Høgskolen
i Innlandet

Fakultet for lærerutdanning og pedagogikk

Edvart Granseth

Masteroppgave

Programmeringsoppgaver i Kikora:

En studie av Kognitive Krav, Oppgavetype, Affordances & Constraints

Programming Tasks in Kikora:

A Study of Cognitive Demands, Task Type, Affordances & Constraints

Grunnskolelærerutdanning for 5-10.trinn

2023

Samtykker til publisering i digitalt arkiv Brage JA ☒ NEI ☐

Forord

Jeg ønsker å takke Kikora for tillatelsen til å bruke bilder fra deres sider. Dette har vært avgjørende for å kunne illustrere og berike oppgaven med visuelt innhold.

Jeg ønsker også å uttrykke min takknemlighet til min veileder for hjelp gjennom hele skriveprosessen.

Dette har vært en lang, slitsom, interessant, og til tider spennende prosess, som jeg aldri ønsker å gjøre igjen, og stort mer er det ikke å si.

15.september 2023

Edvart Granseth

Sammendrag

Denne studien har sett på programmeringsoppgaver på 9. trinn i det digitale læreverket Kikora med fokus på kognitive krav, oppgavetyper og affordances og constraints. Gjennom en grundig analyse av to temaer, Fra blokk til tekstprogrammering og Programmere Pytagoras, ble det identifisert viktige funn. Analysen avslørte at det er en betydelig overvekt av oppgaver som krever lavere nivåer av kognitiv tenkning, spesielt innenfor kategorier som memorering og prosedyrer uten sammenhenger. Imidlertid var det mangel på oppgavetyper som lar eleven skape egne koder, forklare hvordan koder fungerer med egne ord og oppgavetyper som lar elevene lage hypoteser om hva som kommer til å skje når en kode kjøres. En viktig begrensning som ble påpekt, var mangelen på formativ evaluering. Tilbakemeldinger begrenset seg i stor grad til å vurdere om oppgaven var løst riktig eller ikke, og kunne ikke gi innsikt i elevenes prosess eller misoppfatninger.
På den positive siden ble fordelen av selvrettende oppgaver understreket, noe som sparer tid for lærerne, men samtidig begrenser deres evne til å identifisere elevenes utfordringer. Konklusjonen indikerer at det er behov for en større variasjon av oppgavetyper som utfordrer høyere nivåer av kognitiv tenkning for å støtte både programmeringskompetanse og kritisk tenkning blant elevene. Denne studien fokuserer på kun to temaer som utgjør en liten del av programmeringsoppgavene, og gir dermed også en begrenset innsikt i Kikoras programmeringsoppgaver.

Abstract

This study has investigated programming tasks for 9th-grade students within the digital learning platform Kikora, with a focus on cognitive requirements, types of tasks, and affordances and constraints. Through a thorough analysis of two themes, From Block to Text Programming and Programming Pythagoras, significant findings were identified. The analysis revealed a significant prevalence of tasks that require lower levels of cognitive thinking, especially within categories such as memorization and context-less procedures. However, there was a lack of task types that allow students to create their own code, explain how code works in their own words, and task types that enable students to make hypotheses about what will happen when code is executed. An important limitation that was pointed out was the lack of formative assessment. Feedback was largely limited to assessing whether the task was solved correctly or not and could not provide insight into students' processes or misconceptions. On the positive side, the benefit of self-assessing tasks was emphasized, which saves time for teachers but at the same time limits their ability to identify students' challenges. The conclusion indicates the need for a greater variety of task types that challenge higher levels of cognitive thinking to support both programming skills and critical thinking among students. This study focuses on only two themes, which constitute a small part of the programming tasks in Kikora, thus providing limited insight into Kikora's programming tasks.

Innholdsfortegnelse

1. Innledning

I 2016 kom stortingsmelding 28, Fag – Fordypning – Forståelse En fornyelse av Kunnskapsløftet Meld. St. 28 (2015–2016). Dette var et dokument som påpekte at samfunnet hele tiden er i endring og foreslo at man burde endre innholdet som brukes i skolen for at det skal være mer relevant sett i lys av dagens samfunn.

> Samfunnet er i endring og endringstakten øker på mange samfunnsområder. Store deler av arbeids livet preges sterkt av den teknologiske utviklingen. Teknologiske endringer og nyvinninger vil bidra til å endre digitale og fysiske produksjons prosesser for både varer og tjenester. Produktivitetsveksten i Norge avhenger av evnen til å utnytte ny teknologi som i stor grad skapes utenfor landet. Meld. St. 28 (2015–2016): 6

Dette sitatet er hentet fra denne stortingsmeldingen. en av grunnene til at man ønsket å gjøre dette var som nevnt i sitatet på grunn av at samfunnet vi lever i går mot en mer og mer digital hverdag hvor man er avhengig av å være i stand til å beherske det digitale. Det påpekes at den digitale og teknologiske utviklingen skaper endringer i skolefagene, og det er behov for å ha et større fokus på digitale ferdigheter. Det nevnes også at i andre land blir det et større fokus på digitale ferdigheter og at de vil at elevene skal produsere og forstå ikt istedenfor at de bare blir konsumenter av ikt. Dette tydet på at Norge hang etter når det kom til digitale ferdigheter og de viser til en undersøkelse fra 2013 som understreker dette. Det blir påpekt at i undersøkelsen var nærere en fjerdedel av elevene på 9.trinn som hadde digitale ferdigheter som ble sett på som så svake at de mest sannsynlig ville få problemer med å delta fullt i senere jobb og utdanning. Videre i stortingsmeldingen argumenteres det for at digitale ferdigheter bør ses i sammenheng med fagene der de er relevant, og det bør skilles tydeligere mellom ulike deler av digitale ferdigheter. Det legges også vekt på at digitale ferdigheter skal være en integrert del av innholdet i fagene, og det skal vurderes hvilke fag som har hovedansvar for ulike sider ved digitale ferdigheter. De sier også at kunnskapsdepartementet støtter dette synspunktet og vil videreføre digitale ferdigheter som en grunnleggende ferdighet i opplæringen. Meld. St. 28 (2015–2016) Altså med innføringen av den nye læreplanen eller LK20 (Kunnskapsdepartementet, 2019) som den også blir kalt, så fikk programmering og koding en betydelig og helt klart etterlengtet plass både i

matematikkfaget, men også i skolen generelt. Utdanningsdirektoratet eller Udir sier også blant annet dette om hvorfor programmering er i de nye læreplanene.

> Algoritmisk tenkning og programmering er nye elementer i siste revisjon av læreplanene. Dette er sentral kompetanse for fremtiden. Algoritmisk tankegang er en problemløsningsmetode som dreier seg om å tilnærme seg problemer på en systematisk måte og kunne foreslå løsninger som kan bruke datamaskiner til å løse hele eller deler av dem. Programmering i skolen handler mye om å la elevene være kreative og skapende med teknologi (Kunnskapsdepartementet, 2023).

I den nye læreplanen for matematikk (Kunnskapsdepartementet, 2019) står det ikke veldig mye om Programmering eller koding eksplisitt, men det er fortsatt tydelig at programmering har vært i fokus når LK20 ble skrevet, og dette gjenspeiles i flere deler av den nye læreplanen for matematikk.
i Kjerneelementene i den nye læreplanen ser man spesielt tydelig at programmering har vært i tankene selv om programmering ikke nevnes eksplisitt her. Det blir nemlig i kjerneelementene understrekt spesielt viktigheten av algoritmisk tenkning som en viktig del når det gjelder å utvikle strategier og fremgangsmåter for problemløsning. Dette stemmer veldig overens med det som står i sitatet over, men det som ikke nevnes eksplisitt er at algoritmisk tenking er også sterk knyttet oppimot programmering i form av at dette er en av de mest sentrale måtene man jobber med programmering og koding. Algoritmisk tenking er noe jeg vil gå nærmere inn på senere i denne oppgaven. Kompetansemålene er en annen plass programmering kommer tydelig fram. kompetansemålene skal dekke alt en elev skal kunne etter de diverse trinnene, og fra femte klasse og oppover finner man kompetansemål som direkte går på å bruke programmering i diverse sammenhenger knyttet til matematikk. For eksempel ligger dette kompetansemålet under 9.trinn «simulere utfall i tilfeldige forsøk og beregne sannsynligheten for at noe skal inntreffe, ved å bruke programmering» (Kunnskapsdepartementet, 2019). Under grunnleggende Ferdigheter finner man også programmering og da Under kategorien digitale ferdigheter. I digitale ferdigheter blir programmering nevn som en av hovedpunktene uten at det blir gått noe dypere inn i det. Den økte vekten som LK20 har lagt på programmering i matematikkfaget er et tydelig grep for å forberede elevene til en fremtid hvor teknologi spiller en sentral rolle i både arbeidsmarkedet og samfunnet generelt.

I en artikkel produsert av Høgskolen i Østfold (Forskning.no, 2020) diskuteres utfordringene lærere kan stå overfor når de forsøker å integrere programmering i matematikkundervisningen. Selv om artikkelen er fra 2020, så er den fortsatt veldig relevant, da programmering som nevnt fortsatt er et relativt nytt innhold i skolen. En av de mest fremtredende utfordringene i artikkelen, er den manglende kompetansen blant lærere når det gjelder programmering. Dette er en bekymring som er langt fra å være utdatert, da mange lærere fremdeles kan ha begrenset erfaring med programmering og mangler nødvendige ferdigheter for å veilede studentene effektivt. Videre påpeker artikkelen begrensningene i forskningen på dette feltet. Det er behov for mer studier og forskning som kan si noe om hvordan i praksis man kan bedre integrere programmering i matematikkundervisningen. Dette er viktig for å forstå hvordan studentenes læring og argumentasjonsevner kan forbedres gjennom programmering.
Artikkelen refererer også til en studie (Kaufmann & Stenseth, 2021) som har forsket på hvilke argumenter elever presenterer når de løser oppgaver med programmering. Kaufmanns forskning tydeliggjorde viktigheten av lærerens kompetanse innen programmering for å kunne veilede studentene på en effektiv måte. Dette understreker behovet for opplæring av lærere og støtte for å sikre at lærere er rustet til å mestre utfordringene knyttet til programmering i matematikkundervisningen.

Min egen erfaring i skolen har gitt meg et førstehåndsinnblikk i de utfordringene lærere kan møte når de prøver å innføre programmering i klasserommet. Videre har deltakelse i programmeringskurs for lærere bidratt til å forstå de praktiske utfordringene som lærere står overfor når de tar fatt på dette komplekse emnet. En av de mest fremtredende observasjonene i min skoleerfaring er lærernes tendens til å lene seg tungt på forhåndsopprettede ressurser når de underviser i programmering. Dette går for det meste på ferdiglagde læremidler og digitale plattformer som tilbyr ferdige programmeringsoppgaver. Dette er en av grunnene til at jeg har valgt å fokusere analysen på en slik ressurs.

Når det kommer til slike ressurser så er det mange og velge mellom. For eksempel har man noe som heter Lær Kidsa Koding (U.å.). De kaller seg selv for en frivillig bevegelse som har som mål å lære barn koding. De har mange oppgaver knyttet både til forskjellige former for programmering og til forskjellig utstyr som skoler kan ha. En annen digital ressurs er multi smartøving (Gyldendal, U.å.). Det er en oppgavebank knyttet til matematikk som selv

regulerer hvilke oppgaver elever får ettersom hvordan de presterer. Begge disse har ferdiglagde oppgaver som en lærer kan la elevene jobbe med.

1.1 Problemstilling og forskningsspørsmål

På grunn av det som står i den innledende delen og fordi det her er noe jeg selv finner interessant har jeg har valgt i denne oppgaven å skrive om programmeringsoppgaver. Den resursen jeg har valgt å se på oppgaver fra heter Kikora (U.å.) og er et digitalt læreverk knyttet spesifikt til matte. Fra dette læreverket har jeg valgt to temaer fra 9.trinn som begge går under programmering. Jeg har selv sett dette læreverket bli brukt i skolen, derfor er dette veldig relevant å se på fordi jeg vet innholdet brukes i skolen i dag. Problemstillingen jeg har satt er da som følger:

«Hvordan er programmeringsoppgavene for 9.trinn i Kikora»

Dette i seg selv er en veldig vid problemstilling og det er mange innfallsvinkler man kan ta. Jeg har derfor valgt å se på oppgavene i lys av to analyseverk. Det ene baserer seg på kognitive krav (Stein & Smith, 1998) altså hva som kreves kognitivt av eleven. Det andre ser på hva slags oppgavetyper innen programmering (Bråting & Kilhamn, 2022) som man finner. altså oppgavetyper som man spesifikt finner i programmering. Utfra dette har jeg laget disse forskningsspørsmålene som da danner et grunnlag til analysen.

- Hvilke kognitive krav stilles av oppgavene I Kikora
- Hvilke programmerings oppgavetyper finner man på Kikora
- Hvor stor forskjell er det mellom temaet Fra blokk til tekstprogrammering og temaet Programmere Pytagoras innen bruk av oppgavetyper og kognitive krav

Etter å ha gjennomført analysen tilknyttet forskningsspørsmålene over var det flere funn som var interessante og relevante sett i lys av problemstillingen, men som falt utenfor disse forskningsspørsmålene. Jeg har derfor valgt å legge til de to forskningsspørsmålene under for å styrke analysen.

- Hvilke kognitive krav stilles i de forskjellige programmeringsoppgavetypene i Kikora?
- Hvilke affordances og constraints finnes i programmeringsoppgavene i Kikora?

1.2 Oppgavens struktur

Denne oppgaven er delt inn i 8 kapitler

Innledning
I denne delen presenteres selve studien, samt problemstilling, forskningsspørsmål og sentrale begreper som videre er relevant.

Teori og tidligere forskning
I dette kapitelet presenteres analyseverktøyene som er brukt i selve analysen, samt relevant teori og tidligere forskning

Metode
Metodekapitlet tar for seg metoden som er brukt og presenterer analyseobjektet. Her blir også selve analyseprosessen gått nøye igjennom og det blir sett på oppgavens Relabilitet og validitet

Resultat
I dette kapitelet blir alle resultatene og funnene i analysen presenter systematisk sammen med eksempeloppgaver for å belyse funnene

Drøfting
I drøftingskapittelet drøftes funnene oppimot rammeverket, relevant teori, tidligere forskning og egne tanker rundt funnene.

Konklusjon
I dette kapitelet blir det gitt en konklusjon på problemstillingen og forskningsspørsmålene blir besvart

Områder for videre forskning
Her presenteres mulige områder for videre forskning sett i lys av denne oppgaven funn og begrensinger.

Referanser
Det siste kapitelet viser alle referanser og kilder brukt i denne masteroppgaven

1.3 Begrepsforklaring

I denne oppgaven er det flere begreper og ord som konsekvent brukes gjennom hele oppgaven. Det er ikke alle punktene her som er nødvendig å forklare, men dette gjøres for å gi en mest mulig klar og tydelig oversikt over denne studien så det er

Tema

Kikora kaller sine oppgavesett for temaer jeg har derfor valgt gjennom hele oppgaven å konsekvent bruke ordet Tema for å referere til disse for å ha det mest mulig oversiktlig

Nivå

Begrepet nivå brukes i hovedsak i forbindelse med rammeverket for Kognitive krav (Stein & Smith, 1998). Både i metodekapittelet og i resultatkapitelet er nivå kun brukt i forhold til kognitive krav. Nivå 0 til 4 refererer spesifikt til nivåene i de kognitive kravene. Det er kun et unntak hvor nivå blir brukt i en annen sammenheng og det er i forhold til Blooms taksonomi.

Formativ og summativ vurdering

«Der formativ vurdering peker mot fremtidig læring, peker summativ vurdering mot en allerede gjennomført prestasjon» (SNL, 2021). For eksempel en tilbakemelding fra en test som sier at du klarte 46 av 50 oppgaver er en summativ tilbakemelding. En formativ vurdering vil derimot påpeke hva du har gjort feil og hva du kan gjøre videre for å bli bedre.

2. Teori og tidligere forskning

I dette kapitelet blir det først gått gjennom hva algoritmisk tenking etterfulgt av rammeverkene brukt i analysene. deretter kommer mer relevant teori

2.1 Computational thinking og Algoritmiske tenkeren

Computational thinking (Grover & Pea, 2013) eller ofte forkortet CT og er et begrep som stadig kommer opp i sammenheng med programmeringsundervisning. Grunnen til dette er at Computational thinking er et helt sentralt når det kommer til hvordan man skal tenke når man programmerer eller koder. Computational thinking handler om måten man tenker på når man møter forskjellige problemer og hvordan man setter det opp for å finne løsningen. Dette innebærer blant annet at man systematisk går gjennom informasjonen man får og hvordan man videre deler opp, strukturer og setter opp problemet. I programmering så vil en del av dette være å sette opp problemet på en måte som gjør at en datamaskin eller et program kan lese og løse problemet. Ofte i form av gitte algoritmer (Grover & Pea, 2013). Computational thinking handler i hovedsak om hvordan man skal bearbeide problemer. Altså analyserer problemet og finne ut av hva av informasjonen som er viktig, forså å sette det opp på en måte som er mer hensiktsmessig for å finne en løsning på problemet som er gitt.

Den algoritmiske tenkeren (Utdanningsdirektoratet, 2019) er en modell som viser hvordan Man kan jobbe med programmering i skolen. Algoritmisk tenking tilsvarer nesten det samme som Computational thinking på Engelsk. Det er det nærmeste vi kommer på norsk, men det stemmer ikke helt fordi definisjonen på begrepet algoritmisk tenking kan variere, men selv om definisjonen kan variere så er det noen hovedområder som går igjen og disse kommer tydelig frem i den algoritmiske tenkeren. Den algoritmiske tenkeren er en modell som påpeker forskjellige begreper og arbeidsmetoder som er viktige innen programmering. Begrepene som nevnes er:

- Logikk: Dette handler om å analysere og forutsi, det vil si det samme som å lese gjennom koden og prøve å forstå hva som vil skje når koden blir kjørt.

- Algoritmer: Dette tar for seg reglene og den stegvise utførelsen av en kode, altså hvordan en kode fungerer.
- Dekomposisjon: Dette handler om å dele opp koden i mindre og mer håndterbare deler. Dette gjør man både for å strukturere hvordan man jobber med koden, og for å sette fokus på mindre deler så man ikke tar for seg for mye om gangen.
- Mønstre: Dette går på å identifisere likheter og ulikheter innad i koden. Dette gjør det både for å letere forstå hvordan en kode fungerer, og lettere å effektivisere sine egne koder.
- abstraksjon: Dette innebærer evnen til å fjerne uviktige detaljer for å bedre kunne se helheten i koden og finne ut hvordan dens funksjoner fungerer.
- Evaluering: Dette handler om å kunne ta egne vurderinger og reflektere over løsninger.

Arbeidsmåter som nevnes i den algoritmiske tenkeren er:

- Fikle: Dette er en arbeidsmetode som involverer utforskning og eksperimentering med eksisterende koder.
- Skape: Dette omhandler evnen til å skrive og designe egne koder.
- Feil søke: Dette innebærer å analysere eksisterende koder, enten egne eller andres, for å finne ut hvorfor koden ikke fungerer som den skal. Et eksempel på hvordan denne arbeidsmetoden kan brukes er at læreren kan gi elevene en kode med feil og sier hva den egentlig skal gjøre så må elevene selv fikse koden
- Holde ut: Dette handler om å ikke gi opp selv om man møter på vanskeligheter. Dette kan være for eksempel være at koden ikke gjør det du vil eller at det er en feil i koden som ikke er veldig lett å finne.
- Samarbeide: Dette går på viktigheten av å dele arbeidet og samarbeide med andre, slik at man kan få innspill, støtte eller et nytt syn på problemet i løsningsprosessen.

Disse begrepene og arbeidsmetodene er grunnleggende innenfor programmering og utgjør kjernen av den algoritmiske tenkerens tilnærming til problemløsning i programmering. Gjennom å utvikle disse evnene og tilnærmingene kan elever bli i stand til å bli dyktigere og mer kritiske programmerere som kan møte utfordringene innenfor digital teknologi på en mer strukturert, effektiv og reflektert måte. Som nevnt i innledningen så er algoritmisk tenking nå en sentral del av kjerneelementene i læreplanen i matematikk (Kunnskapsdepartementet, 2019) og den algoritmiske tenkeren viser veldig godt hva dette innebærer for elevene.

Algoritmisk tenking (Utdanningsdirektoratet, 2019) er ofte sammenliknet med Computational thinking (Grover & Pea, 2013). Dette er ikke veldig overaskende siden begge handler om hvordan man skal angripe en oppgave og har helt tydelige fellestrekk, spesielt med tanke på det å dele opp problemet i mindre og mer håndterlige deler. Det er heller ikke utenkelig at algoritmisk tenking til en viss grad har tatt utgangspunkt i deler fra Computational thinking

2.2 Forskning implementering av programmering I svensk skole

Siden det bare er noen få år siden programmering kom inn i den norske skolen så er det fortsatt lite forskning som er kommet på dette området, men hvis vi ser utenfor Norge så er det mye mer forskning på dette feltet. Sverige blant annet, fikk programmering inn i skolen noen år før Norge. kajsa bråting og cecilia kilhamn (2022) har i sin artikkel sett på implementeringen av programmering i den svenske skolen. De skriver at i Sverige ble programmering satt inn som en del av algebra, og at dette skulle inn i alle klassetrinn. De påpeker også at dette er overaskende siden programmering ikke har vært nevnt i en signifikant grad innen tidligere forskning knyttet til algebraisk tenkning. I artikkelen ser de på innholdet knyttet til programmering som er utgitt i svenske lærebøker for barneskolen. Målet de hadde med denne analysen var å få en bedre forståelse av innholdet knyttet til programmering i de svenske skolebøkene og hva som karakteriserer dette innholdet. I tillegg ønsket de å se hvordan og til hvilken grad oppgaver i programmering knyttet programmering til det matematiske innholdet.

de har sett på programmeringsoppgaver i fire forskjellige læreverk med lærebøker, og valgt å ta for seg de som er rettet mot 1-6. trinn. Totalt så de på 390 programmeringsoppgaver. De har deretter sett på hva slags oppgaver man kan finne i lærebøkene knyttet til programmering og plassert de innen noen kategorier. Disse kategoriene er:

Følge

i denne typen oppgaver får elevene en prosedyre eller en metode som skal følges for å løse oppgaven. Dette kan være alt fra en oppskrift som forteller steg for steg hvordan oppgaven skal løses til et mønster eller pattern som det heter på engelsk, som elevene skal fortsette. For eksempel hvis elevene får i oppgave å løse en matteoppgave ved hjelp av programmering og det er en instruksjon som viser steg for steg hvordan dette gjøres så vil det være en følge oppgave.

Finne ut

Finne ut er oppgaver som går på å finne prosedyren eller regelen som passer for å løse oppgaven. Her er ikke prosedyren gitt, men elevene skal bruke det de har lært til å finne ut svaret selv. Det kan være og gjøre om på en formel for å finne svar på noe annet enn det den originale formelen gir. For eksempel gjøre om Pytagoras læresetning til å finne en katet istedenfor hypotenusen.

Finne feil

Finne feil er det samme som det man kaller debugging på engelsk. Det går ut på å se gjennom en kode å finne ut hvorfor den ikke fungerer som den skal. Her kan man se etter alt fra skrivefeil og feil tegnsetting til det og ha brukt feil kommando. Altså hva må endres for at programmet skal lese koden og gjøre det du vil at den skal gjøre.

Forme og skape

Denne typen oppgave går ut på å lage en egen kode. Her skriver man selv en kode istedenfor å jobbe med eller bearbeide allerede eksisterende koder. Dette kan være koder som gir elevene et problem hvor de selv skal konstruere en kode som løser problemet.

Forklare

Disse oppgavene ber eleven å forklare noe. Dette kan være hvordan en kode fungerer, eller hva en kommando gjør.

Se for seg

Se for seg er ber elevene gi en hypotese på hva de tror kommer til å skje. Her må elevene lese gjennom en allerede eksisterende kode og prøve å forstå hva den gjør før de faktisk kjører koden.

