den samme koden i **Scratch**

Oppgave 4

Denne koden sjekker om en person er myndig

```
alder = int(input("Hvor gammel er du? "))
if alder < 18:
  print("Du er ikke myndig")
else:
  print("Du er myndig")
```

a) Lag den samme koden i **Scratch**

b) Lag en egen kode i **Trinket** basert på den over for å sjekke om en person er i pensjonistalder (67år eller eldre). Skriv også koden ned her:

c) Ekstra utfordring: Hvordan kunne dere sjekket både myndighetsalder og pensjonistalder i samme kode? *Tips: Dere kan søke opp dette på internett*