De fant ut at det var en stor andel av oppgavene som kun gikk på å følge prosedyrer eller steg for steg instruksjoner. Det var ca. 200 av oppgavene som gikk under følge prosedyre. De skriver at dette ikke var en veldig stor overraskelse gitt at steg for steg instruksjoner er nevnt i den svenske læreplanen på 1-3. trinn. Det var også en stor andel oppgaver under forme og skape, ca. 140 av oppgavene gikk under forme og skape.
Et annet funn er at det var veldig få oppgaver som gikk på feilsøking eller debugging, kun ca. 20. De nevner at dette er overaskende siden feilsøking er en fundamental del når kommer til programmering. finne regel hadde ca. 70 oppgaver. Forklare hadde ca. 25 oppgaver og se for

seg hadde bare 8 oppgaver.

De skriver at det i virker som lærebøkene fokuserer mye på noen av konseptene innen programmering, og nesten utelukker andre som er like viktige innen programmering. I Sverige så er programmering fordelt på matematikk og naturfag, og dette nevnes som en mulig grunn til at det er gjort på denne måten (Bråting & Kilhamn, 2022).

2.3 Kognitive krav

I skolen møter elevene en rekke ulike oppgaver som kan stille varierende krav til elevenes ulike evner. Spesielt oppgavene i matematikkfaget vil variere når det kommer til hva som kreves av elevens evne til å tenke og oppfatte sammenhenger mellom matematiske elementer. Disse kravene kan vi kalle for kognitive krav (Stein & Smith, 1998). En oppgave som stiller lave kognitive krav vil naturlig ikke kreve mye tenking eller forståelse av elevene, og vil dermed være oppgaver som er ganske rett fram i hvordan man løser de, mens en oppgave som stiller høyere kognitive krav vil være det motsatte og kreve større grad av blant annet refleksjon fra elevene.

I artikkelen «Mathematical Tasks as a Framework for Reflection: From Research to Practice» av Stein og Smith (1998) skrives det om fire ulike nivåer av kognitive krav man kan møte i matematikkoppgaver. I artikkelen skilles det mellom lavere og høyere nivåer av kognitive krav. På det lavere nivået finner vi kravene om memorering og prosedyrer uten sammenheng. Dette innebærer at elevene må huske fakta og regler uten å se hvordan de henger sammen med andre matematiske begreper. På det høyere nivået finner vi derimot kravene om prosedyrer med sammenhenger og det å utføre matematikk. Dette innebærer at elevene må kunne anvende og forstå matematiske prosedyrer i en større sammenheng og aktivt utforske og engasjere seg i matematikken. Altså ved å ta i bruk dette rammeverket når man ser på matematikkoppgaver så kan man se hvor mye som kreves av elevene i hver oppgave.

> «The framework is not meant to be a rigid prescription; rather, it is a tool for reflection. When used well, it should draw attention to what students are actually doing and thinking about during mathematics lessons. This focus on student thinking, in turn,

> helps the teacher adjust instruction to be more responsive to, and supportive of, students' attempts to reason and make sense of math» (Stein & Smith, 1998)

Dette betyr med andre ord at rammeverket er laget for at man skal reflektere rundt oppgavene man gir elevene, og at dette vil kunne gjøre at man forstår mer av hvordan elevene oppfatter oppgavene, så man kan endre eller tilpasse seg så man kan støtte oppunder elevens forsøk på matematisk forståelse.

2.3.1 Memorering nivå 1

Memorering handler om det og huske informasjon som formler, definisjoner og annen relevant kunnskap og kunne gjengi dette uten behov for dypere forståelse. Oppgaver som kun stiller krav til memorering, er såpass enkle at de ikke krever prosedyrer for å løses. De skaper heller ikke sammenhenger eller legger til annen kontekst. Altså det blir ikke bygget relasjoner eller gitt noen relasjoner eller forbindelser til andre definisjoner eller konsepter innen matematikk. Et eksempel på en slik oppgave eller spørsmål kan være: hvilke år fikk Norge sin grunnlov? I denne oppgaven er det ingen krav eller behov for dypere matematisk resonnement eller problemløsning, kun det å kunne gjengi et spesifikt årstall (Stein & Smith, 1998).

2.3.2 Prosedyrer uten sammenhenger nivå 2

I oppgaver som faller under kategorien prosedyrer uten sammenhenger, blir det heller ikke etablert noen forbindelser til andre definisjoner eller konsepter. I disse oppgavene så blir det gitt en spesifikk prosedyre eller metode som elevene skal bruke for å finne svar på oppgaven. Forskjellen på disse oppgavene og de som går under memorering er at her kan ikke elevene bare se svaret eller repetere allerede kjent informasjon. Det kreves faktisk at de bruker metoden som blir gitt for å komme frem til løsningen. Med andre ord i disse oppgavene blir det fokusert mer på å anvende en spesifikk metode eller strategi, og elevene må følge denne metoden for å løse oppgaven på riktig måte. Dette krever en litt større grad av forståelse og evne til å anvende prosedyren i praksis (Stein & Smith, 1998).

2.3.3 Prosedyrer med sammenhenger nivå 3

I oppgavene som går under kategorien prosedyrer med sammenhenger, møter man også prosedyrer og metoder, men det legges større vekt på å skape sammenhenger og en dypere forståelse i det aktuelle emnet. Her går fokuset bort fra selve prosedyren eller metoden i seg selv, og rettes i stedet mot å etablere forbindelser og sammenhenger til andre kontekster, definisjoner eller konsepter innenfor matematikk. Disse oppgavene har som mål å stimulere elevenes evne til å se hvordan ulike matematiske begreper og ideer henger sammen og påvirker hverandre. Det handler om å forstå hvordan prosedyren eller metoden passer inn i en større sammenheng og hvordan den kan brukes til å løse problemer på en mer helhetlig måte. Dette bidrar til å utvikle en dypere forståelse for matematikkens struktur og sammenhenger (Stein & Smith, 1998).

2.3.4 Å gjøre matematikk nivå 4

På nivå 4 av kognitive krav å gjøre matematikk, kreves det mer av elevene og at de kan skape en dypere forståelse av emnet, definisjoner og konsepter som er knyttet til oppgaven. I disse oppgavene er det ikke nødvendigvis en spesifikk metode eller prosedyre som skal brukes for å løse dem. Hensikten med disse oppgavene er at elevene selv må utforske og se sammenhenger. De må analysere og forstå problemet grundig, bruke sin kunnskap til å utforske ulike tilnærminger og finne en løsning. Elevene kan også bli oppfordret til å forklare hvordan de har tenkt. Altså begrunne sine valg og argumentere for hvorfor de tenker slik de gjør (Stein & Smith, 1998).

2.3.5 Tidligere bruk av rammeverket for kognitive krav

Selv om de skriver at rammeverket er ment som et verktøy for å reflektere rundt oppgavene og hva som kreves av elever, så har det blitt brukt tidligere hvor man har funnet ut at bruk av kognitive krav kan påvirke elevers matematiske forståelse. Stein og Smith (Stein & Smith, 1998) refererer til bruk av rammeverket i det de kaller QUASAR prosjektet. «QUASAR (Quantitative Understanding: Amplifying Student Achievement and Reasoning) is a national reform project aimed at fostering and studying the development and implementation of enhanced mathematics instructional programs in six urban middle schools.» (Stein & Smith,

1998). Dette var et prosjekt som gikk over 5 år og ble gjennomført i USA fra 1990 til 1995. i dette prosjektet ble det undersøkt nøye hvilke oppgaver som ble brukt i undervisning i flere klasserom. De oppdaget en forskjell på klasserom ettersom hvilke nivå oppgavene de pleide å få gikk under. Elevene som befant seg i klasserom hvor læreren ga oppgaver i større grad som krevde høyere kognitive krav, oppnådde bedre resultater når det gjaldt problemløsning og resonering (Stein & Smith, 1998).

2.4 Blooms taksonomi

Blooms taksonomi (Adams, 2015) er en modell som pedagoger kan bruke for å formulere læringsmål, med det mål å konkretisere hvilke ferdigheter og evner de ønsker at elevene skal beherske og utrykke. Modellen deler kognitive ferdigheter inn i en hierarkisk struktur med ulike nivåer, hvor hvert nivå legger til mer sofistikerte eller krevende tankeprosesser. Denne hierarkiske strukturen gjenspeiler en form for pyramide hvor de lavere nivåene danner grunnlaget for de høyere, og hvert høyere nivå bygger på de tidligere.

Kunnskap

Dette er det laveste nivået i pyramiden. Det krever at elevene husker og gjengir informasjon som fakta, definisjoner eller konsepter. Dette kan være i form av gjenkjenning eller henting av informasjon.

Forståelse

Dette nivået innebærer at elevene kan tolke og reprodusere informasjon ved å forklare prinsipper, klassifisere elementer i grupper, eller sammenligne og kontrastere forskjellige begreper.

Anvendelse

På dette nivået bruker elevene kunnskap, ferdigheter eller teknikker i nye sammenhenger. Dette kan inkludere løsning av praktiske problemer ved å anvende lært materiale på nye måter.

Analyse

Analyse krever at elevene bryter ned informasjon i mindre komponenter, oppdager mønstre og relasjoner, og identifiserer grunnlaget for argumenter.

Syntese

Dette nivået involverer kombinasjonen av ulike elementer av kunnskap og ferdigheter for å skape noe nytt og originalt. Det kan inkludere utvikling av nye ideer, formulering av spørsmål, eller design av nye tilnærminger.

Evaluering

Dette er det høyeste nivået i pyramiden. Her evaluerer elevene informasjon, argumenter eller resultater kritisk. De bedømmer verdien eller relevansen av noe basert på en grundig analyse.

Blooms taksonomi understreker at man må ha mestret de lavere nivåene for å kunne stige opp til de høyere. Med andre ord, for å kunne analysere må man først ha forstått, og for å kunne syntetisere må man ha analysert. Denne strukturen bidrar til en dypere og mer sammenhengende læringserfaring (Adams, 2015).

I forhold til kognitive krav

Blooms taksonomi (Adams, 2015) har mange likheter med rammeverket for kognitive krav (Stein & Smith, 1998) og man kan se en tydelig likhet innad i nivåene. For eksempel så har nivået Kunnskap fra Blooms taksonomi helt tydelig likheter til nivået Memorering fra de kognitive kravene. Begge fokuserer på å ta til seg og huske informasjon. Det er også tydelige likheter mellom de høyere nivåene. I Begge teoriene eller rammeverkene fokuserer de høyere nivåene på at elevene selv skal reflektere og ta egne valg. Den store forskjellen ligger i at de kognitive krav er et rammeverk som brukes til å undersøke hva som kreves av elevene i hver oppgave, mens Blooms taksonomi forklarer et hierarki hvor man må mestre de lavere nivåene før man kan gå videre til høyere nivå.

2.5 Affordanes and constraints

Affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014) er to begreper man fort møter i sammenheng med programmering. Affordances handler i hovedsak om hvilke utfall man kan se for seg ut ifra den informasjonen man har tilgjengelig. Hvis man skal oversette til norsk så er det nærmeste man kommer muligheter. I motsetning til begrepet muligheter som ofte blir brukt som et positivt begrep, så er ikke Affordances knyttet oppimot hvor vidt et utfall blir sett på som negativt eller positivt. Hvis man for eksempel tar for seg en stekeovn og undersøker den så kan man finne flere forskjellige Affordances. En stekeovn gir deg for

eksempel mulighet til å varme og tilberede mat, men den gir deg også muligheten til å brenne både deg selv og maten. Dette er noen forskjellige affordances knyttet til den informasjonen man har tilgjengelig om en stekeovn. Det er også en mulighet for at det kan skje noe helt uventet, og det vil også kunne regnes for å være en affordance fordi alle mulige utfall vil gå under affordances selv om man ikke kan se for seg utfallet, men det som det oftest ses på eller blir satt fokus på er perceived affordances. Med andre ord er dette de utfallene man kan se for seg at skjer. Dette er naturlig fordi det er vanskelig å skrive om noe man ikke vet eller ikke kan se for seg. Det er også disse jeg kommer til å se på videre i oppgaven av affordances (Humble & Mozelius, 2023; Gibson, 2014).

Constraints eller begrensninger kan ses på som det motsatte av affordances. Dette er fordi det ser på hva som ikke er mulig eller hvor grensen går for hva man kan gjøre. I likhet med affordances så kan constraints være både negative og positive (Gibson, 2014). constraints kan være like viktige og ha like mye å si som det affordances har. Hvis man ser på constraints i forhold til eksemplet om stekeovnen så noen constraints være for eksempel at den trenger strøm eller størrelsen på stekeovnen, altså man kan ikke bruke den hvor som helst og det er begrenset hvor mye man får plass til i stekeovnen.

Det er også viktig å se på dette fra forskjellige vinkler fordi hvor vidt noe blir sett på som en constraint eller en affordance kan variere. For eksempel en stekeovn kan veie mye og dermed være vanskelig å ta med seg rundt, men det faktum at den kan tas med rundt er en affordance, samtidig som vekten er en constraint som kan begrense hvor langt man flytter den og hvor man tar den med seg.

2.6 Primm

PRIMM (Sentance et al, 2019) er en teori som tar for seg hvordan man kan strukturere programmerings undervisning, ved jobbe med eksisterende programmering og kodeeksempler. PRIMM-teorien er en teori som er bygget på tidligere forskning rundt programmering. PRIMM i seg selv er et akronym som står for Predict, Run, Investigerte, Modify og Make, og hvert av disse ordene representerer ulike trinn i en tilnærming til hvordan man kan jobbe med programmering. De begynner med å forutsi eller anta hvordan en gitt kode

vil oppføre seg, deretter utfører de koden for å observere resultatene. Etter det utforsker de kodeeksemplet grundig for å forstå hvordan koden fungerer. Basert på denne undersøkelsen, gjør de endringer og modifiserer for å se hvordan det påvirker koden og resultatene. Til slutt lager de egne programmer og utvider sin kompetanse. PRIMM-teorien gir dermed en strukturert tilnærming som oppmuntrer elevene til å tenke kritisk, utforske og eksperimentere i programmeringsprosessen (Sentance et al, 2019).

2.6.1 Predict

Predict-trinnet er det første steget i PRIMM-teorien, hvor man starte og undersøker koden og prøver å få en forståelse for dens funksjon. Her handler det om å prøve å forutsi hva som vil skje eller med andre ord skape hypoteser om hva sluttresultatet av koden vil være. Elevene analyserer og identifiserer blant annet kodens struktur og andre viktige komponenter for så å prøve å se hva slags påvirkning de har på koden. Her må elevene bruke sin eksisterende kunnskap og erfaring, samt observere mønstre og sammenhenger i koden, for å kunne formulere hypoteser om hvordan koden vil oppføre seg og hvilket resultat den vil gi. Predict-trinnet oppfordrer elevene til å tenke kritisk, bruke logisk resonnement og visualisere utførelsen av koden før den faktisk kjøres. Dette bidrar til å utvikle deres evne til å forutsi og planlegge programmeringsløsninger på forhånd (Sentance et al, 2019).

2.6.2 Run

Run-trinnet, også kjent som kjør på norsk, er det enkleste steget i PRIMM-teorien. Her begynner elevene med en gang og kjører koden for å se hva som faktisk skjer. Dette trinnet innebærer å teste den hypotesen som ble formulert i Predict-trinnet, og bekrefte eller avkrefte om den var riktig eller ikke. Ved å kjøre koden, observerer elevene resultatene og sammenligner dem med det de forutså. Dette gir dem en konkret erfaring med hvordan koden fungerer i praksis. Hvis hypotesen viser seg å være korrekt, styrker det deres tillit til deres evne til å forutsi og forstå koden. Hvis hypotesen derimot er feil, gir det dem en mulighet til å reflektere over hva som gikk galt og justere deg deretter. Run-trinnet bidrar til å skape eksperimentering, refleksjon og læring. Gjennom denne praksisen utvikler de både sine tekniske ferdigheter og evnen til kritisk tenkning i programmeringsoppgaver (Sentance et al, 2019).

2.6.3 Investegate

I dette trinnet legger læreren opp til at elevene selv skal ta å undersøke koden grundig gjennom ulike oppgaver og utfordringer. Dette gir dem muligheten til å videreutvikle sin forståelse og kunnskap om koden og dens funksjoner. Læreren kan for eksempel gi en oppgave der elevene må gi en grundig forklaring på hvordan koden fungerer og beskrive hva som skjer når koden kjøres. Dette oppfordrer elevene til å uttrykke sin kunnskap på en sammenhengende og logisk måte, og bidrar til å styrke deres evne til å kommunisere om programmeringskonsepter.

En annen tilnærming i dette trinnet kan være at det på forhånd er lagt inn feil i koden som elevene må finne og løse for at den skal fungere som forventet. Dette oppfordrer dem til å være aktive problemløsere og utforske mulige feilkilder i koden. Her må de altså identifisere og rette feilene. Dette er med på å styrke elevene sin feilsøkingskompetanse og blir bedre til å håndtere utfordringer som kan oppstå når de selv skal programmere.

Investigate-trinnet legger dermed til rette for en dypere forståelse av koden ved å spille på elevenes nysgjerrighet og utforskningsevne. Ved å la elevene utforske på egen hånd, oppmuntrer PRIMM-metoden til læring og gir dem muligheten til å oppdage nye aspekter ved programmering. Dette trinnet er dermed en viktig del for å styrke elevenes evne til problemløsning og selvstendig utforskning innenfor programmeringsområdet (Sentance et al, 2019).

2.6.4 Modify

I Modify-trinnet oppfordres elevene til å ta del i programmeringsprosessen ved å endre og tilpasse koden, ofte skjer dette gjennom gitte utfordringer eller mål. Dette gir dem muligheten til å utøve sin kreativitet og bruke sin forståelse av hvordan koden fungerer og kodens struktur. For eksempel kan elevene få en oppgave der de skal justere en kode som får en lyspære til å blinke én gang og deretter endre den slik at den blinker to ganger i stedet. Dette krever at elevene identifiserer de relevante delene av koden som styrer blinkingen. Deretter må de finne ut hvordan de skal tilpasse koden for å oppnå at lyspæren blinker så mange ganger som de ønsker (Sentance et al, 2019).

Gjennom Modify-trinnet får elevene mulighet til å eksperimentere og teste ulike tilnærminger til programmeringsløsninger. De må vurdere hvordan endringene i koden påvirker dens funksjon, og gjennom denne erfaringen utvikler de en dypere forståelse for sammenhenger og konsekvenser i programmering. Det oppmuntrer også til en læreprosess der feil og feilsøking er en naturlig del av utforskningen (Sentance et al, 2019).

2.6.5 Make

I Make-trinnet får elevene muligheten til å ta fullt eierskap over koden og sin egen programmeringsprosess. Her skal de selv lage en helt ny kode, der de bruker oppsettet eller funksjoner de har lært fra den tidligere koden de har arbeidet med. Dette trinnet er en viktig del av PRIMM-metoden, da det oppfordrer elevene til å anvende den kunnskapen de har opparbeidet seg og koble den til sin egen kreativitet og ideer.

Ved å skape sin egen kode legges det opp til at elevene kan bli mer selvstendige problemløsere og programutviklere. De må tenke kritisk og planlegge hvordan de skal gå fram for å oppnå det ønskede resultatet. Samtidig utvikler de sine ferdigheter i å utforme kodestrukturer og logiske løsninger. Make-trinnet gir med andre ord elevene muligheten til å utforske sitt potensiale som programmerere og oppmuntres til å være kreative og eksperimentelle. Dette bidrar til å bygge selvtillit og styrke deres forståelse av programmering som et kreativt verktøy for problemløsning og uttrykk. Ved å skape sin egen kode, tar elevene også større kontroll over sitt eget læringsforløp og blir i større grad deltagende i sin egen utviklingen av sine programmeringsferdigheter. Dette trinnet skaper en dypere forankring av kunnskapen som blir tilegnet. Det gir også elevene en følelse av mestring og tilfredshet over å kunne realisere sine egne programmeringsprosjekter (Sentance et al, 2019).

3. Metode

I dette kapitlet gjennomgår jeg først generelt om metoden jeg har brukt, altså dokumentanalyse. Sammen med dette står det om nøyaktig hva som er analyseobjektet i oppgaven, og hvordan Læreverket Kikora er bygd opp. Etter dette går jeg gjennom selve analyseprosessen. Her vises nøyaktig hvordan analysen er gjennomført, samt valg som er tatt under analyseprosessen, og hvorfor disse valgene er tatt. Til slutt i dette kapitelet tar jeg opp og både drøfter og forklarer rundt oppgavens relabilitet og validitet

3.1 Dokumentanalyse

Siden jeg valgte å analysere programmeringsoppgaver fra et læreverk, så var det et logisk valg å velge dokumentanalyse som metode for å nærmere undersøke dette emnet. Dokumentanalyse er kort forklart analyse av dokumenter. Det er mange måter å gjennomføre en dokumentanalyse på. En av grunnene til dette er at det er mange ulike former for dokumenter. Et dokument kan blant annet være: bilder, lydopptak, bøker, artikler, sosiale medier, nettsider og så videre. «Dokumenter er beretninger som ikke er generert av forskerens innsats, men er overlevert materiale fra en situasjon i fortiden» (Christoffersen & Johannessen, 2012). altså et dokument er et allerede eksisterende materiell som er skapt av noen andre enn den som skal gjennomføre analysen. Dokumentene jeg har sett på og valgte å analysere i denne analysen vil dermed bli oppgavene i Kikora (Christoffersen & Johannessen, 2012; Postholm et al, 2018).

I dokumentanalyse må man også ha en form for vurdering eller en måte å analysere dokumentene. Altså man må ha noe å se på dokumentene oppimot. Det kan være ulike områder eller ting som er interessante å se på. Hva som er interessante kommer an på hva målet og fokuset for analysen er. Forskjellige områder som kan være interessante å se på er for eksempel: forfatter, kontekst, formål, budskap og så videre. I denne analysen har jeg valgt å bruke rammeverkene om kognitive krav (Stein & Smith, 1998) og oppgavetype (Bråting & Kilhamn, 2022) som analyseverktøy for å undersøke oppgavene i Kikora.

3.1.1 Analyseobjekt

Analyseobjektet i denne analysen er to av temaene innen programmering i Kikora på 9.trinn. Det er flere grunner at jeg valgte å se på oppgaver i Kikora. En av grunnene va at fokuset lå på programmeringsoppgaver i matematikkfaget, og i Kikora så kan man finne flere oppgavesett eller temaer knyttet til Programmering. På 9. trinn er det totalt 7 ulike temaer fordelt på to undergrupper som går innunder programmering. Dette gjorde også at jeg kunne ta for meg to temaer med ulikt fokus knyttet til programmering. Fra blokk til tekstprogrammering er et tema som i stor grad setter fokus på selve programmeringen og det å lære seg det grunnleggende innen tekstprogrammering, mens i Programmere Pytagoras så er det naturlig et større fokus på å knytte det oppimot et matematisk tema.
En annen grunn til at jeg valgte å se på oppgaver fra Kikora, og som jeg har nevnt tidliggere er at Kikora er et læreverk jeg vet blir tatt i bruk i skolen, og det er et læreverk som jeg selv har vært borti tidligere. Det at Kikora er et læreverk som blir brukt i skolen i dag gjør det til et veldig relevant og aktuelt læreverk å analysere.

Kikora er også et digitalt læreverk som bringer med seg egne utfordringer og fordeler. For eksempel siden det er digitalt så kan man jobbe med det hvor som helst, det er en Platform med ferdiglagde oppgaver som retter seg selv, og den gir automatisk med en gang tilbakemelding til elevene som sier hvor vidt oppgaven er gjennomført riktig. Kikora påpeker også selv på sin framside (Kikora, U.å.) at de gir elevene Formative tilbakemeldinger og at Kikora selv både tolker og retter svarene elevene kommer med

3.1.2 Kikora oppbygning

Når du først går inn på Kikora så får du opp alle trinnene deretter når du har valgt hvilke trinn du skal inn på, så får du opp ei liste med emner. På 9.trinn er Programmering emne nummer 2 som videre er delt opp i to undergrupper. Det ene heter «2.1 Grunnleggende programmering» og det andre heter «2.2 Programmere geometri og sannsynlighet.» under hver av disse finner du oppgavesett eller temaer som Kikora selv kaller det. Som nevnt har jeg valgt to temaer som heter Fra blokk til tekstprogrammering og Programmere Pytagoras. (Kikora, U.å.)

Temaene i Kikora er satt opp på en måte der du går fra oppgave til oppgave helt til du er ferdig med det temaet. Det er som regel en oppgave eller et spørsmål på hver side, og når du er ferdig eller har fått riktig svar så får du opp en knapp, hvor det står «fortsett», som tar deg til neste

side. Hver side er nummerert, for eksempel 1,2 eller 4,3. Det første tallet i nummereringen altså tallet før kommaet er med på å gruppere oppgaver innad i temaet, dette er ganske likt med hvordan mange lærebøker gjør det for å skille mellom underkapitler. Kikora bruker grupperingen til å skille mellom små variasjoner innad i temaet, for eksempel hvis noe nytt introduseres eller hvis det er en tydelig endring i hvordan oppgaven blir gitt. Det andre tallet eller tallet etter komma er bare enkel nummerering, altså for eksempel 3,2 er oppgave 2 i gruppering 3.

Jeg må også påpeke noen endringer som har skjedd mens jeg har gjennomført denne analysen. Kikora har flyttet litt på noen temaer, så noen av de ligger nå under et annet trinn enn de gjorde før og noen ligger under et annet trinn i tillegg til at de ligger der de lå før. Dette inkluderer de temaene jeg har sett på. Dette har veldig liten innvirkning på analysen i seg selv, fordi temaene jeg har undersøkt ligger begge fortsatt under 9.trinn, men nå også under 10.trinn. oppgavene er også fortsatt de samme som før de ble flyttet. Så i praksis har ikke oppgavene jeg har sett på endret seg, men det er fortsatt verdt å nevne fordi det viser at det er et læreverk som stadig er i endring.

3.2 Analyseprosessen

3.2.1 Tematisk analyse

Tematisk analyse (Braun & Clarke, 2006) er en kvalitativ forskningsmetode hvor det er tatt i bruk en strukturert tilnærming for å utforske og finne meningsfulle mønstre fra datamateriale, slik som tekster, intervjuer eller observasjoner. Denne metoden er ofte brukt i samfunnsvitenskapelig forskning for å forstå komplekse fenomener, og den gir forskere muligheten til å organisere og analysere variert informasjon på en systematisk måte. Dette lar også forskere utforske dataene sine på en dyp og deskriptiv måte. Den åpner opp for å oppdage nye temaer og sammenhenger som kanskje ikke var åpenbare på forhånd. Denne metoden er også veldig fleksibel eller Tilpasningsdyktig. Altså er dette en metode som kan tilpasses forskningsspørsmål og datamateriale. Det er ikke en streng mal som må følges, noe som gir forskere rom for kreativitet. Tematisk analyse innebærer både objektive og subjektive elementer. Forskerens tolkning spiller en viktig rolle, men metoden krever også en synlighet i analytiske beslutninger. Altså gir denne formen for analyse en frihet til å endre på elementer

man mener er viktig, men man må som forsker vise til og forklare valg som er tatt underveis (Christoffersen & Johannessen, 2012; Postholm et al, 2018).

Tematisk analyse er en passende og nyttig metode for min oppgave om programmeringsoppgaver i Kikora. Den gir meg muligheten til å utforske og organisere komplekse data på en strukturert måte, samtidig som den kan tilpasses mine spesifikke forskningsspørsmål og behov. Den utforskende og fleksible naturen til tematisk analyse gjør den til en verdifull ressurs for å oppnå en dypere forståelse av kognitive krav (Stein & Smith, 1998) og oppgavetyper (Bråting & Kilhamn, 2022) i et digitalt læreverk som Kikora.

3.2.2 Kognitive krav

Som nevnt tidligere har jeg valgt å analysere oppgavene i lys av teorien om kognitive krav (Stein & Smith, 1998). Jeg kommer til å referere til de forskjellige kravene i teorien utfra hvilke nivå de tilsvarer. For eksempel så vil nivå 1 tilsvare det kognitive kravet Memorering.

Eksempeloppgaver nivå 0

I noen av oppgavene får du en ferdig kode hvor du bare skal trykke på kjør kode og se hva som skjer. Disse oppgavene faller utenfor alle nivåene i teorien om kognitive krav fordi de kun krever at eleven klikker seg videre til neste oppgave. Altså de stiller ikke noen krav til elevene som faller innunder teorien om kognitive krav (Stein & Smith, 1998). Derfor har jeg valgt å legge til et nivå 0 for eksempeloppgaver. Nivå 0 vil dekke alle oppgavene som ellers da ikke faller innunder de andre kognitive kravene.

Hvordan gjennomføres analysen kognitive krav

Jeg kommer til å ta for meg en og en oppgave for så å notere ned hvilke nivå oppgaven faller under i et Excel-dokument. I Excel-dokumentet kommer oppgavene til å bli referert til utfra hvilken rekkefølge de møtes i, og det vil markeres med x hvilke nivå de faller innunder. Som nevnt så vil jeg kun referere til kognitive kravene som nivåer det betyr at med nivå 0 så blir det sånn:

Nivå 0 = Eksempeloppgaver
Nivå 1 = Memorering
Nivå 2 = Prosedyrer uten sammenhenger
Nivå 3 = Prosedyrer med sammenhenger

Nivå 4 = Å gjøre matematikk

oppgavenr	nivå 0	nivå 1	nivå 2	nivå 3	nivå 4
1	x				
2			x		
3					x
4			x		

Figur 3-1 eksempeltabell for analysen av kognitive krav

3.2.3 Oppgavetype

Det andre analyseverktøyet jeg har valgt å bruke er det samme som er brukt i artikkelen om integrering av programmering i svensk skolematematikk. Altså jeg skal undersøke hva slags oppgavetyper (Bråting & Kilhamn, 2022) innen programmering som er brukt

Endringer i analyseverktøyet

Også i dette analyseverktøyet er det oppgaver som faller utenfor, og ikke passer inn i noen av kategoriene. Dette er et analyseverktøy for å se på hva slags type oppgaver som er brukt i forbindelse med programmering. Siden dette er et verktøy spesifikt for programmeringsoppgaver, så vil jeg ikke bruke de på oppgavene som bare er rene matteoppgaver. Det som skiller disse oppgavene fra de andre er at de ikke inneholder noen som helst form for tilknytning til programmering og fokuserer kun på matematisk innhold. Grunnen til at disse oppgavene i det hele tatt er med i den andre analysen er fordi Kikora selv har lagt de inn under emne programmering, og er en del av temaene.

Selv om oppgavene som ikke er programmeringsoppgaver er luket ut så er det fortsatt oppgaver som faller utenfor analyseverktøyet. Disse oppgavene kan beskrives som fakta spørsmål, men er tydelig rettet mot programmering og ikke matematikk. Jeg har derfor valgt å legge til enda en kategori, som da heter fakta spørsmål.

Tolkning av kategorier

I analyseverktøyet er det to kategorier som kan overlappe avhengig av definisjon. Det er kategoriene finne ut og forme og skape. Kategorien å finne ut handler om å finne prosedyren som må til for å løse et problem, men det vil innebære å skape en kode som da løser problemet. Forme og skape handler i hovedsak om å skape en egen kode som gjennomfører en ønsket handling. Dermed blir disse kategoriene veldig like og en oppgave som faller innunder den ene kategorien vil nesten helt sikkert dermed falle inn under den andre. Jeg har derfor valgt å se på disse kategoriene oppimot PRIMM (Sentance et al, 2019) for å skille mellom kategorien. For selv om det virker som kategoriene tar for seg det samme, så er de beskrevet med forskjellige ord. Finne ut er beskrevet på en måte som heller mer mot investegate, mens forme og skape heller mer mot modify og make. Altså så har jeg valgt å se på finne ut som en kategori hvor elevene selv må tolke og undersøke for å finne ut hvordan de skal løse oppgaven, men i forme og skape tar elevene mer eierskap til koden og lager en egen kode for å løse oppgaven de har fått.

Hvordan gjennomføres analysen oppgavetype

Analysen her vil i stor grad gjennomføres på like måte som analysen for kognitive krav, men i motsetning til den analysen må alle oppgavene først sorters så det kun er programmeringsoppgavene som står igjen. Jeg kommer også her til å legge alt inn i et Excel-dokument.

oppgavenr	følge	finne regel	feil søke	forme og ska	forklare	forestille seg	fakta spørsmål
1	x						
2			x				
3		x					
4							x

Figur 3-2 eksempeltabell for analysen av oppgavetype

3.2.4 Se på tvers av analyseverktøyene

Hvorfor sammenlikne

Etter å ha gjennomført analysene hver for seg så valgte jeg og se på tvers av analyseverktøyene. Grunnen til dette er at jeg oppdaget at jeg så underveis i analyseprosessen

at det var noen oppgavetyper (Bråting & Kilhamn, 2022) som jeg automatisk antok ville tilsvare høyere kognitive krav (Stein & Smith, 1998) som viste seg å gå under et av de lavere kognitive kravene. Jeg har derfor sett på hvilke kognitive krav som kreves av de forskjellige oppgavetypene.

Hvordan gjennomføre

Fordi dette baserer seg på begge analyseverktøyene så kommer også her bare oppgavene som er programmeringsoppgaver til å bli sett på. Også siden Ingen av de oppgavene i programmere Pytagoras som gikk under flere kategorier var programmeringsoppgaver, så vil alle oppgavene her kun telles en gang. Dette betyr at når alle oppgavene legges sammen vil man ikke ende opp på mer enn 100 prosent

Begge analysene var ført inn i ett Excel-dokument, dermed var det var det forholdsvis ukomplisert å se på de oppimot hverandre. Ved å bruke funksjoner i Excel så ble det sjekket hvilke oppgavetyper som gikk under hvilke krav forså å sjekke hvilke oppgavetyper de går under. For eksempel så sjekket Excel alle oppgavene om de gikk under det kognitive kravet memorering eller nivå 1 og om de gikk under oppgavetypen følge. Hvis en oppgave gikk under begge så ble den telt. når alle oppgavene var sjekket så hadde man summen over hvor mange oppgaver som gikk under nivå1 og følge oppgaver. Dette ble gjort for alle de mulige kombinasjonene av kognitive krav og oppgavetyper.

3.2.5 Affordances og constraints

For å opprettholde en systematisk og ryddig tilnærming til analysen, ble gjennomgangen av affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014) gjennomført som en sekundær fase av forskningen. Dette ble gjort etter de tidligere analysene. For å identifisere affordances og constraints, ble først en innledende forståelse av oppgavene og deres innhold etablert. Dette ble oppnådd gjennom arbeidet med de tidligere utførte analysene. Deretter ble de mest åpenbare og tydelige affordancene og constraints notert, i tillegg til de som ble ansett som særlig interessante.

Målet her var ikke å finne alle affordancene og Constraints som er i Kikora, men derimot kun de mest tydelige og relevante i syn av de tidligere analysene. Derfor er kun disse affordances og constraints identifisert og dokumentert. Selv om denne tilnærmingen av analyse kan gjøre at det er affordances og constraints som ikke blir sett på, så var målet her å gi en dypere innsikt

i hvordan oppgavene er, inkludert grunner til hvorfor og hvordan Kikoras form som digitalt læreverk kan påvirke oppgavene.

3.3 Relabilitet og Validiteten

I resten av dette kapitlet kommer fokuset til å ligge på kvaliteten av analysen. jeg kommer til å ta for meg hva relabilitet og validitet er, for deretter å skrive noe om hvordan dette går oppimot analysene jeg har gjennomført.

3.3.1 Reliabilitet

Hva er Reliabilitet

Reliabilitet kommer fra det engelske ordet reliability som betyr pålitelighet på norsk. Reliabilitet handler om hvor vidt analysen eller undersøkelsen er gjennomført på en måte som gir pålitelige data. En enkel måte å forstå Reliabilitet på er å stille spørsmålet: Er dataene som har kommet fram i analysen riktige? Reliabilitet går i størst grad på å se etter grunner eller muligheter for at noe kan ha påvirket dataene og at de derfor ser annerledes ut enn det som stemmer med realiteten. Derfor er det viktig at forskeren forklarer godt hvordan den har gjennomført analysen og hvilke valg den har tatt, samt om det har skjedd noe under analysen som kan ha påvirket dataene (Christoffersen & Johannessen, 2012; Postholm et al, 2018). Et eksempel på dette kan være som jeg nevnte at det var noen endringer i Kikora etter at jeg hadde påbegynt analysen. For når jeg begynte på analysen lå Programmere Pytagoras under 10.trinn og Fra blokk til tekstprogrammering under 9. trinn i Kikora, men nå ligger de under både 9.trinn og 10. trinn. Hadde Kikora endret på temaene så kunne dette ha hatt en stor påvirkning på analysen, men siden alle oppgavene var like og det var verken fjernet eller lagt til oppgaver så hadde det ingen påvirkning på analysen. Derimot så kan man argumentere at resultatene nå sier mer om innholdet generelt innen programmeringsoppgavene til Kikora for både på 9. og 10.trinn siden alle oppgavene finnes på begge trinn.

Hva i oppgaven kan påvirke reliabiliteten

Det som mest sannsynlig kan ha hatt størst påvirkning på analysene jeg har gjennomført er meg. Som forsker velger man hvordan analysen skal gjennomføres og hva som skal ses på. Både i analysen for kognitive krav (Stein & Smith, 1998) og analysen for oppgavetyper

(Bråting & Kilhamn, 2022) så har jeg tatt i bruk analyseverktøy som allerede eksisterer og tilpasset de for at de skalle passe til det jeg har valgt å analysere. Dette er på grunn av hvordan jeg har valgt og tolke og definere de forskjellige kategoriene i de to analyseverktøyene. Hvis jeg hadde valgt å definere kategoriene annerledes så hadde resultatet mest sannsynlig sett annerledes ut.

Når analyseverktøyene blir sett oppimot hverandre så er det ingen ting nytt som kan påvirke relabiliteten. Dette er fordi det ikke hentes inn noen nye data. Her blir det kun brukt data som er funnet gjennom de andre analysene. på en måte kan man se på det som en omstrukturering av data for å få fram andre funn.

Når det kommer til affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014) så lener dette seg sterkt på mine observasjoner. Altså det jeg har funnet i affordances og constraints vil kun være av det jeg har sett og observert. Dette betyr at det kan være ting jeg ikke har sett eller lagt merke til som andre ville kunne funnet. Med andre ord hvis en annen person hadde gjennomført analysen så kunne resultatet sett helt annerledes ut enn det som er presentert i denne oppgaven. Det kan dermed argumenteres at det faktum at jeg bare er en person er en svakhet i denne analysen fordi det gjør at det er mulig noe blir oversett som en annen person hadde sett.

Hvordan har jeg prøvd å øke relabiliteten

Gjennom analysene har jeg systematisk tatt for meg en og en oppgave og for hver oppgave notert ned i ett Excel-dokument hvilke kategorier de faller innunder. Jeg har også konsekvent sett på oppgavene oppimot definisjonene for de forskjellige kategoriene for at jeg skal analysere og plassere oppgavene i de riktige kategoriene. Jeg har også gjort valg som at i analysen om kognitive krav (Stein & Smith, 1998) så kan oppgaver gå innunder flere kategorier, dette er for at oppgavene skal bli passert mest mulig riktig.

En metode som brukes for å øke reliabiliteten til en analyse er ved at analysen gjennomføres på nytt. Dette kan ses på som en test for å se om man får de samme resultatene flere ganger. Dette kan også senke reliabiliteten til en analyse hvis det viser seg at man får forskjellige resultater. Den beste måten å gjennomføre dette på er ved at det gjøres av en annen person enn

den som originalt har gjennomført analysen. (Christoffersen & Johannessen, 2012; Postholm et al, 2018) I denne analysen er ikke det mulig å få til, men i likhet med denne metoden i form av en kvalitets test har alle oppgavene blitt gjennomgått tre ganger. Dette gjøres som sagt for å prøve og få mest riktige resultater og muligens finne feil som kan oppstå første gang oppgavene ble gått igjennom. Alt dette endrer fortsatt ikke det at analysen kun er gjennomført av en person, og dette kan som sagt ses på som en svakhet.

,

3.3.2 Validitet

Hva er validitet

Validitet er i likhet med Reliabilitet et ord som er hentet fra et lignende ord på engelsk. Validitet kommer fra det engelske ordet validity og betyr gyldighet. I motsetning til reliabilitet som ser på hvor pålitelige resultatene er så ses det her på hvor mye sier resultatene om det man ønsker å undersøke. Altså så er spørsmålet her gir disse resultatene en riktig oppfatning av det vi lurer på.

For eksempel la oss si at man lurer man på hvor mange personer som krysser grensen mellom Norge og Sverige hver dag. En ide kan være å se på hvor mange kjøretøy som krysser grensen hver dag. Man vil kunne få resultater som er ganske pålitelig, men hvor gyldig blir resultatene i forhold til det man lurer på? Det man lurte på var hvor mange personer som krysser grensen. Ikke hvor mange kjøretøy, for i et kjøretøy kan det sitte flere personer. Man kan derfor si at resultatet selv om det kan ha høy reliabilitet så ha det ikke høy validitet fordi det ikke svarer på det vi lurte på i utgangspunktet.

Validitet deles ofte opp i to deler, Indre validitet og ytre validitet. Disse begrepene ser på to ulike områder av validiteten som hver spiller en vesentlig rolle i å evaluere gyldigheten til en forskningsstudiet. Disse to typene validitet er fundamentale for å vurdere hvorvidt studiens funn er riktige og om de kan brukes til å forstå eller som et argument på hvorfor noe er som det er i andre liknende sammenhenger.

Indre validitet

Indre validitet refererer til graden av nøyaktighet i å trekke konklusjoner om årsakssammenhenger i en studie. Med andre ord, så dreier det seg om hvor godt studien måler det den faktisk ønsker å måle, uten at resultatene blir forvrengt av andre faktorer. Det er altså viktig å fastslå at det er en direkte sammenheng mellom det man faktisk har sett på og at det ikke er noe annet som har hatt en påvirkning. For eksempel hvis man ønsker å se på sammenhengen mellom elevers sosioøkonomiske bakgrunn og hvilke karakterer de får i matematikk, så må det undersøkes på en måte som gjør at man vet at det ikke er andre faktorer som er grunnen til at elevene får de karakterene de gjør. Indre validitet sikrer at det vi har funnet sier noe om det vi har valgt å forske på og ikke er på grunn av andre uidentifiserte faktorer.

For å oppnå høy indre validitet i en studie, må forskeren kontrollere og minske påvirkningen av andre mulige variabler. Dette kan inkludere kontrollgrupper, randomisering og eksperimentelt design. Ved å begrense påvirkningen av disse variablene, kan forskeren være mer sikker på at resultatene representerer den faktiske sammenhengen mellom variablene som studeres.

Indre validitet i denne oppgaven

I denne oppgaven er det egentlig vanskelig å si noe om indre validitet fordi, som nevnt over så ser indre validitet på hvor vidt man har klart å tydelig undersøke sammenhengen mellom to variabler. Altså bevise at variabel A påvirker variabel B, og forklare hvordan man har hindret andre Variabler fra å kunne påvirke Variabel B. Metoden som er brukt er som sagt dokumentanalyse, og målet er å undersøke hvordan oppgavene i Kikora er, ikke å finne ut hvorfor de er som de er. Dermed er det lite man kan si om den indre validiteten. Dermed kan man påstå at denne oppgaven har en veldig god indre validitet fordi det er ikke mulig at noen faktorer kan påvirke resultatet, men det blir egentlig feil fordi indre validitet ikke er relevant sett i sammenheng med metoden og hva som undersøkes

Ytre validitet

Ytre validitet handler om i hvilken grad studiens funn kan generaliseres til andre populasjoner, situasjoner eller kontekster. Det er viktig å vurdere om resultatene fra studien er representative for andre sammenhenger enn den som er til stede i selve studien. Med andre ord, handler det

om å avgjøre om studiens funn er gyldige utenfor den spesifikke gruppen eller situasjonen som ble studert.

For å oppnå høy ytre validitet, er det nødvendig å velge et representativt utvalg og i noen sammenhenger forsøke å gjenskape situasjoner som er så lik de virkelige forholdene som mulig. Det vil variere ettersom hva man studerer og hvilken metode man har brukt. For eksempel i en spørreundersøkelse så kan dette kunne omfatte å sørge for at man har et utvalg av deltakere som er varierte i alder, kjønn, geografi eller andre relevante faktorer. Ved å gjøre dette kan forskeren være mer sikker på at funnene fra studien kan generaliseres til ulike populasjoner og settinger.

Det er også verdt å nevne at det å øke den indre validiteten kan komme på bekostning av ytre validitet, og motsatt. For eksempel kan en svært kontrollert studie med begrenset deltakergruppe ha høy indre validitet, men den kan ha begrenset relevans for å forstå virkeligheten utenfor og dermed få en redusert ytre validitet.

Ytre validitet i denne oppgaven

I motsetning til indre validitet der det ikke er mulighet for at det er noe som har en påvirkning på validiteten, så er det her en del ting man må se på når det kommer til ytre validitet.

For å få en bedre og videre forståelse av programmeringsoppgavene i Kikora, ble det som nevnt tidligere valgt to ulike temaer, Fra blokk til tekstprogrammering og Programmere Pytagoras. Denne tilnærmingen ble valgt med tanke på å gi bedre innsikt i variasjonen som er i oppgavene i Kikora, da det første temaet introduserer tekstprogrammering, mens det andre fokuserer på anvendelse av Pytagoras i programmering. Dette bidrar til å gi en bredere representasjon av oppgavetyper (Bråting & Kilhamn, 2022) og kognitive krav (Stein & Smith, 1998) i Kikora enn hvis to temaer med lignende innhold hadde blitt valgt.

Det bør også bemerkes at analysen har en begrenset rekkevidde når det gjelder utvalg av oppgaver i Kikora. Av programmeringsoppgavene på 9. trinn, ble kun to av syv tilgjengelige temaer som faller under programmering utforsket. Dette utgjør en relativt liten andel av alle programmeringsoppgavene i Kikora, spesielt når man tar i betraktning at det også finnes programmeringsoppgaver på andre trinn. Som et resultat av dette begrensede utvalget er det utfordrende å generalisere funnene til alle programmeringsoppgaver i Kikora. Når det gjelder programmeringsoppgaver i andre digitale læreverk, så vil det sannsynligvis være enda mindre

overføringsverdi fordi det kun er oppgaver fra Kikora, analyseverktøyene som er brukt her har også til dels blitt tilpasset oppgavene i Kikora. Som igjen kan gi resultatene enda mindre overføringsverdi

Det er også relevant å merke seg at analysen i stor grad fokuserer på en veldig liten del av oppgavene i Kikora totalt sett. For temaene under programmering har kun en del B etter introduksjonen. Mange andre temaer i Kikora inneholder også del A og C, som betyr at det er mulig oppgavene som ble sett på her bare representerer en tredjedel av det to temaer innen et annet område enn programmering ville gjort. Derfor vil generaliseringen av funnene i Kikora som helhet ha en veldig begrenset overføringsverdi til andre områder i Kikora.

Unntaket fra denne generelle begrensningen er imidlertid når det kommer til affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014). Disse aspektene ble sett på med tanke på Kikora som et digitalt læreverk og har derfor en mer direkte relevans. Likevel er det viktig å merke seg at selv disse aspektene er knyttet spesifikt til oppgaver fra Kikora og hvordan Kikora er bygget opp deres overføringsverdi til andre digitale læreverk kan være begrenset.

Samlet sett har denne analysen en begrenset ytre validitet, da den kun går spesifikt på oppgaver i Kikora og har begrenset dermed generaliseringsverdi til andre læreverk og kontekster. Dette burde tas i betraktning om man skulle vurdere funnene fra denne analysen i en annen sammenheng.

4. Resultater

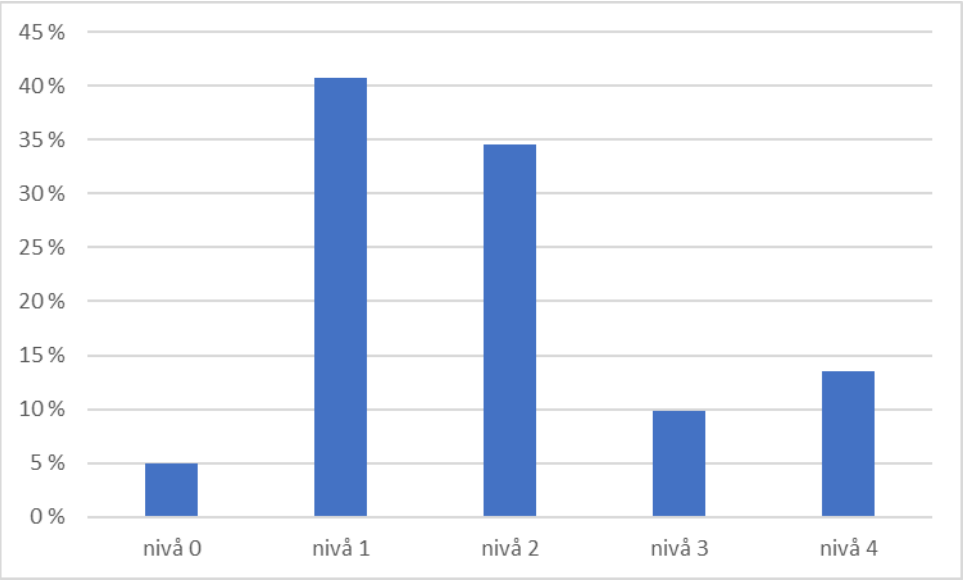
I dette kapitelet presenterer jeg funnene som har kommet frem gjennom resultatene av analysene av oppgavene i Kikora. Funnene vil bli systematisk presentert gjennom at først presenteres funnene knyttet til analysen av kognitive krav (Stein & Smith, 1998). Deretter presenteres funnene av analysen av oppgavetype (Bråting & Kilhamn, 2022). Etter dette vises funn og resultater for analysene sett oppimot hverandre slik som forklart i metodedelen. Til slutt i kapitlet blir funn av affordances og constrains presentert.

4.1 Kognitive krav

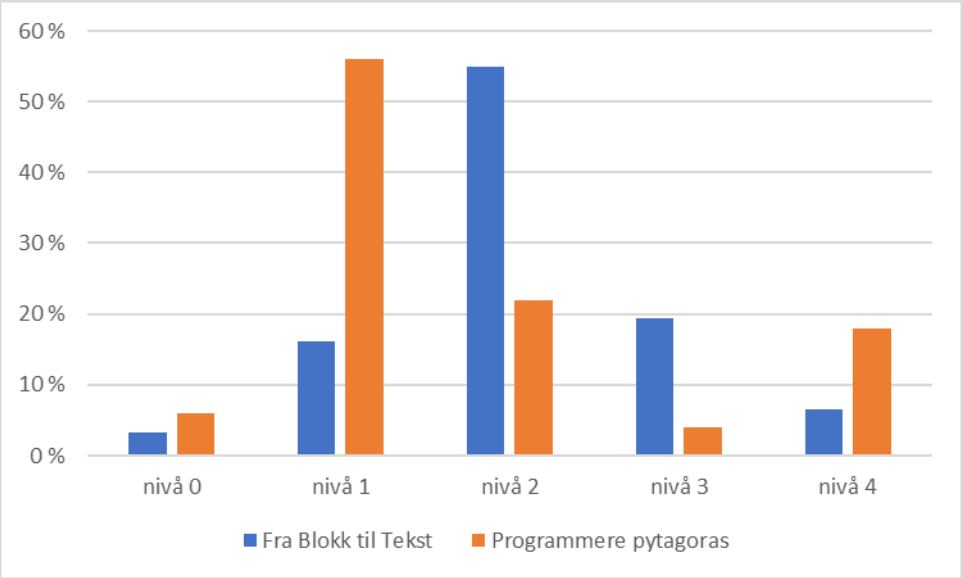
Resultatet av analysen viser at det er varierende hvilke kognitive krav (Stein & Smith, 1998) som stilles av elevene i de to temaene jeg har sett på. Figur 4-1 er et av to søylediagram som viser hvor stor prosentandel av oppgavene som går innunder de forskjellige kognitive kravene. Dette søylediagrammet viser totalt hvor stor andel av oppgavene som går innunder hver av de kognitive kravene i begge temaene samlet. Figur 4-2 er det andre søylediagrammet og viser det samme, men her er temaene delt så man kan se hvor stor prosentandel de kognitive kravene utgjør innen hvert av de to temaene. Som nevnt tidligere så referer nivåene til de forskjellige kognitive kravene.

- Nivå 0 = Eksempeloppgaver
- Nivå 1 = Memorering
- Nivå 2 = Prosedyrer uten sammenhenger
- Nivå 3 = Prosedyrer med sammenhenger
- Nivå 4 = Å gjøre matematikk

Fordi det er oppgaver i Programmere Pytagoras som faller under flere av de kognitive kravene så vil ikke prosentandelene til sammen bli hundre prosent. Det er kun noen 3 totalt oppgaver som faller under flere av de kognitive kravene, men på grunn av dette viser søylediagrammene bare hvor stor prosentandel de forskjellige kognitive kravene har av det totale antallet oppgaver. med andre ord hvis man legger sammen alle prosentandelene vil man få mer enn 100 prosent fordi de 3 oppgavene vil bli telt 2 ganger.



Figur 4-1 søylediagram som viser fordeling av kognitive krav samlet



Figur 4-2 søylediagram som viser fordeling av kognitive krav i hvert tema

Noen ting man kan se ut fra søylediagrammene er at det tydelig er flest av oppgaver under nivå 1 og nivå 2 altså memorering og prosedyrer uten sammenhenger. Vil også påpeke at det er kun noen få oppgaver som faller under nivå 0 eller eksempeloppgaver altså de oppgavene som ikke stiller noen krav til elevene. Ellers er antallet oppgaver som faller under de andre nivåene også ganske varierende, men i mindre grad. det er også tydelig at begge temaene er ganske ulike i forhold til de kognitive kravene. Det er også verdt å påpeke igjen at i temaet Fra blokk til tekstprogrammering så er det bare 31 oppgaver, men i å Programmere Pytagoras så er det 50 oppgaver. Dette vil som sagt bety at når temaene sammenlignes så vil hver oppgave

i Fra blokk til tekstprogrammering utgjør en større prosentandel, men når temaene legges sammen for å se på det totale antall oppgaver som går under hvert kognitive krav, så vil oppgavene fra Programmere Pytagoras påvirke resultatet mer fordi det er flere oppgaver.

Jeg har også valgt å lage en tabell som viser hvor stor prosentandel hvert nivå har totalt, altså jeg har lagt sammen de to temaene og sett på det som en helhet

	fra Blokk til Tekst	prosent Blokk	ptogrammere pytagoras	prosent pytagoras	totale nr	totale pro
totalt antall	31	100,00 %	50	100,00 %	81	100,00 %
nivå 0	1	3,23 %	3	6 %	4	4,94 %
nivå 1	5	16,13 %	28	56 %	33	40,74 %
nivå 2	17	54,84 %	11	22 %	28	34,57 %
nivå 3	6	19,35 %	2	4 %	8	9,88 %
nivå 4	2	6,45 %	9	18 %	11	13,58 %

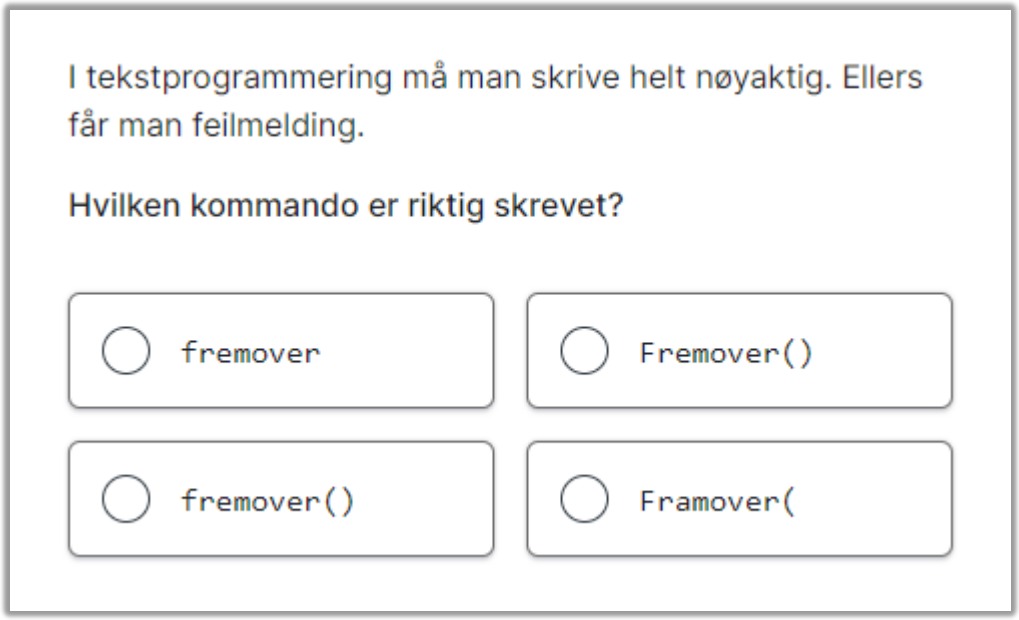
Figur 4-3 tabell som viser prosent og antall kognitive krav utgjør

4.1.1 Nivå 1 memorering

Det totale antall oppgaver som går under nivå 1 er 33 oppgaver av 81. Dette utgjør litt over 40 prosent av alle oppgavene. Det er tydelig en veldig stor variasjon mellom temaene i nivå 1. I Fra blokk til tekstprogrammering utgjør nivå 1 litt mer enn 16 prosent av oppgavene, mens i Programmere Pytagoras så er 56 prosent av oppgavene nivå 1 oppgaver. Dette utgjør en forskjell på nesten 40 prosent mellom temaene. Dette er den største forskjellen i kognitive krav mellom temaene.

Eksempeloppgave nivå1

Oppgaven her er hentet fra temaet Fra blokk til tekstprogrammering, og er oppgave 2.1 i intro delen. Denne oppgaven er en flervalgsoppgave som spør etter hvilken av de fire alternativene som har skrevet kommandoen «fremover» på den riktige måten. Dette er et tydelig eksempel på nivå1 eller memorering ved at det kun handler om det å huske hvordan man skal skrive en kommando.



Figur 4-4 oppgave 2.1 i intro fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

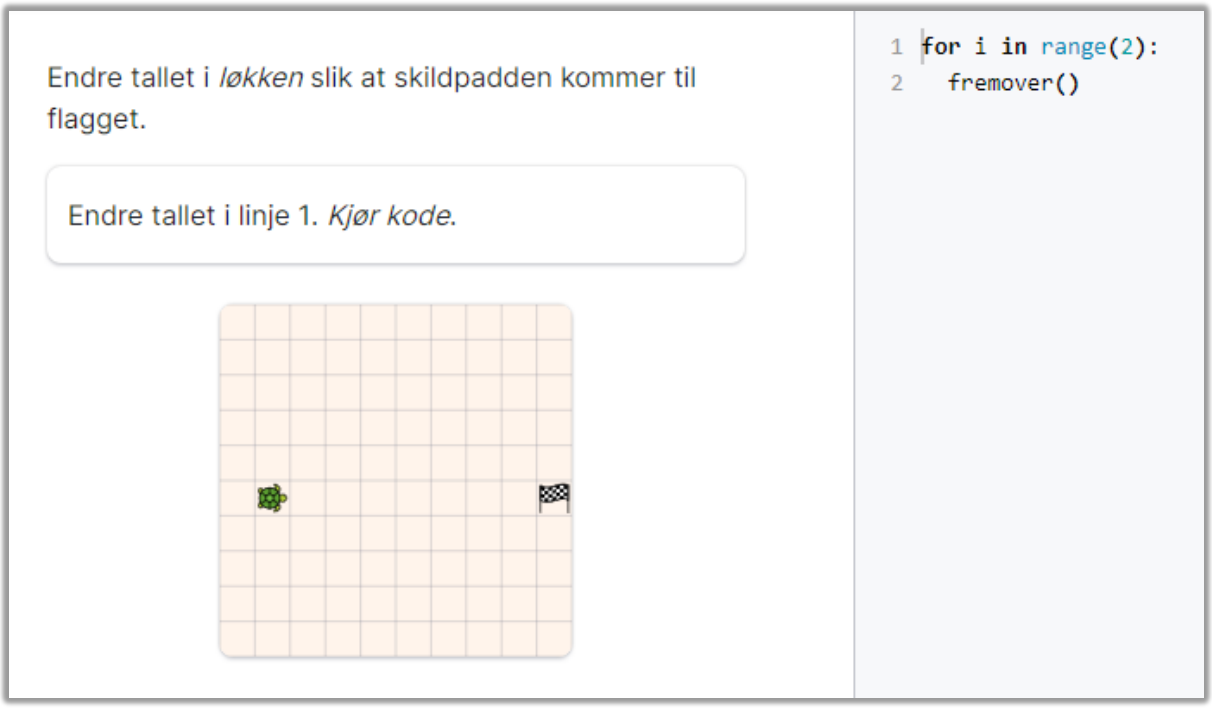
4.1.2 Nivå 2 prosedyrer uten sammenheng

Nivå 2 har i likhet med nivå 1 stor variasjon mellom temaene og dette kan man tydelig se på det andre søylediagrammet (figur 4-2). man kan blant annet se at nivå 2 har den høyeste søylen innen temaet Fra blokk til tekstprogrammering. Det er litt mer enn 34 prosent av alle oppgavene som faller innunder nivå 2. Dette tilsvarer 28 av de 81 oppgavene. Som sagt er det også her veldig stor forskjell mellom temaene, men i motsetning til nivå 1 så har Fra blokk til tekstprogrammering størst prosentandel. Nivå 2 utgjør nesten 55 prosent av oppgavene i Fra blokk til tekstprogrammering, men kun 22 prosent i Programmere Pytagoras. Det er en forskjell i prosentandel på litt under 33 prosent.

eksempeloppgave nivå2

Denne oppgaven er hentet fra temaet Fra blokk til tekstprogrammering, og er oppgave 4.3 i sti-B. Her har eleven fått i oppgave å få skilpadden til å gå til målflagget. Eleven må endre et tall i ei løkke for å få skilpadden til å bevege seg riktig antall ruter til høyre, så den lander på flagget. Det er tydelig gitt hva og hvordan eleven skal gjøre oppgaven, altså har eleven fått en

prosedyre som skal følges. Det skapes heller ikke noen matematiske sammenhenger til andre fagområder eller emner i matematikk. På grunn av dette er denne oppgaven en nivå 2 oppgave.



Figur 4-5 oppgave 4.3 i sti-b fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

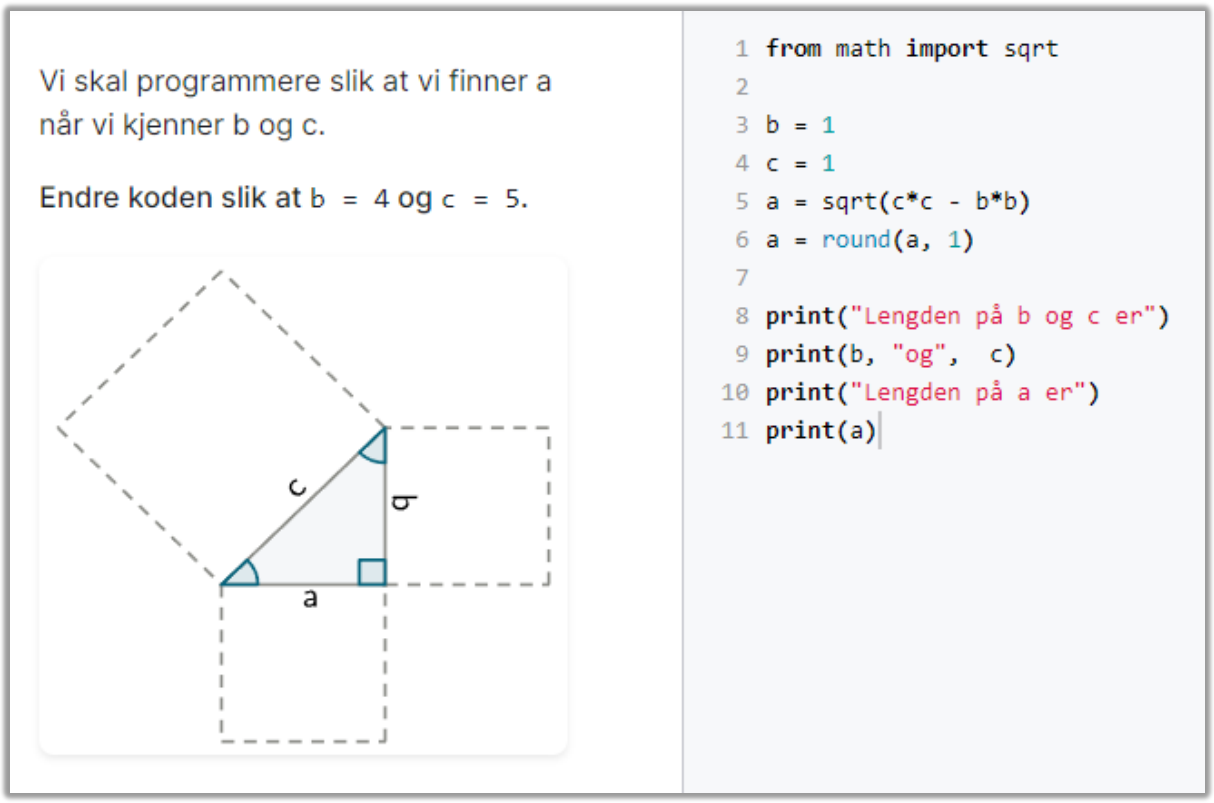
4.1.3 Nivå 3 prosedyrer med sammenheng

Nivå 3 utgjør totalt ca. 10 prosent av oppgavene i begge tamene som utgjør 8 oppgaver totalt. Det er også her litt forskjell mellom temaene i prosentandel oppgaver som faller innunder nivå 3. i Fra blokk til programmering så utgjør nivå 3 litt mindre enn 19,4 prosent, mens i Programmere Pytagoras så utgjør det kun 4 prosent. Dette er en forskjell på litt mer enn 15 prosent. det kan påpekes at det i prosentandel er nesten fire ganger så mange oppgaver i Fra blokk til tekstprogrammering som det er i Programmere Pytagoras, men det er også her i likhet med nivå 0 veldig få oppgaver som gjør at det kan framstå som det er enn større forskjell enn det egentlig er, selv om det totalt er snakk om åtte oppgaver fordelt på to temaer.

eksempeloppgave nivå3

Denne oppgaven er hentet fra temaet Fra Programmere Pytagoras, og er oppgave 5.2 i sti-B. denne oppgaven er et veldig tydelig eksempel på prosedyrer med sammenhenger. Oppgaven går ut på at de skal finne den siste siden i en vinkelrett trekant ved hjelp av koding. Her får

eleven en nesten ferdig kode, hvor de selv må plotte inn lengdene til de to sidene de har fått oppgitt. Det er tydelig hva og hvordan de skal gjøre det, altså har de en prosedyre, samtidig som det tydelig kommer fram en sammenheng om hvordan man kan bruke programmering og koding til å løse matematiske problemer eller oppgaver innen temaet Pytagoras.

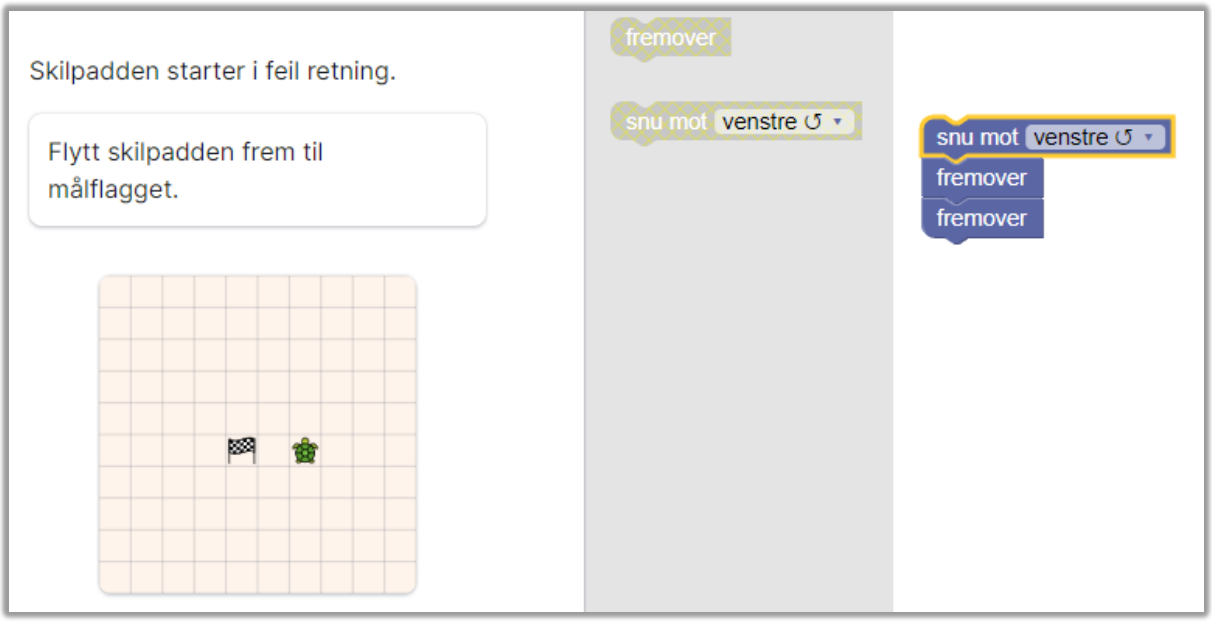


Figur 4-6 oppgave 5.2 i sti-b Programmere Pytagoras (https://feide.kikora.no/)

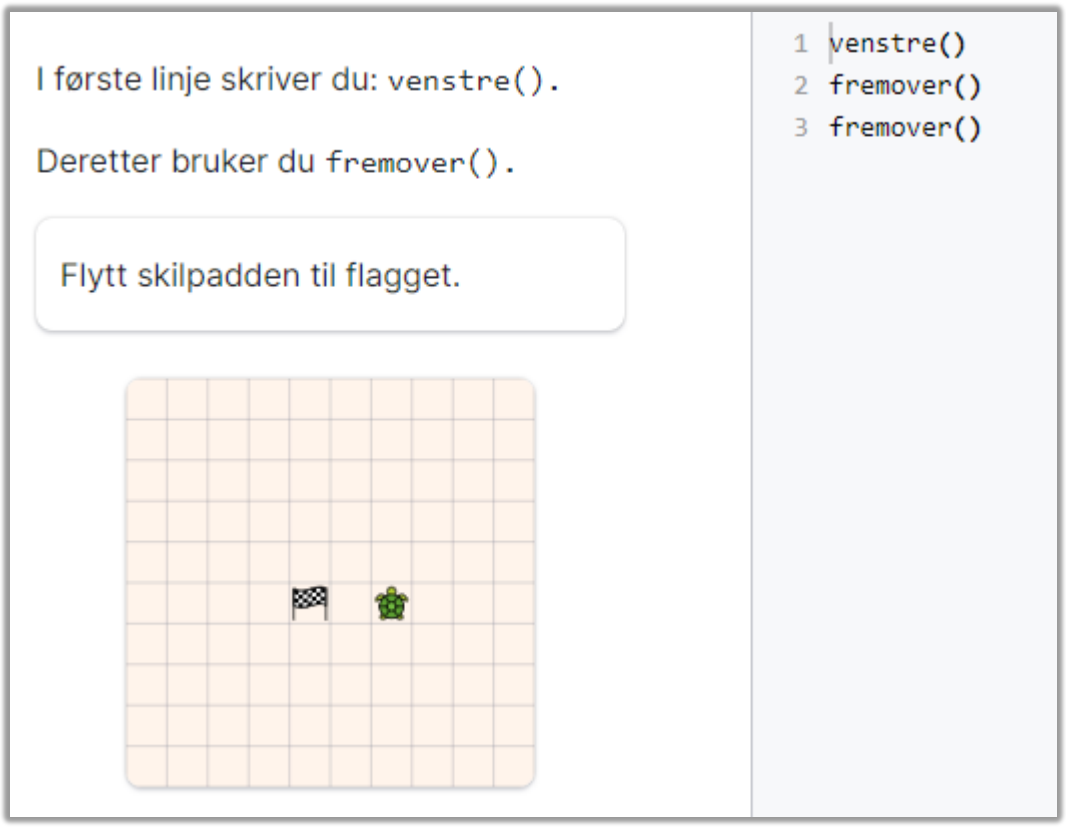
4.1.4 Nivå 2 og 3 finn på bedre navn senere

I analysen var det flere ganger at det var oppgaver som tydelig hørte sammen, og som påvirket hvor vidt en oppgave falt under nivå 2 eller nivå 3. Disse oppgavene kom spesielt tydelig fram under temaet Fra blokk til tekstprogrammering. Et eksempel på dette er oppgave 3.1 og 3.2 fra introdelen i Fra blokk til tekstprogrammering. Her kommer det først komme en oppgave hvor man skulle flytte en skilpadde med blokkprogrammering (se figur 4-9). Etter denne oppgaven ville det deretter komme en ny oppgave som var helt lik, men man skulle bruke tekstprogrammering istedenfor blokkprogrammering (se figur 4-10). Det er tydelig at disse oppgavene henger sammen og viser nettopp sammenhengen mellom blokk og tekstprogrammering som er navnet på temaet, men hver for seg ville begge oppgavene gått under nivå 2 fordi de i seg selv ikke viser eller forklarer noen sammenhenger til andre

matematiske definisjoner eller konsepter. Sammen derimot kommer sammenhengen tydelig fram og vil gå under nivå 3. fordi man ikke kan se sammenhengen før man har vært igjennom begge oppgave så vil den første oppgaven være nivå 2 og den andre oppgaven være nivå 3. Dette er fordi første gangen eleven vil ha mulighet til å kunne se sammenhengen er i den andre oppgaven og dermed er det den som skaper en sammenheng mellom de to forskjellige formene for programmering (Stein & Smith, 1998).



Figur 4-7 oppgave 3.1 i intro fra Blokk til Tekstprogrammering (https://feide.kikora.no/)



Figur 4-8 oppgave 3.2 i intro fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

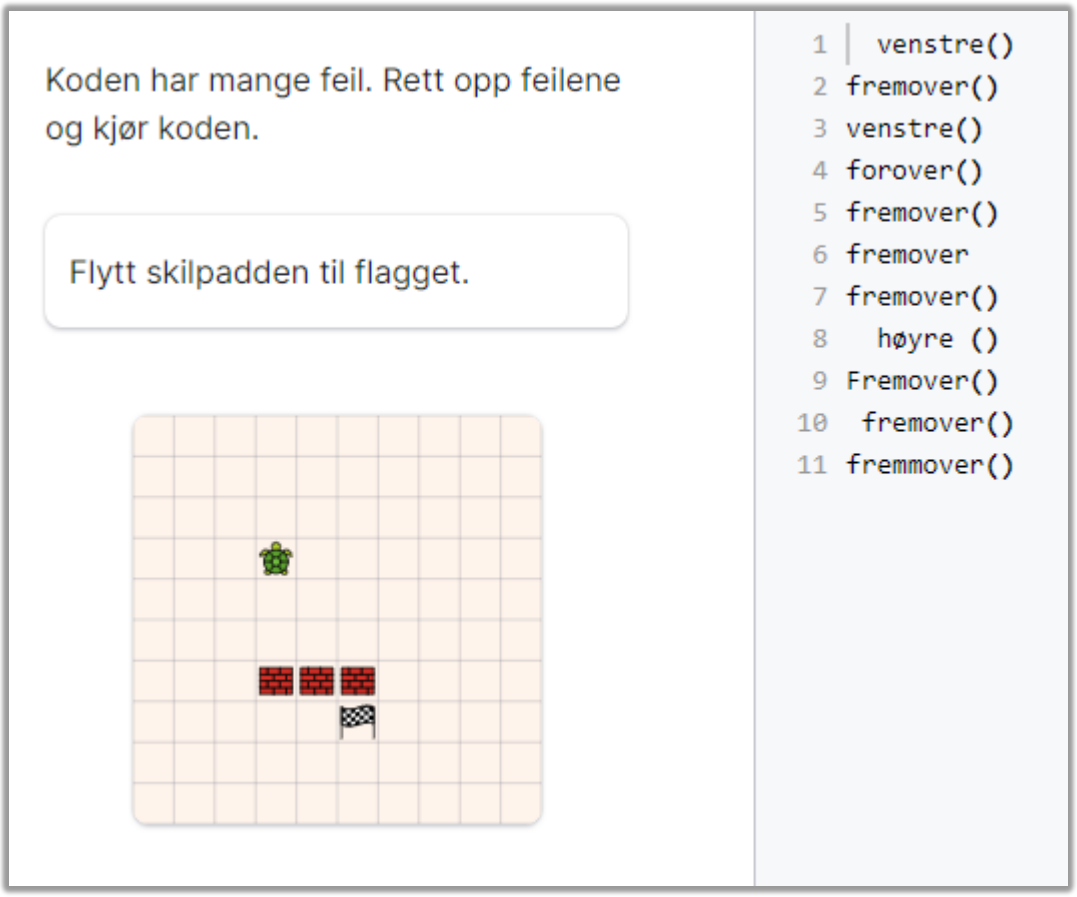
4.1.5 Nivå 4 gjøre matematikk

Det siste nivået ligger ikke langt unna nivå 3 når det kommer til prosentandelen av det totale antall oppgaver. oppgavene i nivå 4 utgjør ca. 13,6 prosent som utgjør 11 oppgaver totalt av 81. Det er også ikke så langt unna nivå 3 når det kommer til variasjon mellom temaene prosentvis, men i motsetning til nivå 3 så er det Programmere Pytagoras som har størst andel oppgaver her. Av Fra blokk til tekstprogrammering er litt mindre en 6,5 prosent av oppgavene som er nivå 4 mens i Programmere Pytagoras så er det 18 prosent av oppgavene som er nivå 4. Dette er en forskjell på litt mer enn 11,5 prosent.

Eksempeloppgave nivå4

Oppgaven er hentet fra temaet Fra blokk til tekstprogrammering, og er oppgave 2.2 i sti-B. denne oppgaven er et eksempel på å gjøre matematikk. Her er ikke prosedyren i fokus, men det at eleven klarer å lese koden. Oppgaven går ut på at elevene må finne og endre feil i koden

får å få skilpadden til å gå til målflagget, Altså de får en kode med flere feil som elevene selv må undersøke og jobbe med får å klare å finne og endre feilen.



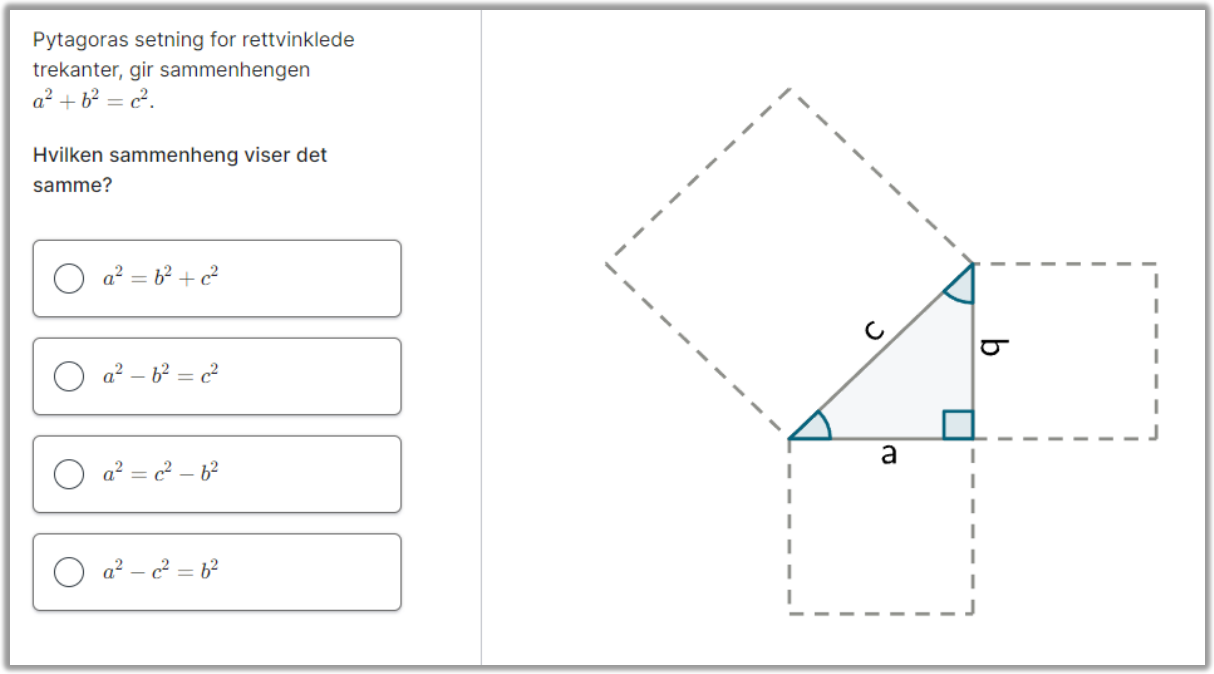
.

Figur 4-9 oppgave 2.2 i sti-B fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

4.1.6 Oppgaver som kan kreve forskjellige kognitive krav

Noen oppgaver kan variere i kognitive krav avhengig av elevenes tilnærming og som nevnt så faller noen av oppgavene innunder både nivå 1 og nivå 4. en av disse oppgavene er oppgave nummer 32 i analysen av Programmere Pytagoras eller oppgave 5,1 fra sti B i Kikora sin inndeling (se figur 4-11). Denne oppgaven er en flervalgsoppgave som spør om hvilke alternativ som viser det samme som «$a^2 + b^2 = c^2$». det riktige svaret er «$a^2 = c^2 - b^2$». Denne oppgaven kan være både nivå 1 og nivå 4 avhengig av elevenes tilnærming. Dette er fordi at for en elev så må man kanskje gå gjennom prosessen ved å trekke fra «b^2» fra begge

sider av likhetstegnet for at oppgaven skal gi mening og man skal klare å komme seg fram til riktig svar. For en annen elev kan det det være helt annerledes. Hvis en elev husker at man kan gjøre det samme ved at man bare kan flytte «b^2» over likhetstegnet og bytte fortegn for å få den ene likningen til å bli den andre, så vil ikke den eleven oppleve at oppgaven krever så mye som nivå 4 innen de kognitive kravene. For den andre eleven vil det være memorering altså nivå 1 fordi det krever ikke mer enn at man husker den ene regelen. Denne sammenhengen eller dynamikken som oppstår i slike oppgaver, der elevenes egne tilnærminger og forståelse spiller en rolle, skaper et scenario der oppgaven vil gå innunder nivå 1 og nivå 4. Dette gjør at noen oppgaver, inkludert den nevnte oppgaven og lignende, kan klassifiseres innen begge av de kognitive nivåene.



Figur 4-10 oppgave 5.1 i sti-b Programmere Pytagoras (https://feide.kikora.no/)

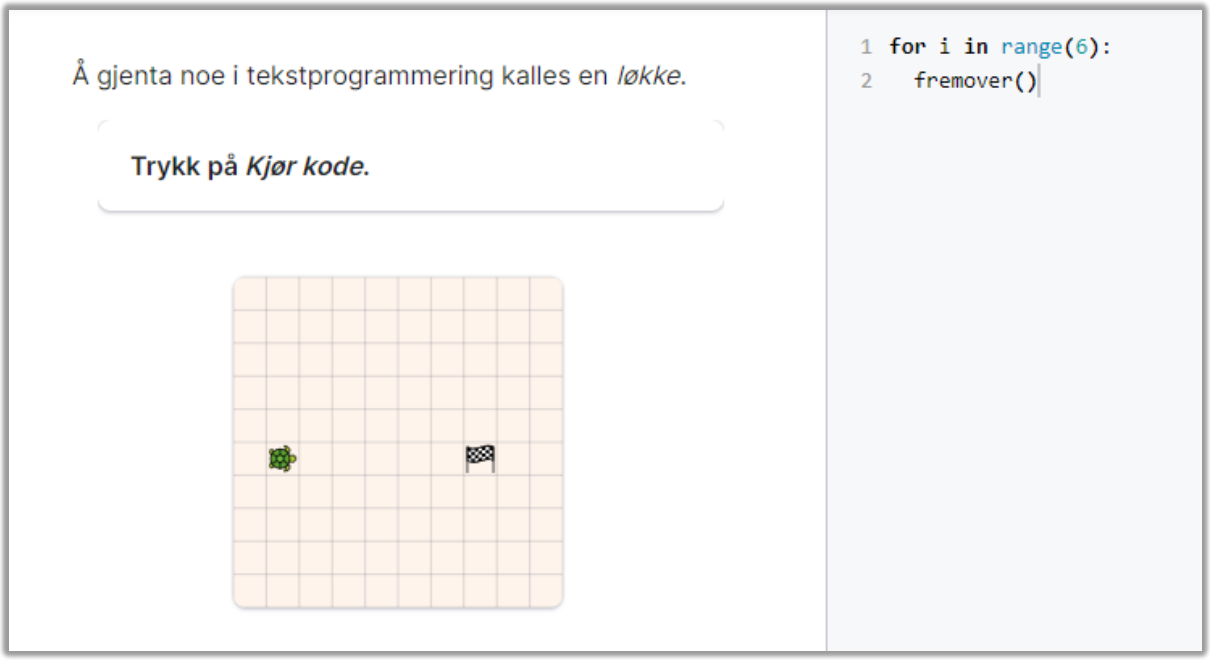
4.1.7 Nivå 0 eksempeloppgaver

Det var svært få oppgaver som gikk under nivå 0, kun ca. 5 prosent av alle oppgavene. med tanke på at dette er eksempeloppgaver så er ikke det veldig overaskende. Hvis man sammenligner de to forskjellige temaene så har temaet Fra blokk til tekstprogrammering prosentvis litt Større enn halvparten av prosentandelen Programmere Pytagoras har, men jeg vil gjenta at det kan være litt misvisende fordi det er kun en oppgave i Fra blokk til

tekstprogrammering og 3 oppgaver i Programmere Pytagoras, men dette er som sagt fordi det er flere oppgaver Programmere Pytagoras.

eksempeloppgave nivå 0

Denne oppgaven er hentet fra temaet Fra blokk til tekstprogrammering, og er oppgave 4.2 i sti-B. denne oppgaven er et godt eksempel på oppgaver som faller innunder nivå 0. her får eleven en ferdig kode med og det eneste som eleven får beskjed om å gjøre er å trykke på kjør kode. Oppgaven her for å vise elever hvordan løkker kan brukes til å gjenta kommandoer i tekstprogrammering.

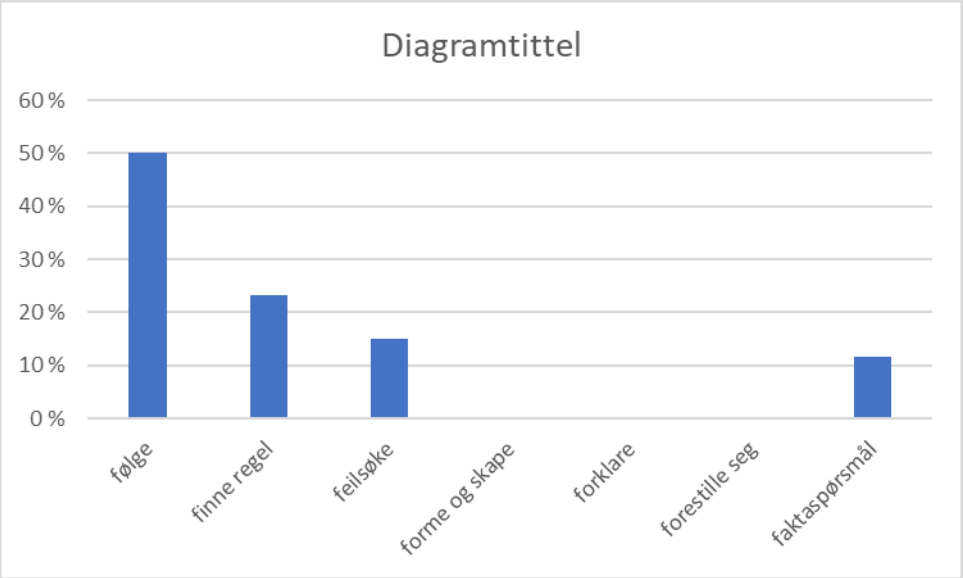


Figur 4-11 oppgave 4.2 i sti-b fra Blokk til Tekstprogrammering

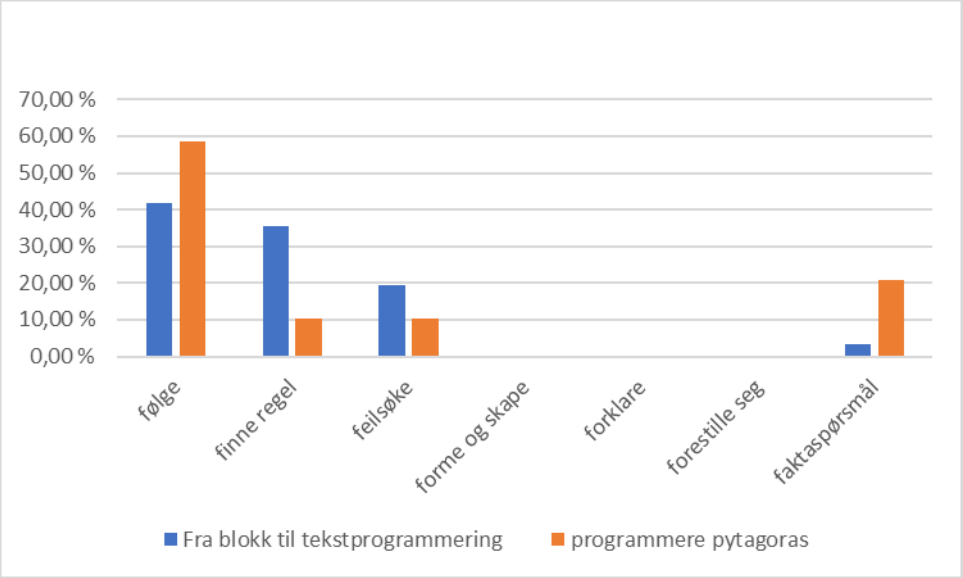
4.2 Oppgavetyper

I likhet med den forrige analysen knyttet til de kognitive kravene har jeg også her laget søylediagrammer som viser både hvor stor prosentandel som hver oppgavetype (Bråting & Kilhamn, 2022) utgjør av det totale antallet oppgaver og innad i hver av temaene. Her er det ingen av oppgavene som faller under flere kategorier, så med andre ord utgjør prosentene til sammen hundre prosent. Denne analysen går også kun på programmeringsoppgaver. Dette betyr at alle oppgaver som kun tar for seg matematisk innhold uten koding ikke vil bli analysert

i denne analysen, som videre betyr at av de 81 oppgavene så er det 31 oppgaver fra blokk til tekstprogrammering og 29 fra Programmere Pytagoras. Altså er 60 av oppgavene analysert med dette analyseverktøyet. Dette betyr også at prosentandelen hver oppgave utgjør er mye nærmere hverandre på tvers av temaene, i motsetning til det som var tilfellet i analysen om kognitive krav. Dette vil også gjøre at det er lettere og sammenligne temaene på tvers i forhold til antall oppgaver som er i hvert tema.



Figur 4-12 søylediagram som viser fordeling av oppgavetyper samlet



Figur 4-13 søylediagram som viser fordeling av oppgavetyper i hvert av temaene

Jeg har også valgt å lage en tabell som viser nøyaktige tall på analysen. Denne tabellen er satt opp på samme måte som tabellen i analysen knyttet til kognitive krav, og viser antall for hvor mange oppgaver som går innunder hver oppgavetype i temaene hver for seg, og hvor stor prosentandel oppgavetypene utgjør av temaene samlet.

	fra Blokk til Tekst	prosent Blokk	ptogrammere pytagoras	prosent pytagoras	totale nr	totale pro
totalt antall	31	100,00 %	29	100,00 %	60	100,00 %
Følge	13	41,94 %	17	58,62 %	30	50,00 %
Finne regel	11	35,48 %	3	10,34 %	14	23,33 %
Feilsøke	6	19,35 %	3	10,34 %	9	15,00 %
Forme og skape	0	0,00 %	0	0,00 %	0	0,00 %
Forklare	0	0,00 %	0	0,00 %	0	0,00 %
Forestille seg	0	0,00 %	0	0,00 %	0	0,00 %
Faktaspørsmål	1	3,23 %	6	20,69 %	7	11,67 %

Figur 4-14 tabell som viser prosent og antall oppgavetyper

213

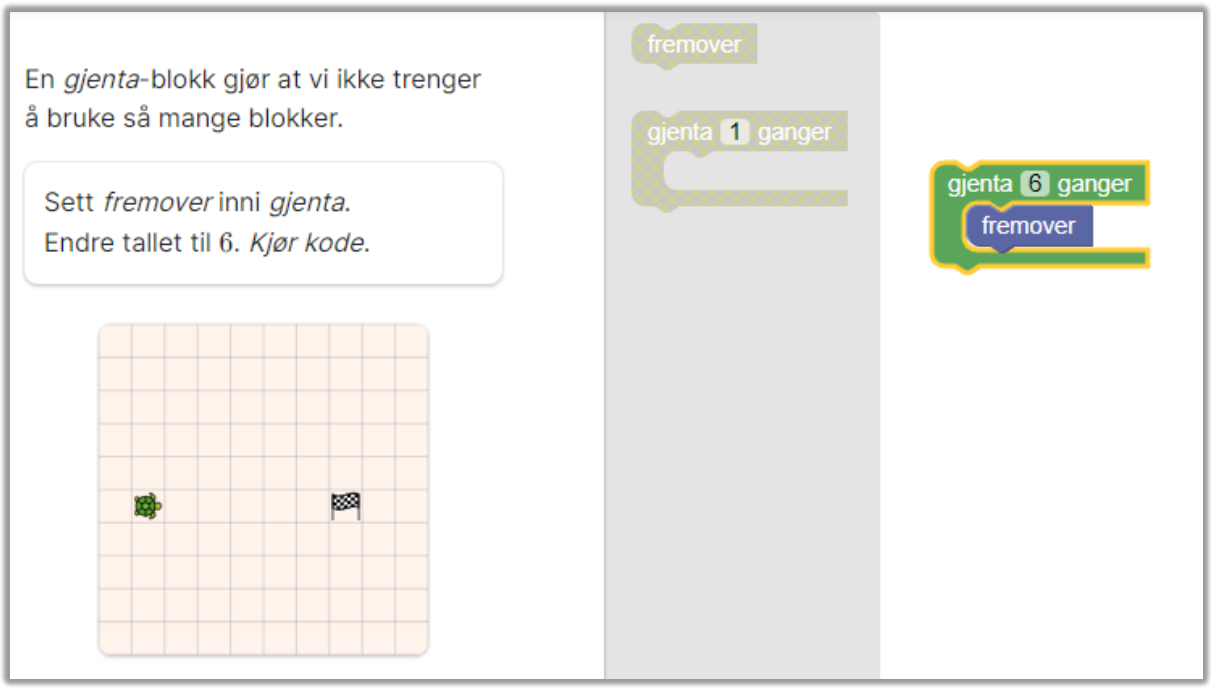
4.2.1 Følge oppgaver

Det er tydelig at oppgavetypen følge hadde den største andelen, både innenfor begge temaene og samlet sett. På tvers av begge temaene utgjorde følge oppgaver hele 50 prosent av det totale antall oppgaver. dette betyr at 30 av de 60 oppgavene.

Når man sammenligner temaene her finner man også en ganske stor forskjell selv om følge er den oppgavetypen det er mest av på tvers av temaene. Det er tydelig en større andel følge-oppgaver i temaet Programmere Pytagoras sammenlignet med temaet Fra Blokk til Tekst. Det er nesten en 17 prosent mer i forskjell i prosentandel til Programmere Pytagoras enn det er i Fra Blokk til Tekst.

eksempeloppgave 1 følge

Denne oppgaven er hentet fra temaet Fra blokk til tekstprogrammering og er oppgave nummer 4.1 i sti B. i denne oppgaven skal man flytte skilpadden til målflagget ved å bruke funksjonen fremover i ei løkke i blokkprogrammering eller en gjenta-blokk som de kaller det i oppgaven. De må også sette inn riktig tall i gjenta-blokken for at den skal repetere fremover funksjonen riktig antall ganger. Det er i denne oppgaven gitt hvordan oppgaven skal løses, med et tydelig steg for steg instruksjon eller oppskrift elevene må følge. Dermed er dette en følge oppgave.

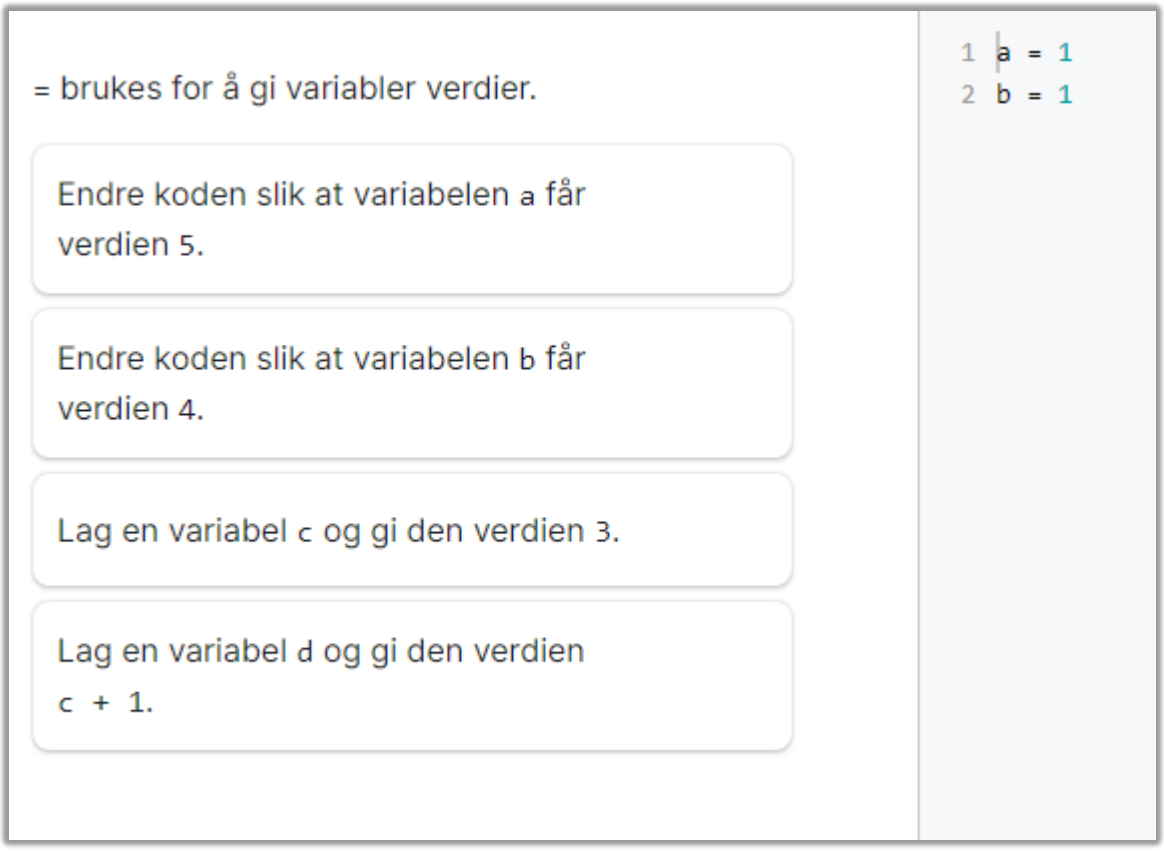


Figur -4-15 oppgave 4.1 i sti-b fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

eksempeloppgave følge 2

Denne oppgaven er hentet fra Programmere Pytagoras og er oppgave 5.2 i introdelen. I denne oppgaven skal elevene lære hvordan de gir variabler verdier. Dette er en veldig enkel oppgave hvor elevene skal lage og sette verdi på fire forskjellige variabler. I likhet med den forrige eksempeloppgaven, så har også denne oppgaven en instruksjon som steg for steg sier hva eleven skal gjøre og er dermed også en følge-oppgave.

bilde



Figur 4-16 oppgave 5.2 i intro Programmere Pytagoras (https://feide.kikora.no/)

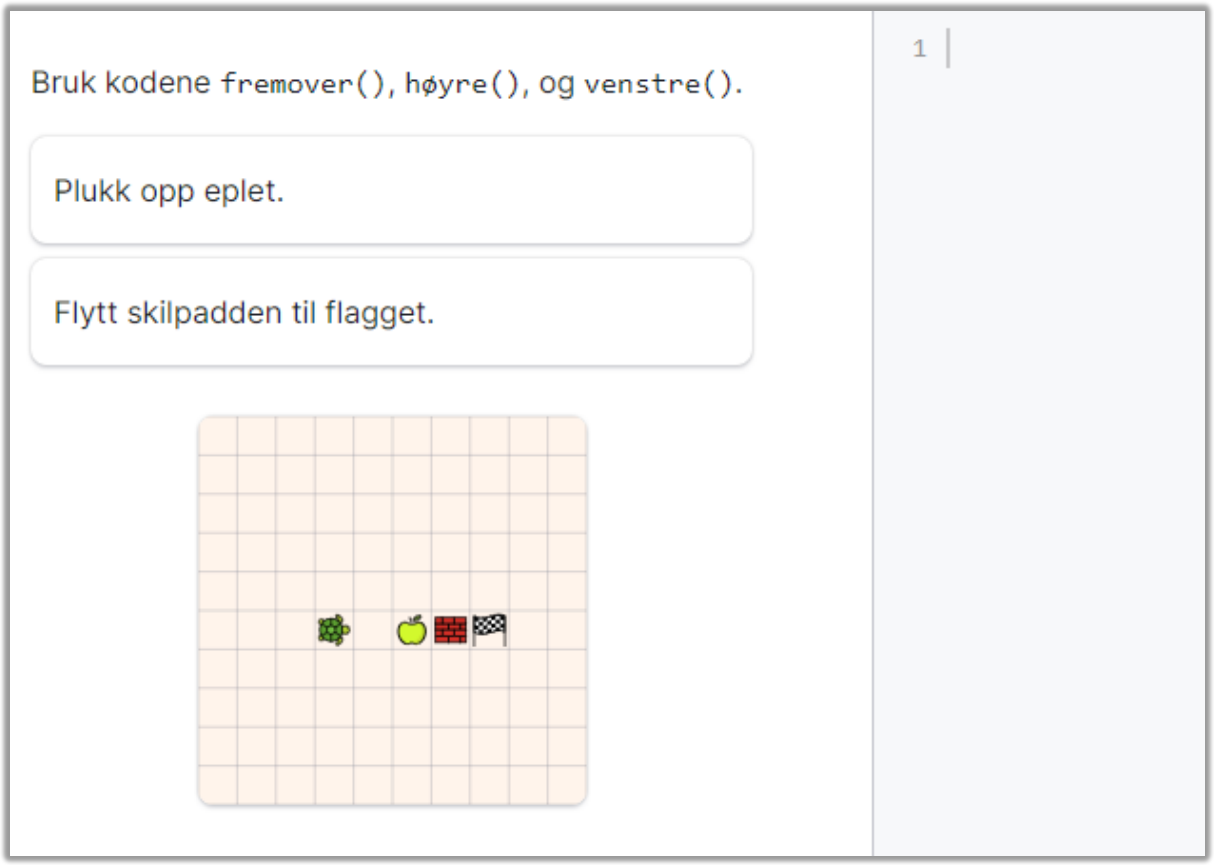
4.2.2 Finne regel oppgaver

Finne regel er den oppgavetypen hvor det er størst forskjell mellom temaene. I temaet Fra blokk til tekstprogrammering så utgjør finne regel en betydelig del av oppgavene. Over 35 prosent av oppgavene i Fra blokk til tekstprogrammering, mens kun litt over 10 prosent av temaet Programmere Pytagoras. Altså ser vi her en stor forskjell på ca. 25 prosent

Eksempeloppgave finne regel

I dette eksempelet blir elevene bedt om å bruke kommandoene som er oppgitt for å styre en skilpadde. I oppgaven må elevene først styre skilpadden mot et eple, deretter mot et målflagg. Det som kjennetegner finne regel oppgaver, spesielt innenfor temaet Fra blokk til tekstprogrammering, er at de ber elevene finne en løsning ved hjelp av kommandoer gitt på forhånd eller gjennom gitte kodefragmenter. I denne sammenhengen består disse

kommandoene av instruksjoner som påvirker skilpaddens bevegelser, slik som framover, venstre, høyre og bakover. Elevenes oppgave blir dermed å arrangere og ordne disse kommandoene i riktig rekkefølge. Dette blir tydeliggjort gjennom eksempelet der elevene bruker kommandoene for å navigere skilpadden først mot et eple, etterfulgt av å styre den mot et målflagg, altså må elevene finne den riktige prosedyren for at skilpadden skal gjøre det de vil.



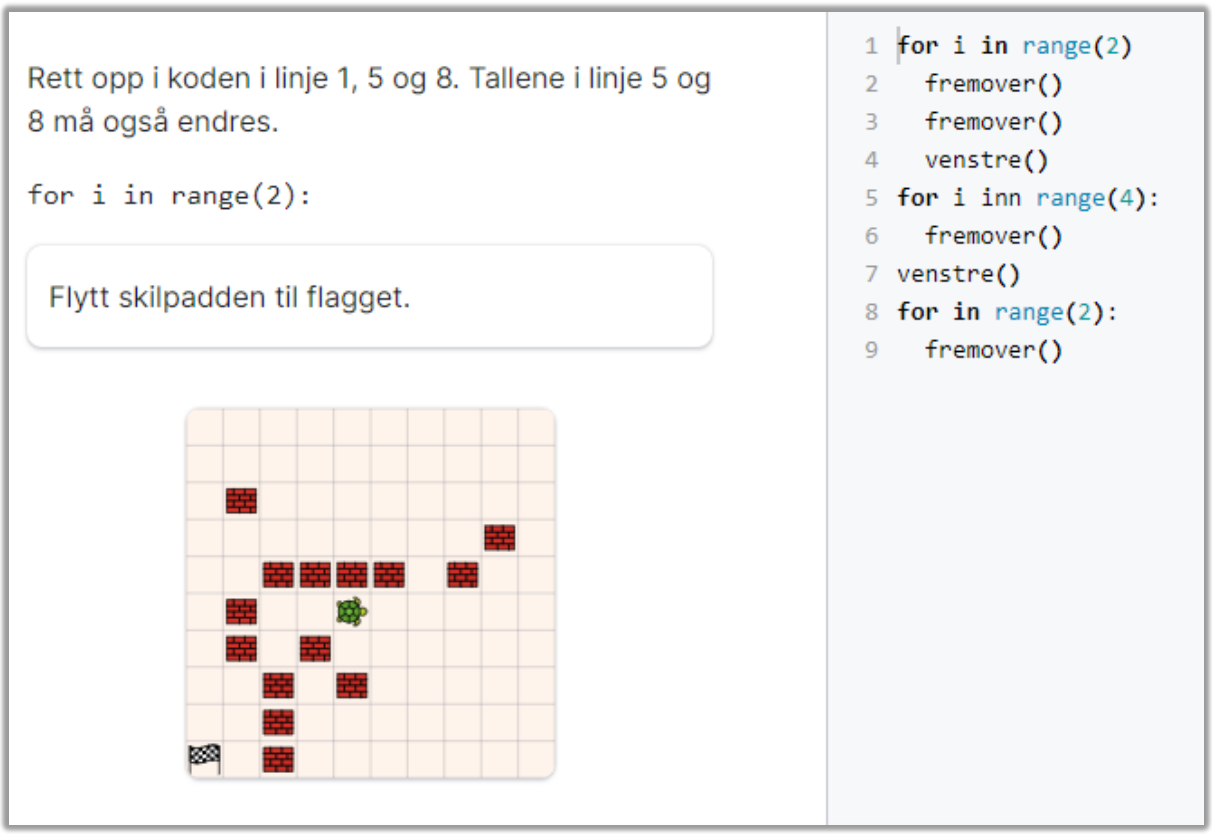
Figur 4-17 oppgave 1.4 i sti-b fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

4.2.3 Feilsøke oppgaver

Når det kommer til oppgavetypen feil søke eller debugging så er dette den oppgavetypen med minst forskjell mellom temaene. Her utgjør oppgavetypen ca. 20 prosent på begge temaene og utgjør derfor også totalt 20 prosent av alle oppgavene totalt

Eksempeloppgave feilsøke

Denne oppgaven er et godt eksempel på debugging eller feilsøkeoppgaver. Oppgaven er hentet fra Fra blokk til tekstprogrammering og er oppgav nummer 6.1 i sti b. I likhet med flere av de tidligere eksempeloppgavene skal denne koden styre en skilpadde til et målflagg. Det er lagt inn flere feil i denne koden som elevene må finne og rette opp i. Det er totalt 5 feil oppgaven. Oppgaveteksten sier på hvilken linje feilene ligger, men elevene må fortsatt selv undersøke koden for å finne feilene. Hadde oppgaveteksten sagt nøyaktig hva som feil og hvordan man skulle endre det hadde oppgaven vært en følge oppgave isteden, men fordi elevene selv må undersøke og lete fram feilene er dette en feilsøkeoppgave.



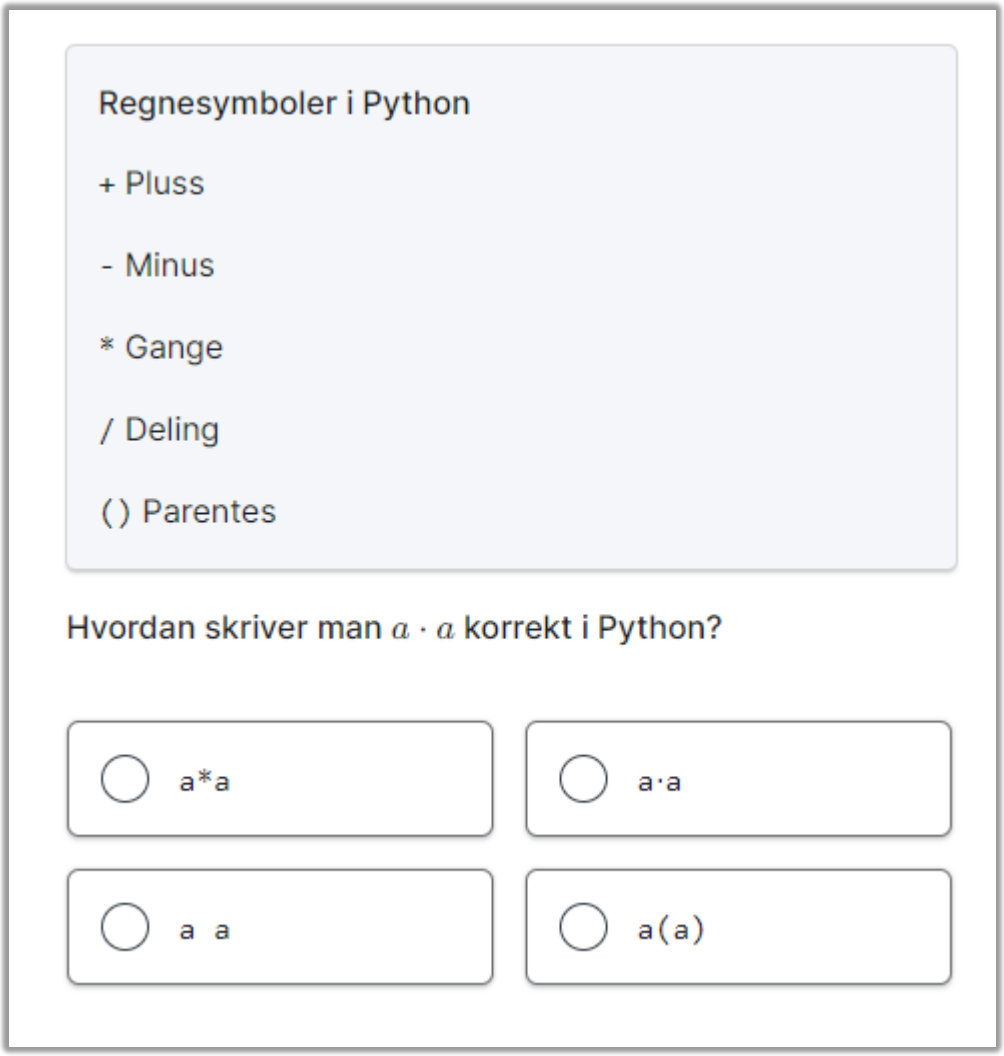
Figur 4-18 oppgave 6 fra sti-b fra Blokk til Tekstprogrammering (https://feide.kikora.no/)

4.2.4 Faktaspørsmål

Innen oppgavetypene man finner, er fakta spørsmål den oppgavetypen det helt klart er minst av. Det er kun 7 oppgaver totalt som faller innunder denne kategorien og det utgjør ikke mer enn litt under 7 prosent av alle oppgavene. Det kan påpekes at det er tre ganger så mange oppgaver av denne typen i temaet Programmere Pytagoras enn det er i Fra blokk til tekstprogrammering, men det vil være litt misvisende siden det er snakk om fire oppgaver fordelt på to temaer.

Eksempeloppgave Faktaspørsmål

Denne oppgaven er hentet fra temaet Programmere Pytagoras og er oppgave 6.2 fra introdelen. Denne typen oppgaver ber ikke elevene om å gjøre noe annet enn å svare direkte på et spørsmål om hva som er riktig. oppgaven spør om hvordan man skriver a gange a i Python. Svaret er a*a. på akkurat denne oppgaven ligger svaret i teksten over og elevene trenger kun å lese seg fram til hva som er riktig. Her kreves det heller ingen jobbing med koder for å kunne svare. Denne oppgaven går på grunn av alt dette under oppgavetypen Faktaspørsmål.



Figur 4-19 oppgave 6.2 i intro Programmere Pytagoras (https://feide.kikora.no/)

4.2.5 Feilsøke oppgaver i form av flervalg

Noen av oppgavene er flervalgsoppgaver. Disse oppgavene er i størst grad fakta spørsmål fordi det blir stilt et spørsmål om hva som er riktig. for eksempel hva betyr ordet True. Deretter velger man det alternativet eller alternativene man mener inneholder riktig svar. Utafra dette kan man påstå at alle flervalg oppgaver som stiller et objektivt spørsmål er et fakta spørsmål, men jeg har valgt og plassere noen av disse oppgavene under oppgavetypen feil søke isteden. Dette har jeg gjort fordi noen av oppgavene spør etter hvilke av alternativene inneholder en kode som er skrevet riktig. Disse oppgavene legger opp til at elevene skal undersøke alternativene med den kunnskapen de har lært tidligere og se etter feil som kan forårsake at koden ikke kjører som man har tenkt. Dermed legger disse oppgavene opp til en annen tenkemåte eller annen måte og jobbe på enn det de andre flervalgsoppgavene gjør. Derfor har

jeg valgt å legge disse under oppgavetypen feil søke istedenfor fakta spørsmål hvor de andre flervalgsoppgavene er plassert.

4.2.6 Manglene oppgavetyper

Analysen viser at ingen av oppgavene som jeg har sett på går under oppgavetypene forme og skape, forklare og forestille seg. Dette betyr at i Temaene Fra blokk til tekstprogrammering og Programmere Pytagoras ikke inneholder programmeringsoppgaver som ber elevene skape helt egne koder, forklare med egne ord, eller oppgaver hvor elevene skal skape egne hypoteser om hva som kommer til å skje når de kjører en kode. Dette er et åpenbart klart funn som også kommer fram senere i kapitlet i forbindelse med affordances og constraints.

4.3 Analyseverktøyene sett oppimot hverandre.

Siden dette bygger på analysen om oppgavetyper så får vi også her kategorier uten noen resultater. Jeg har derfor under utelatt og skrive om oppgavetypene forme og skape, forklare, og forestille seg. Resultatene for disse kategoriene vil være (Bråting & Kilhamn, 2022) oppgaver innen alle nivåene. Dette sammen med at tabellen blir mye mer oversiktlig er grunnen til at de ikke er inkludert i tabellen under.

I analysen for oppgavetype får man en oversikt over det totale antall oppgaver som går under de forskjellige oppgavetypene. Dette kan man også se her, men fordi det alt har blitt gjennomgått i tidligere resultater så kommer det ikke til å nevnes her. Derimot når det kommer til kognitive krav så vil dette vise ny informasjon ved at det kun er programmeringsoppgaver her. Altså antallet av de diverse kognitive kravene som de rene matteoppgavene utgjør vil ikke nevnes her. Derfor vil dette bli sett på. det vil også bli sett på hvilke kognitive krav som går innunder de aktuelle oppgavetypene.

	nivå 0	nivå 1	nivå 2	nivå 3	nivå 4
Følge	4	0	20	5	1
Finne regel	0	0	8	3	3
Feilsøke	0	5	0	0	4
Faktaspørsmål	0	7	0	0	0

Figur 4-20 tabell som viser hvilke kognitive krav de forskjellige oppgavetypene krever.

Tabellen over er satt opp ved at man på venstre siden har de aktuelle oppgavetypene og øverst på tabellen har de kognitive kravene. Dette er satt opp slik at man kan enkelt kan se hvor mange av de diverse oppgavetypene som går under de forskjellige kravene. For eksempel kan man enkelt se at det er 4 oppgaver som går under oppgavetypen følge, som også går under nivå 0 på de kognitive kravene.

4.3.1 Kognitive kravene

Man kan tydelig se at det er en overvekt av programmeringsoppgavene som faller innen nivå 2. altså utgjør prosedyrer uten sammenhenger nesten halvparten av de 60 oppgavene. I motsetning til dette utgjør nivå 0 bare 4 oppgaver av det totale. Nivå 1 utgjør 12 av oppgavene. De to siste nivåene ligger mye tettere hverandre da både nivå 3 og 4 begge utgjør 8 oppgaver hver. Hvis man ser bort ifra nivå 2 så varierer det bare med 8 oppgaver fra nivå 0 med minst oppgaver til nivå 1 med 12. dette i seg selv virker som en ganske liten variasjon, men prosentvis er det en ganske stor forskjell mellom dem. for nivå 0 utgjør bare litt over 6,5 prosent, mens nivå 1 utgjør hele 20 prosent.

4.3.2 Oppgavetyper

Følge

Av følge oppgavene var det i likhet med det som står over aller flest oppgaver innen nivå 2, altså prosedyre uten sammenhenger. Følge er også den kategorien som utgjør mest av oppgavene som går innen nivå2

Det var også fire av oppgavene som går innunder nivå 0, som betyr at alle eksempeloppgavene var følge oppgaver. ellers var det ingen oppgaver innen nivå 1, det var noen få nivå 3 oppgaver og kun en oppgave innen nivå 4

Finne regel

Finne regel hadde i likhet med følgeoppgaver, flest oppgaver innen nivå 2. ellers var resten av oppgavene fordelt likt på nivå 3 og 4. altså av 14 finne regel oppgaver var det 8 oppgaver under nivå 2 og 3 oppgaver under nivå 3 og 4.

Feilsøke

Feilsøke oppgavene gikk enten innunder nivå 4 eller nivå 1. det var fire oppgaver innunder nivå 4 og 5 oppgaver under nivå 1.

Faktaspørsmål

Fakta spørsmålene hadde minst variasjon da alle 7 oppgavene gikk under nivå 1, altså Memorering.

4.4 Affordances og constraints

Kikora som et digitalt læreverk introduserer unike elementer som ikke finnes i tradisjonelle lærebøker. Utforskningen av affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014) i Kikora har avdekket flere funn. Det er helt tydelige funn knyttet både til affordances og constraints, som tydelig avdekker områder av hva Kikora er i stand til som et digitalt læreverk.

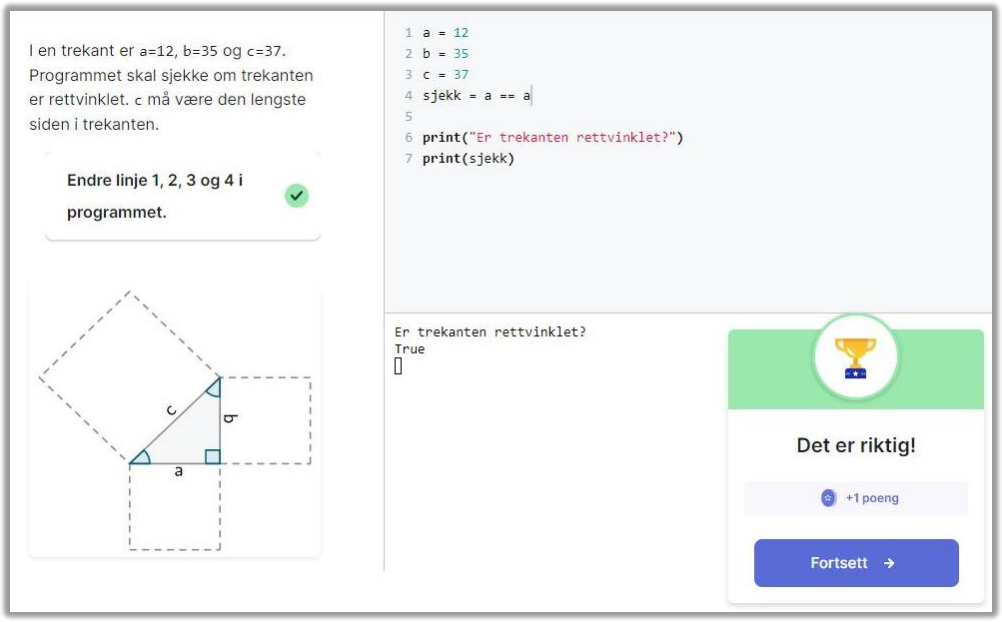
4.4.1 Oppgaver kan løses feil.

I Kikora er det flere oppgaver som kan løses feil. Det jeg mener med det er at det er mulig å få fram tilbakemeldingen som sier riktig, selv om oppgaven ikke er løst. Utafra oppgavene hvor dette har skjedd, og måten det er gjort på så ser det ut til kun å være mulig på oppgavene som inneholder tekstprogrammering. Det er ikke mulig å si noe om direkte årsak uten å ha tilgang til hvordan Kikora funger helt nøyaktig, men antagelig når Kikora bruker tekstprogrammering så ser Kikora i hovedsak etter et spesifikt utfall som deretter sier ifra at oppgaven er løst riktig. for eksempel la oss si at Kikora vil at man skal skrive 2+2=4. Da vil 4 være utfallet Kikora ser etter og dermed hvis man skriver 1+3=4 så vil det også gi riktig selv

om målet med oppgaven var å skrive 2+2. igjen så er ikke dette sikkert, men det stemmer overens med det som er sett i Kikora.

Eksempeloppgave:

I oppgave 7,4 i del b fra Programmere Pytagoras blir du bedt om å endre en kode slik at den sjekker om en trekant er rettvinklet. Det Kikora vil at man skal gjøre er å endre slik at koden sjekker at hypotenus opphøyd i andre er det samme som katet opphøyd i andre pluss det andre katetet opphøyd i andre. Altså Kikora vil at vi skal skrive $sjekk = c ** 2 == a ** 2 + b ** 2$ eller $sjekk = c * c == a * a + b * b$. Dette vil kunne gi oss enten svaret True eller False ettersom hvorvidt trekanten som er oppgitt er rettvinklet eller ikke. I denne oppgaven vil vi få utfallet True fordi trekanten som er oppgitt er rettvinklet. Jeg testet litt og prøvde å sette inn $sjekk = a == a$. Dette skal også kunne gi oss et True eller False svar, men siden a alltid vil være det samme som seg selv, så vil jo dette alltid gi oss utfallet True. Når jeg puttet dette inn fikk jeg opp som vist på bilde under at jeg hadde løst oppgaven riktig selv om jeg ikke hadde brukt Pytagoras. Dette tyder jo på at i denne oppgaven så sjekker Kikora om utfallet blir True, og ikke om hvor vidt man faktisk har sjekket om en trekant er rettvinklet.



Figur 4-21 oppgave 7.4 i sti-b Programmere Pytagoras (https://feide.kikora.no/)

Dette er som nevnt en av flere oppgaver hvor man kan unngå å løse oppgaven ved å anta hva Kikora ser etter. Dette tyder på en tydelig begrensing i Kikora.

4.4.2 Selvrettende oppgaver

Effektivitet for læreren: En annen affordance (Humble & Mozelius, 2023; Gibson, 2014) er som nevnt at Kikora frigjør lærerens tid og ressurser ved å ta seg av vurderingen. Når oppgavene er selvrettende, kan læreren bruke mindre tid på å rette enkeltoppgaver og heller fokusere mer på veiledning og tilrettelegging. Dette gir læreren generelt mer tid til å gjøre andre ting knyttet til lærerjobben.

Som nevnt tidligere så kan man løse oppgaver feil og fortsatt få de opp som riktig. Dette kan skje uten at læreren legger merke til det, nettopp fordi Kikora retter seg selv, og fordi det retter seg selv skal man ikke trenge å se over oppgavene etterpå. Selv om Kikora er designet for å vurdere elevenes svar automatisk, er det ikke alltid i stand til å fange opp feil eller feilaktige løsninger som ikke passer inn i det forhåndsdefinerte svaret.

Tilbakemelding med en gang: En av de større affordancene ved Kikora er muligheten for elevene å motta tilbakemelding på oppgavene sine med en gang. Dette betyr at elevene ikke

trenger å vente på at læreren skal rette oppgavene deres, noe som kan være både tid og ressurskrevende. Dermed kan elevene med en gang oppdage eventuelle feil.

En constraint ved Kikora er at tilbakemeldingene som gis, ofte er begrenset til om svaret er riktig eller ikke. Dette betyr at det er veldig begrenset i hvilken gard elevene kan få veiledning eller kvalitative kommentarer og tilbakemeldinger på oppgavene. Spesielt når det gjelder å forklare hvorfor et svar er feil. Elevene kan dermed gå glipp av en dypere forståelse av fagstoffet fordi fokuset er på hvor vidt oppgavene er riktige og ikke på å forstå hvorfor ting fungerer som det gjør.

4.4.3 Begrenset hvilke oppgavetyper man finner

Som nevnt flere ganger tidligere så var det noen oppgavetyper (Bråting & Kilhamn, 2022) som ikke var å finne fra analyseverktøyet for programmeringsoppgavene. Dette er en tydelig constraint ved å bruke Kikora, nettopp fordi Kikora ikke inneholder disse oppgavene.

Alle oppgavene i Kikora ser ut til å ha ett riktig svar, med unntak av flervalgsoppgaver som tillater flere svaralternativer. Dette viser at Kikora primært fokuserer på oppgaver som har en definert, korrekt løsning.

Det ble også tydelig identifisert med at noen oppgavetyper ikke er til stede i Kikora. For eksempel er oppgavene Forklare og Se for seg fraværende. Disse oppgavene krever at elevene formulerer egne forklaringer og beskriver sine tanker og resonnementer. Med andre ord gir ikke Kikora elevene muligheten til å uttrykke sine egne forståelser og resonnementer. Dette utgjør en klar constraint i Kikora.

På samme måte er oppgavene Forme og skape også fraværende i Kikora. Disse oppgavene gir elevene mulighet til å utvikle egne koder ved å bruke de funksjonene de har lært. Siden funksjoner kan anvendes på forskjellige måter, gir disse oppgavene rom for variasjon og kreativitet. Dette viser at fravær av disse oppgavene er tydelig er i tråd med det jeg nevnte over om at alle oppgavene ser stort sett ut til å kun ha et riktig svar.

4.4.4 Kan ikke erstatte dialogen mellom lærer og elev

Selv om Kikora gir muligheten for selvrettende oppgaver, mangler det den menneskelige interaksjonen og tilpasningen som en lærer kan tilby. Å ha en lærer som kan engasjere seg i dialog, svare på spørsmål og tilpasse undervisningen etter elevenes behov. Man mister også som påpekt tidligere muligheter til å fange opp misoppfatninger. Kikora kan ikke erstatte den personlige tilstedeværelsen og den unike tilpasningen som en lærer kan tilby bare ved å være til stede og gi en mulighet for dialog.

5. Drøfting

5.1 Kognitive krav

Gjennom analysen av kognitive krav (Stein & Smith, 1998) ble det oppdaget flere interessante funn som kan gi en verdifull innsikt i oppgavene til Kikora. Blant resultatene, dukket det opp noen veldig overraskende funn. Det mest overaskende funnet var den tydelige overvekten av oppgaver som falt innenfor de lavere kognitive nivåene. Dette funnet reflekterer at oppgavene som elevene ble presentert med i stor grad krevde lavere kognitive ferdigheter, som memorering og prosedyrer uten sammenhenger. Hvis man ser nærmere på prosentandelen som oppgavene utgjør i nivåene så kan man se hvor stor del av oppgavene som faller innunder de lavere nivåene. Selv om det er verdt å merke seg at tre av oppgavene går inn under både nivå 1 og nivå 4, men hvis man kun hadde telt de tre oppgavene under nivå 4, så ville fortsatt nivå 1 og 2 sammen utgjøre mer enn 71,6 prosent av alle oppgavene. Dette betyr at mer enn sju av hver tiende oppgavene plasserte seg på de lavere nivåene, og dette er uten å inkludere eksempeloppgavene, altså nivå 0.

5.1.1 Sett oppimot tidligere forskning på kognitive krav

Om vi tar i betraktning Steins og Smith (1998) tidligere forskning som viste at de elevene som ble utsatt for oppgaver på et høyere kognitivt nivå, oppnådde bedre resultater innen problemløsning og resonering, og vi legger disse funnene i sammenheng, reises spørsmålet om hvordan de lavere nivåene av kognitive krav (Stein & Smith, 1998) i de analyserte oppgavene kan påvirke elevene. For utfra det Stein og Smith observerte så vil jo dette kunne ha en negativ påvirkning. Og dette vil jo da være et argument for at å kun bruke Kikora til å lære elevene programmering ikke er nokk, altså bør elevene ut ifra dette heller få oppgaver som i større grad går under de høyere kognitive kravene.

Det er også verdt å tenke på at det er noen forskjeller mellom funnene gjort her og det Smith og Stein. For det første er det verdt å bemerke at Smith og Steins observasjoner ble gjort i klasseromsmiljøer, der oppgavene ble brukt som en del av undervisning. I kontrast til dette er oppgavene som har blitt analysert her, hentet fra et digitalt læreverk. Denne forskjellen i kontekst kan gjøre at det er variasjoner i hva som kreves av oppgavene for at de skal kunne ha høyere kognitive krav. Det er mulig at Kikora som digital plattform kan gi andre muligheter

og tilnærminger som man ikke har i en klasseromssammenheng og motsatt, spesielt med tanke på at lærerne selv kan ha en stor påvirkning i måten de velger å presentere eller gi elevene en oppgave.

5.1.2 Funn i lys av Blooms taksonomi

Hvis man ser på dette oppimot Blooms taksonomi (Adams, 2015) så kan man også her se at de fleste oppgavene i stor grad peker mot de laveste nivåene i denne modellen også. Dette støtter oppunder iden om at de valgte temaene har som hensikt å introdusere konsepter og bygge et fundament for videre læring. Ved å konsentrere seg om de lavere nivåene som Kunnskap og forståelse, så legger disse temaene grunnlaget som er nødvendig for å forstå mer komplekse tankeprosesser. Denne tilnærmingen gir en gradvis progresjon der elevene først må mestre de grunnleggende elementene før de kan utforske mer dyptgående og utfordrende konsepter. Dermed kan man argumentere for at denne tilnærmingen gir en effektiv læringsbane ved å bygge en solid kognitiv plattform som gir elevene nødvendige verktøy for å engasjere seg i mer avansert tenkning senere. Det er vanskelig å si noe om hvor riktig dette er. Spesielt med tanke på at dette går direkte imot det Stein og Smith (1998) så i sine observasjoner. Nemlig det at elever som får oppgaver som stiller høyere kognitive krav av gjør det bedre knyttet til problemløsing enn de som får oppgaver som krever lavere kognitive krav. For ved å se det oppimot Blooms taksonomi så fungerer Kikora som et bra læreverk til å bygge en grunnmur for elvene som de kan bygge videre på, men at de må senere få oppgaver som krever mer enn det Kikora har.

5.1.3 Påvirkning av temaenes oppsett

man bør også ta i betraktning temaenes oppsett. Oppgavene i Kikora er satt opp så man har forskjellige oppgavesett eller temaer hvor man går fra oppgave til oppgave. Det er mulig at det er satt opp som en progressiv læringsbane der tidligere oppgaver legger grunnlaget for å håndtere mer komplekse utfordringer senere. Dette passer veldig godt med noe av det vi så i Fra blokk til tekstprogrammering, der man først fikk en oppgave i form av blokkprogrammering, forså at man får den samme oppgaven i form av tekstprogrammering. Elevene løser da oppgaven først med blokkprogrammering som har en enklere oppbygning og når de da skal løse den samme oppgaven med tekstprogrammering så vil de ha bedre forutsetning for å få det til. Slik kan enkelte oppgaver fungere som forberedelse til mer

utfordrende oppgaver, og dermed kan den totale gjennomgangen av oppgavene gi en mer helhetlig vurdering av kognitive evner, men på grunn av dette så vil da den første oppgaven være på et lavere kognitivt krav for at man skal få den ønskede effekten. Dette vil bety at man ikke kan se på oppgavene på den måten som er gjort i forbindelse med kognitive krav (Stein & Smith, 1998) fordi man må da se på temaene i en helhet, men dette er noe som er utenfor det som er mulig å se i analysen gjennomført i denne oppgaven

5.1.4 Målsetningen til oppgavene

Et annet området handler om målsetningen med oppgavene. Hvis målet er å lære en bestemt kommando eller regel innenfor Python-programmering, kan det være nødvendig å først introdusere disse elementene i en enklere kontekst. Det kan være mulig at det å lage oppgaver med høyere kognitive krav (Stein & Smith, 1998) er mer utfordrende å implementere i de tidligere delene av temaene. Det er også mulig at dette ikke er like relevant i andre temaer. Temaene jeg har valgt introduserer mye nytt. Spesielt Fra blokk til tekstprogrammering som tydelig er et tema med mål om å introdusere tekstprogrammering. Det er også mulig at oppgavene skal være enkle for at det ikke skal være for høy Vanskelighetsgrad på oppgavene, men dette er også noe som ligger utenfor det som er undersøkt i denne studien

5.1.5 Forskjell på temaenes kognitive krav

Det er tydelig at det er vesentlige forskjeller i andelen av oppgaver som faller innenfor ulike kognitive krav (Stein & Smith, 1998) mellom de to temaene, Fra blokk til tekstprogrammering og Programmere Pytagoras. Disse forskjellene kan tyde på at det er ulikheter i tilnærmingen og hensikten for hvert oppgavesett.

I Programmere Pytagoras utgjør oppgaver på nivå 1 hele 56% av oppgavene, en betydelig høyere andel enn det som finnes i Fra blokk til tekstprogrammering. Dette kan indikere at det er et sterkt fokus på grunnleggende konsepter og kommandoer innen programmering. Siden temaet handler om å bruke Pytagoras læresetning i programmering, er det rimelig å anta at når oppgavene ble lagd lå fokuset på matematisk innhold og selve implementasjonen av algoritmer og nye kommandoer. Det er også verdt å nevne at i analysen om oppgavetype (Bråting & Kilhamn, 2022) så ble bare 29 av 50 oppgaver analysert fordi det analyseverktøyet gikk kun på programmeringsoppgaver. Det betyr at det er 21 oppgaver som kun hadde fokus på det matematiske innholdet. Dette kan ha resultert i et større antall av oppgaver på nivå 1.

På den annen side er andelen oppgaver på nivå 2 i Fra blokk til tekstprogrammering på hele 54,8%, som igjen er en vesentlig høyere andel enn det som er i det andre temaet Programmere Pytagoras. Dette kan skyldes at oppgavesettet Fra blokk til tekstprogrammering har som mål å introdusere tekstbasert programmering. Der kan det være et mye større behov for å gi elevene en solid forståelse av programmeringskonsepter og hvordan man skal bruke diverse prosedyrer.

Som nevnt, indikerer resultatene en tendens mot de lavere nivåene i Blooms taksonomi (Adams, 2015). Når vi derimot ser på de to ulike temaene individuelt, så er det en viss forskjell. For eksempel har Programmere Pytagoras som nevnt flest oppgaver som faller under nivå 1. Som påpekt tidligere samsvarer nivå 1 eller memorering godt med det nivået som kalles kunnskap i Blooms taksonomi, fordi det er et stort fokus på å huske og gjengi informasjon. Man kan derfor argumentere for at Programmere Pytagoras fokuserer på å legge grunnlaget, med hovedvekt på å etablere en grunnleggende kunnskapsbase.

På den andre siden har Fra blokk til tekstprogrammering flest oppgaver som hører til nivå 2. Dette nivået går mer mot en kombinasjon av kunnskap og forståelse i Blooms taksonomi (Adams, 2015), da det går litt dypere enn ren memorering. Dette tyder på at Fra blokk til tekstprogrammering beveger seg videre fra bare å memorere informasjon, og prøver på å utvikle en dypere forståelse av konseptene. Dette kan betraktes som et skritt videre fra et grunnleggende nivå av kunnskap til en mer nyansert innsikt. Med tanke på at man beveger seg vekk fra blokkprogrammering og over til testprogrammering kan det tenkes at elevene allerede er ment å ha en grad av forståelse ved at de tidligere har vært igjennom blokkprogrammering og dette ses på som et steg videre, men det er ikke mulig å bekrefte at dette er tilfelle fordi dette er utenfor det denne studien har undersøkt

Disse forskjellene i mengden av oppgaver innen de forskjellige kognitive kravene (Stein & Smith, 1998) kan indikere en bevisst strategi i hvert tema. Programmere Pytagoras ser ut til å prioritere et solid fundament med fokus på rene fakta og konsepter, mens Fra blokk til tekstprogrammering synes å legge vekt på å bygge forståelse og evnen til å anvende kunnskapen i forskjellige sammenhenger. Slik sett kan man tolke den ulike fordelingen som en refleksjon av temaenes overordnede pedagogiske mål.

5.1.6 Algoritmiske tenkeren krever mer

Som nevnt flere ganger har algoritmisk tenking fått en sentral plass i kjerneelementene i læreplanen for matematikk (Kunnskapsdepartementet, 2019). Dette er i forbindelse med å styrke elevens evner knyttet til problemløsing. Den algoritmiske tenkeren (Utdanningsdirektoratet, 2019) som viser de sentrale aspektene ved algoritmisk tenking. Det er flere punkter som tydelig peker mot de høyere kognitive kravene. For eksempel Evaluering og abstraksjon. Begge disse punktene innebærer at elevene må ta egne valg og vurderinger. Dette samsvarer veldig med gjøre matematikk eller nivå 4 av de kognitive kravene (Stein & Smith, 1998). Dette betyr at elevene trenger å få oppgaver som ligger på de høyere kognitive kravene for å utvikle evnene sine knyttet til problemløsning. Altså funnene i analysen sett oppimot at algoritmisk tenking er en del av læreplanen tyder på at oppgavene i Kikora ikke er nokk til å fult utvikle elevenes evner knyttet til problemløsning, men at det kan brukes for å danne et grunnlag som videre kan bygges på.

5.2 oppgavetyper

når det kom til analysen om oppgavetyper (Bråting & Kilhamn, 2022) som var å finne i Programmeringstypene så var det noen store og veldig overaskende funn. Det er ikke overaskende i seg selv at det var en skjev fordeling, men at hele 50 prosent var følge oppgaver var uventet. Spesielt med tanke på at programmering er et emne som krever utforskning og arbeidsmetoder som Fikle og feil søke som den algoritmiske tenkeren (Utdanningsdirektoratet, 2019) nevner. Et annet funn som var veldig uventet var at det var flere oppgavetyper som ikke var å finne. Det at det mangler oppgavetyper kan ha en stor påvirkning på hva slags innhold elevene får og kan være en stor begrensning. Selv om dette var et stort funn her, så kommer jeg også til å se på dette i slutten av kapittelet i forbindelse med affordances og Constraints for å belyse dette bedre.

5.2.1 Algoritmisk tenking og PRIMM

Om vi ser på fordelingen av oppgavetyper (Bråting & Kilhamn, 2022) i lys av den Algoritmiske tenkeren (Utdanningsdirektoratet, 2019) og PRIMM (Sentance et al, 2019) så kan det tyde på at det er lite variasjon i hvordan eleven jobber med programmering. Hvis man tar for seg PRIMM, så forklares en helt tydelig rekkefølge eller progresjon i hvordan man kan jobbe med programmering, men utfra de oppgavetypene som ble funnet i analysen ser det ut

som man i stor grad kan miste forskjellige steg. Spesielt Predict og Make. Her kommer tydelig fram at det er oppgavetyper som ikke ble funnet. Både forklare og forestille seg er oppgavetyper som går på at elevene skal forklare med egne ord og skal se for seg hva som kan skje når man kjører en kode, dette er de oppgavetypene som best representerer Predict. Man kan se det samme med Make ved at det ikke er noen oppgaver under forme og skape da dette handler om å skape en egen kode. Dette tyder på at disse måtene å jobbe på mangler i Kikora. Hvis man ser på de andre oppgavetypene, så synes dette her også ved at de naturlig faller innunder de andre delene av PRIMM, altså Run, Investegate og Modify. Det samme kan man se oppimot Den algoritmiske tenkeren (Utdanningsdirektoratet, 2019). For her er det også tydelig at det er mangler som for eksempel punktet Logikk. Logikk går som nevnt på at elevene skal forutsi og forklare hvordan Koder fungerer, altså går det innunder det samme som Predict. Det er også arbeidsmåter som blir nevnt i den algoritmiske tenkeren som blir borte. Skape handler om det samme som Make fra PRIMM og er også da manglende. Begge modellene peker på at det er mangler i hvordan oppgavene gir elevene mulighet til å jobbe med Programmering. Dette støtter oppunder det som ble nevnt før om at Kikora alene ikke er nokk og må supplementæres med andre oppgaver.

5.2.2 Mulig oppgavene er lagd på samme måte som tradisjonelle matematikkoppgaver

Som nevnt så ser det ut som en skjevfordeling i hvilke oppgavetyper (Bråting & Kilhamn, 2022) som er i Temaene. Spesielt med tanke på at det er så mange følge oppgaver. dette sammen med at det er 23 prosent av oppgavene som er finne regel gjør at 73 prosent av alle oppgavene går under 2 av de 4 kategoriene som ble funnet. hvis man ser på dette i lys av hvordan en helt standard matteoppgave er, så vil følge og finne regel være de oppgavene man antagelig ser mest av. Dette er fordi i de aller fleste sammenhenger går matteoppgaver ut på å løse et mattestykke, enten ved at man bare får oppgaven og må bruke det man har lært fra før som vil tilsvare finne regel eller ved at det er gitt en spesifikk metode man må bruke som da vil tilsvare en følge oppgave. Altså ut fra hvordan man vanligvis møter matteoppgaver gir dette mening. Med andre ord så er de oppgavetypene det er mest av, også de oppgavetypene som ligger nærmest en tradisjonell oppgave hvor man får et mattestykke og skal vise utregning, men dette er jo også en antagelse ut ifra hvordan en standard matteoppgave ser ut, noe som ligger utenfor hva denne studien har sett på.

5.2.3 Forskjell mellom temaene i forhold til oppgavetype

Selv om Temaene er mer lignende når det kommer til oppgavetype (Bråting & Kilhamn, 2022) enn de var i forhold til de kognitive kravene (Stein & Smith, 1998) så er det fortsatt tydelige forskjeller her. Den største forskjellen er at temaet Fra blokk til tekstprogrammering har en jevnere fordeling av oppgavetypene enn det Programmere Pytagoras har. Nesten 60 prosent av oppgavene i Programmere Pytagoras er følgeoppgaver. Ut ifra dette virker det som Fra blokk til Tekstprogrammering har et større fokus på selve programmeringen enn det Programmere Pytagoras har. Det virker som når matematikken kommer inn så deles fokuset mellom det å skulle lære elevene å programmere, og det å skulle lære elevene selve matematikken. Av de 50 oppgavene som er i temaet Programmere Pytagoras så ble 29 oppgaver sett på i forhold til oppgavetyper. Dette betyr at det var 21 oppgaver som ikke inneholdt noe form for programmering. Dette sammen med at det er en mindre variasjon i oppgavetypene styrker denne mistanken. Det kan også være en mulighet at tanken med dette Tema er og lære elevene Pytagoras gjennom å bruke programmering, men dette vanskelig å si helt sikkert fordi dette ligger utenfor hva som ble undersøkt i denne studien.

5.3 analysene sett oppimot hverandre

Det var noen veldig overaskende funn knyttet til hvilke oppgavetyper (Bråting & Kilhamn, 2022) som krevede hvilke kognitive krav (Stein & Smith, 1998). Dette var spesielt tilfelle når det kom til Feilsøkeoppgaver. Feilsøkeoppgaver er oppgaver som krever at elevene undersøker en kode og selv finner og retter feilene. Hvis man ser på PRIMM (Sentance et al, 2019) så går dette helt tydelig på investigate som skal legge til rette for selvstendig utforskning og problemløsning. Også hvis man ser på dette imot den algoritmiske tenkeren (Utdanningsdirektoratet, 2019) så innebærer disse oppgavene dekomposisjon, hvor elevene selv må dele opp koden og strukturere det for å få en god oversikt. alt dette peker mot at elevene selv må kunne klare å være selvstendige og ta reflekterte valg, altså vil man anta at dette vil være oppgaver med høye kognitive krav. Det var også delvis riktig for 4 av 9 oppgaver gikk under nivå 4, men det som var overaskende var at de 5 andre oppgavene alle gikk under nivå 1. altså krevde noen av disse oppgave kun lavere kognitive krav. Dette i seg selv er et veldig interessant funn, men det er vanskelig å si noe mer om dette uten å gjennomføre en dypere undersøkelse.

5.4 affordances og constraints

Det er tydelig at der er Affordances og Constraints (Humble & Mozelius, 2023; Gibson, 2014) i Kikora som har en betydelig påvirkning, både når det kommer til hvilke oppgaver vi møter og hvordan disse kan løses. Det er også noen tydelige utfordringer gjennom dette.

5.4.1 oppgaver som kan løses feil

som det ble påpekt i resultatene så kan enkelte oppgaver løses på feil måte eller med andre ord så kan Kikora finne på å godta feil svar. Dette kan ha helt tydelige utfordringer spesielt knyttet til at en elev kan finne på å tro at den har løst oppgaven riktig og forstått hvordan det fungerer, mens i realiteten så har eleven tilfeldigvis skrevet noe Kikora har godtatt. Heldigvis så virker det som at dette er svært usannsynlig da det ser ut til at de fleste oppgavene ikke godtar noen form for feilsvar. Det som derimot er interessant er at det er som nevnt enn viss logikk i hvilke oppgaver som godtar feil svar og ikke. Som forklart i resultatene hvis man klarer å se hva Kikora ser etter så kan man skrive inn nesten hva som helst som gir det utfallet. Det er ikke sikkert det er slik det fungerer, men det er det som gir mest mening ut ifra resultatene. Hvis det er slik det fungerer så kan en elev begynne å lete etter hva Kikora sjekker istedenfor hva oppgaven ber dem om å gjøre. Ved første øyekast vil man anta at dette er en negativ ting fordi eleven gjør ikke det som er ment. Men etter det jeg ser så må en elev ha en så god forståelse av programmering at de fleste elevene ikke vil kunne gjøre dette. Sett oppimot Blooms taksonomi (Adams, 2015) så vil dette kreve at eleven minst ligger på nivået analyse. Altså ut ifra det som er sett på tidligere så vil dette antagelig kreve et høyere nivå enn oppgavene gjør hvis de løses på den måten det er ment. Problemet oppstår eventuelt hvis en elev klarer dette og begynner å gi svaret videre, men det kan jo også skje til vanlig

5.4.2 manglende oppgavetyper

Kikora er et digitalt læreverk med både muligheter og begrensinger og dette kan ha påvirkninger på forskjellige ting. En mulighet eller Affordances (Humble & Mozelius, 2023; Gibson, 2014) er at det er et læreverk med ferdige oppgaver som retter seg selv. Dette gjør at det er et enkelt læreverk for lærere å ta i bruk, men det begrenser også hva lærerne kan gjøre med det. Fordi det retter seg selv så er det vanskelig å ha spørsmål hvor elevene svarer med setninger, fordi 10 elever kan skrive et svar og ordlegge seg forskjellig alle ti, men de kan samtidig alle ha svart riktig. Dette gjør at slik som oppgavetypene (Bråting & Kilhamn, 2022)

Forklare og Se for seg er vanskelig å få til i Kikora. Dette blir jo da en ganske klar begrensing som Kikora har. Det er også mulig at dette er en av grunnene til at det heller ikke er noen forme og skape oppgaver, men det er vanskelig å si noe på fordi forme å skape oppgaver vil gi et svar i form av kode og ikke tekst som forklare og se for seg oppgaver vil.

Forme og skape oppgaver definisjonsspørsmål?

Det er som sagt overaskende at det ikke var noen forme og skape oppgaver. En mulig grunn er at måten jeg definerte forme og skape oppgaver gjennom PRIMM (Sentance et al, 2019) begrenset veldig hvilke oppgaver som kunne falle innunder forme og skape. En god del av oppgavene kunne ha gått under forme og skape hvis jeg hadde definert det annerledes. Som nevnt var det mange oppgaver hvor man fikk oppgitt forskjellige kommandoer. Disse oppgavene ble som også nevnt plassert under finne regel fordi elevenes oppgave var i størst grad å finne ut rekkefølgen disse skulle plasseres i og ikke det at eleven skulle skape en egen kode hvor de i stor grad har eierskap til koden selv. Dette ville jo endret resultatet veldig hvis jeg hadde valgt og ikke gjøre den endringen, men det viser jo også at det ikke er noen oppgaver hvor elevene skaper koder fra bunden uten å allerede ha fått kommandoene tidligere.

En annen grunn til at det er få forme og skape oppgaver er at alle disse oppgavene er tydelig rettet mot et begynnernivå innen koding og programmering. Oppgavene introduserer mange grunnleggende kommandoer og fokuserer mye på hvordan tekstprogrammering fungerer. Det egentlig veldig naturlig med tanke på at det ene temaet heter fra blokk til tekstprogrammering. Fordi det er så på et begynnernivå så kan det være en grunn til at det ikke er noen forme og skape oppgaver. forme og skape oppgaver krever mer av eleven, spesielt når det kommer til hvordan programmering fungerer og hvordan man finner og bruker kommandoer. Det er mulig at forme og skape oppgaver er en oppgavetype (Bråting & Kilhamn, 2022) som krever for mye, altså at det automatisk gjør at oppgavene blir på et så høyt nivå at det ikke lenger passer vanskelighetsgraden man har tenkt at temaene skal ligge på. det er også viktig å presisere at dette hører til teks programmering spesifikt. Blokkprogrammering krever ikke at man må skrive alt riktig. i blokkprogrammering kan elevene bare dra de forskjellige blokkene fram og tilbake til de kobler seg sammen. Altså hadde det vært blokkprogrammering som var hovedfokuset så kunne det sett helt annerledes ut.

Jeg vil også påpeke at det er mulig det er forme og skape oppgaver i de andre temaene, som jeg ikke har sett på. Det er fult mulig det ikke var naturlig i de temaene jeg har sett på å ha forme og skape oppgaver. det er vanskelig å si noe om. For dette er også noe som ligger utenfor det denne studien har sett på

5.4.3 begrensede tilbakemeldinger

som nevnt er det svært begrenset hvilke tilbakemeldinger man kan få. Som sagt så gikk tilbakemeldingene på hvor vidt oppgaven var riktig eller ikke. Dette begrenser veldig hva elevene kan få ut av disse tilbakemeldingene. Som sagt tidligere hevder Kikora på sin framside (U.å.) at Kikora kan gi Formative tilbakemeldinger. Ut ifra det som er sett på i denne studien vil jeg påstå at dette ikke stemmer. Jeg vil tørre å påstå at alle tilbakemeldingene jeg har sett har vært summative ikke formative. For eksempel: to elever kommer til en lærer og begge har løst den samme oppgaven feil, men de har gjort to forskjellige feil og, dette har dermed ført til at de har fått to forskjellige feil svar. Dette vil bety at de få to forskjellige svar som baserer seg direkte på det de har gjort. Hvis de gjør det samme i Kikora, altså gjør to forskjellige feil på samme oppgave og har to forskjellige feilsvar, så vil de få den samme tilbakemeldingen, som baserer seg på hva Kikora antar en elev vil gjøre feil. Jeg vil derfor hevde at disse tilbakemeldingene ikke er formative, men det er ut fra den definisjonen jeg har brukt. Det er mulig Kikora har en annen definisjon som gjør at oppgavene har en formativ vurdering. Det er også mulig at det er andre oppgaver i Kikora som har Formative tilbakemeldinger, men at de jeg har sett på ikke har det. Det er umulig å si noe på uten å ha undersøkt alle oppgavene som er i Kikora.

6. konklusjon

konklusjonen i denne studien og dens problemstilling er i hovedsak at studien tar for seg for lite og sier kun noe om akkurat de temaene som har blitt sett på. Dette gjør det umulig å si noe konkret om hvordan programmeringsoppgavene på 9.trinn i Kikora generelt er. De eneste programmeringsoppgavene jeg kan si noe på er de fra de to temaene som er sett på. derfor er det ikke er noen ordentlig konklusjon, men jeg kommer allikevel til å besvare forskningsspørsmålene oppimot de temaene jeg har sett på. dette blir da det nærmeste denne studien kommer en besvarelse på oppgavens problemstilling.

6.1 Svar på Forskningspørsmål

Hvilke kognitive krav stilles av programmeringsoppgavene I Kikora

Det er en liten variasjon i de kognitive kravene (Stein & Smith, 1998) man møter, og det er ikke noen nivåer innen de kognitive kravene man ikke møter, men det er en helt klar overvekt med oppgaver som går under de lavere nivåene altså nivå 1 og 2 eller memorering og prosedyrer uten sammenhenger. Disse utgjør så mye som over 71,6 prosent av alle oppgavene i disse to temaene. På grunn av dette er ikke Kikora nokk alene da elever bør få flere oppgaver på de høyere nivåene. Oppgavene i Kikora kan begynne og danne et grunnlag eleven kan bygge videre på, men de er nødt til å få oppgaver utenfra i tillegg for å bygge videre på dette grunnlaget

Hvilke programmerings oppgavetyper finner man på Kikora

Av oppgavetyper (Bråting & Kilhamn, 2022) er det tre oppgavetyper som ikke er å finne. Dette er oppgavetypene Forme og skape, forklare og forestille seg. Ellers er alle de andre oppgavetypene å finne, og med en klar overvekt på følgeoppgavene. Dette betyr at det ikke er oppgaver hvor eleven får trene på og skape hypoteser for hva som kommer til å skje videre og eleven får heller ikke forklare med egne ord hvordan de har forstått kodene de ser på. det er heller ingen forme og skape oppgaver som gjør at elevene ikke får skape egne koder fra bunnen av.

Hvor stor forskjell er det mellom temaet Fra blokk til tekstprogrammering og temaet Programmere Pytagoras innen bruk av oppgavetyper og kognitive krav

Det er en helt klar forskjell mellom de to temaene. I forhold til de kognitive kravene (Stein & Smith, 1998) så har begge de to temaene flest oppgaver som går under de lavere nivåene, men Fra blokk til tekstprogrammering har et flertall innen nivå 2 altså prosedyrer uten sammenhenger og Programmere Pytagoras har flertall innen nivå 1 eller memorering. Når det kommer til oppgavetype (Bråting & Kilhamn, 2022) så har begge temaene flest oppgaver innen kategorien følge. Programmere Pytagoras har likevel en betydelig større andel oppgaver innen følge, og generelt få oppgaver innen de andre kategoriene, mens Fra blokk til tekstprogrammering har en jevnere fordeling med unntak av Faktaspørsmål der oppgaveantallet utgjør kun noen få prosenter. Dette tyder på at det er tydelige forskjeller i hva som er i fokus og hva som er målet med de to forskjellige temaene.

Hvilke kognitive krav stilles i de forskjellige programmeringsoppgavetypene i Kikora?

Av de forskjellige oppgavetypene (Bråting & Kilhamn, 2022) var det stor variasjon i hva som krevdes innen de forskjellige kognitive kravene (Stein & Smith, 1998). Følge var den oppgavetypen som gikk under flest av de kognitive kravene. Det var en helt klar størst andel innen nivå 2 og det var den eneste oppgavetypen som hadde oppgaver som gikk under nivå 0. finne regel fordelte seg ganske jevnt mellom de lavere og høyere kognitive kravene. Faktaspørsmål var den eneste oppgavetypen som kun gikk under et kognitivt krav altså nivå 1. Feilsøke var den oppgavetypen med mest overaskende fordeling som fordelte seg jevnt mellom nivå 1 og 4.

Hvilke affordances og constraints finnes i programmeringsoppgavene i Kikora?

Det var flere affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014) som tydelig har en påvirkning på oppgavene Kikora. En av disse er at enkelte oppgaver kan løses feil, altså kan man få Kikora til å si at en oppgave er løst riktig selv om det ikke er tilfelle. Dette i seg selv kan ses på som et problem, men det framstår veldig begrenset. Fordi hvis man konsekvent går for å løse oppgaver feil er man nødt til å ha en bedre forståelse av programmering enn det oppgavene krever. Altså hvis man tilfeldigvis løser en oppgave på feil måte så er det usannsynlig at det skjer igjen. Så man kan diskutere om det egentlig er et problem.

en annen affordance er at oppgavene er selvrettende. Dette er ressursmessig en kjempefordel ved at man kan spare masse tid på rette oppgaver og får dermed bedre tid til å for eksempel planlegge undervisning. En negativ side med dette er at man ikke like lett kan fange opp

misoppfatninger ved å se nøyaktig hva elevene har gjort feil.

en Constraint i Kikora er at tilbakemeldingene som gis er veldig begrenset og stort sett går på hvor vidt man har løst oppgaven riktig. dette er negativt fordi det er begrenset hvor mye en elev kan få ut av disse tilbakemeldingene, men det er da en konsekvens av å ha et læreverk som retter seg selv

en annen constraint er at det ikke kan erstatte dialogen mellom lærer og elev. Dette er fordi Kikora ikke kan like lett se hva en elev ikke har forstått. En lærer kan prate med en elev og gjennom samtale forstå hva som gjør at en elev trenger hjelp med.

7. Områder for videre forskning

Det er mange aspekter rundt denne studien som man ikke kan si noe om fordi det ligger utenfor det som er undersøkt. denne studien i seg selv sier kun noe om en liten brøkdel av oppgavene i Kikora. Hvis man skal få et ordentlig innblikk i Kikora sine oppgaver må det gjøres en større analyse hvor man ser på flere oppgaver. altså istedenfor å ta for seg 81 oppgaver slik som er gjort her så kunne man sett på nærmere 400 for å virkelig fått et innblikk, men dette vil kreve et større omfang enn det en masteroppgave kan klare å ta for seg. Så det er helt klart et potensiale for visere forskning

Et annet område for videre forskning er affordances og constraints (Humble & Mozelius, 2023; Gibson, 2014). Spesielt for å se nøyere på Kikora som digitalt læreverk. Dette går også for andre digitale læreverk. I denne studien er det kun sett på en liten del av hva som er i Kikora og det kan være interessant å se en grundigere studie på dette område

En mulig årsak til at det mangler skape-oppgaver er som sagt de digitale begrensningene i Kikora. AI-språkmodeller er noe det er mye prat om. For eksempel chatGPT (Openai, U.å.). Disse modellene er i stand til å svare på spørsmål og kan ha et kjempepotensial innen digitale oppgaver, så dette er også et område som ville kunnet nytte av videre forskning. Funnene i denne studien tydet på et behov for kreative oppgaver kan slike språkmodeller hjelpe med det?

8. Referanser

Adams, N. E. (2015). Bloom's taxonomy of cognitive learning objectives. Journal of the Medical Library Association, 103(3), 152–153. https://doi.org/10.3163/1536-5050.103.3.010

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

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

Christoffersen, L., & Johannessen, A. (2012). Forskningsmetode for lærerutdanningene (p. 178). Abstrakt forl.

Forskning.no. (2020, 11. Juli) *Programmering vil bli en utfordring for lærere:* hentet 13 september 2023 fra https://forskning.no/barn-og-ungdom-hogskolen-i-ostfold-matematikk/programmering-vil-bli-en-utfordring-for-laerere/1711838

Gibson, J. J. (2014). The Ecological Approach to Visual Perception. Psychology Press. https://doi.org/10.4324/9781315740218

Grover, S., & Pea, R. (2013). Computational Thinking in K—12: A Review of the State of the Field. Educational Researcher, 42(1), 38–43. https://doi.org/10.3102/0013189X12463051

Gyldendal. (U.å.) *Multi smartøving:* Hentet 13 september 2023 fra https://www.gyldendal.no/grunnskole/matematikk/multi-smart-oving/

Humble, N., & Mozelius, P. (2023). Making programming part of teachers' everyday life – Programming affordances and constraints for K-12 mathematics and technology. *The International Journal of Information and Learning Technology*, *40*(1), 98–112. https://doi.org/10.1108/IJILT-03-2022-0069

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

Kikora (U.å.) *Forside:* Hentet 12 september 2023 fra https://feide.kikora.no/

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

LærKidsaKoding. (U.å.) hentet 13 september 2023 fra https://www.kidsakoder.no/

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

Openai (U.å.) Hentet 14 september 2023 fra https://chat.openai.com/

Postholm, M. B., Jacobsen, D. I., & Søbstad, R. (2018). Forskningsmetode for masterstudenter i lærerutdanningen (p. 300). Cappelen Damm akademisk.

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

SNL (2021, 1. februar) *formativ vurdering:* hentet 4 september 2023 fra https://snl.no/formativ_vurdering

Stein, M. K., & Smith, M. S. (1998). Mathematical Tasks as a Framework for Reflection: From Research to Practice. Mathematics Teaching in the Middle School, 3(4), 268–275. https://doi.org/10.5951/MTMS.3.4.0268

Utdanningsdirektoratet. (2019). Algoritmisk tenkning. Hentet 3. september 2023 fra https://www.udir.no/kvalitet-og-kompetanse/digitalisering/algoritmisk-tenkning/

Utdanningsdirektoratet. (2023). Hvordan bruke læreplanene. Hentet 3. september 2023 fra https://www.udir.no/laring-og-trivsel/lareplanverket/stotte/hvordan-ta-i-bruk-lareplanen/#a153411

Alle bilder av oppgaver brukt i denne masteroppgaven tilhører Kikora. På grunn av måten Kikora er satt opp på er det ikke mulig å linke direkte til sidene hvor oppgavene ligger. Derfor har alle bildene tydelig forklaring på hvor de er hentet fra i bildeteksten

Kikora (U.å.) [Oppgave] https://feide.kikora.no/

9. Vedlegg

Her ligger et bilde av alle Excel-dokumentene til analysene for kognitive krav og oppgavetype. De mørke rutene i 9.4 eller oppgavetype for Programmere Pytagoras er de oppgavene som ikke ble analysert i analysen med oppgavetype

9.1 Excel-dokument kognitive krav Fra blokk til tekstprogrammering

oppgavenr	nivå 0	nivå 1	nivå 2	nivå 3	nivå 4	kikoragruppering
1			x			1,1
2				x		1,2
3			x			1,3
4				x		1,4
5		x				2,1
6		x				2,2
7			x			3,1
8				x		3,2
9			x			3,4
10			x			1,1
11				x		1,2
12			x			1,3
13				x		1,4
14			x			1,5
15				x		1,6
16			x			1,7
17					x	1,8
18			x			2,1
19		x				2,2
20			x			2,3
21			x			2,4
22			x			3,1
23			x			3,2
24			x			4,1
25	x					4,2
26			x			4,3
27		x				5,1
28			x			5,2
29			x			5,3
30		x				5,4
31					x	6

9.2 Excel-dokument kognitive krav Programmere Pytagoras

oppgavenr	nivå 0	nivå 1	nivå 2	nivå 3	nivå 4	kikoragruppering
1		x				1
2		x				2,1
3		x				2,2
4		x				3,1
5		x				3,2
6		x				3,3
7			x			4,1
8			x			4,2
9			x			5,2
10		x				6,1
11		x				6,2
12		x				7,1
13				x		7,2
14		x				7,3
15				x		7,4
16		x				8,1
17	x					8,2
18			x			8,3
19		x				9
20		x				1,1
21		x				1,2
22		x				2,1
23		x				2,2
24		x				3,1
25		x				3,2
26		x				3,3
27			x			4,1
28			x			4,2
29			x			4,3
30					x	4,4
31					x	4,5
32		x			x	5,1
33			x			5,2
34			x			5,3
35	x					5,4
36		x				5,5
37					x	5,6
38		x				6,1
39		x				6,2
40		x			x	6,3
41		x				6,4
42		x			x	6,5
43		x				7,1
44	x					7,2
45			x			7,3
46					x	7,4
47		x				7,5
48					x	7,6
49			x			8,1
50					x	8,2

9.3 Excel-dokument oppgavetype Fra blokk til tekstprogrammering

oppgavenr	følge	finne regel	feilsøke	forme og skape	forklare	forestille seg	faktaspørsmål	kikoragruppering
1	x							1,1
2	x							1,2
3	x							1,3
4	x							1,4
5			x					2,1
6			x					2,2
7		x						3,1
8	x							3,2
9	x							3,4
10		x						1,1
11		x						1,2
12		x						1,3
13		x						1,4
14		x						1,5
15		x						1,6
16		x						1,7
17			x					1,8
18	x							2,1
19			x					2,2
20		x						2,3
21		x						2,4
22	x							3,1
23		x						3,2
24	x							4,1
25	x							4,2
26	x							4,3
27							x	5,1
28	x							5,2
29	x							5,3
30			x					5,4
31			x					6

9.4 Excel-dokument oppgavetype Programmere Pytagoras

oppgavenr	følge	finne regel	feilsøke	forme og skape	forklare	forestille seg	faktaspørsmål	kikoragruppering
1								1
2								2,1
3								2,2
4								3,1
5								3,2
6								3,3
7	x							4,1
8	x							4,2
9	x							5,2
10							x	6,1
11							x	6,2
12							x	7,1
13	x							7,2
14							x	7,3
15	x							7,4
16							x	8,1
17	x							8,2
18	x							8,3
19			x					9
20								1,1
21								1,2
22								2,1
23								2,2
24								3,1
25								3,2
26								3,3
27	x							4,1
28	x							4,2
29	x							4,3
30			x					4,4
31			x					4,5
32								5,1
33	x							5,2
34	x							5,3
35	x							5,4
36								5,5
37	x							5,6
38								6,1
39								6,2
40								6,3
41								6,4
42								6,5
43								7,1
44	x							7,2
45	x							7,3
46		x						7,4
47							x	7,5
48		x						7,6
49	x							8,1
50		x						8,2