Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

Marius Føske Danielsen

Introduksjon av tekstbasert programmering i matematikk 1T forankret i matematikk

Test og videreutvikling av et undervisningsopplegg gjennom prakseologisk analyse og didaktisk ingeniørvirksomhet

Masteroppgave i matematikkdidaktikk
Veileder: Vegard Topphol
Juni 2024

NTNU
Kunnskap for en bedre verden

Marius Føske Danielsen

Introduksjon av tekstbasert programmering i matematikk 1T forankret i matematikk

Test og videreutvikling av et undervisningsopplegg gjennom prakseologisk analyse og didaktisk ingeniørvirksomhet

Masteroppgave i matematikkdidaktikk
Veileder: Vegard Topphol
Juni 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

NTNU
Kunnskap for en bedre verden

Sammendrag

Denne studien har som utgangspunkt utvikling, realisering og videreutvikling av undervisningsopplegg innenfor introduksjon av programmering i matematikk 1T. Gjennom analyse av utviklingsprosessen og realiseringen ved bruk av didaktisk ingeniørvirksomhet (Strømskag, 2020a) og ATD - den antropologiske teorien for det didaktisk (Chevallard & Bosch, 2020a) - etableres grunnlaget for videreutvikling av undervisningsopplegget basert på datamateriale i form av intervjuer og observasjoner. Innenfor rammeverket ATD tas blant annet *praksologi*, *didaktiske transposisjoner* og *det å stille spørsmål til verden* i bruk, hvorpå *prakseologi* har størst rolle i analyse og kategorisering av datamateriale fra både utvikling og intervjuer.

Et viktig poeng å understreke i denne studien er ønsket om å ha en matematisk forankring i introduksjonen av tekstbasert programmering. Realiseringen har funnet sted i en 1T klasse, og fra samme klasse finner vi deltakerne som tok del i fokusgruppe intervjuer i etterkant av gjennomføringen.

Studien min viser blant annet at det å ha et veldig åpent undervisningsopplegg, selv innenfor et tema som programmering i matematikk med tilhørende mange løsningsmetoder, kan virke mot sin hensikt. I tillegg har jeg sett at mange elever gir uttrykk for at forkunnskapene de sitter med gjennom blokkprogrammering føles lite anvendbare i overgangen til tekstbasert programmering. En sentral del i videreutviklingen ble derfor å fjerne noe av åpenheten, uten å ta bort muligheten for videre utforskning og heller inkludere flere løsningsmetoder gjennom spesifikke oppgaver. Overgangen mellom blokk- og tekstbasert programmering fikk også en større rolle i videreutviklingen enn hva jeg originalt hadde sett for meg, men basert på intervjuene var dette utvilsomt et viktig element å løfte fram.

Abstract

This study has as its starting point the development, realization and further development of a teaching plan within the introduction of programming in mathematics 1T. Through analysis of the development process and realization using didactic engineering (Strømskag, 2020a) and ATD - the anthropological theory for the didactics (Chevallard & Bosch, 2020a) - the basis for further development of the teaching program is established based on data material in the form of interviews and observations. Within the ATD framework, among other things, *praxeology*, *didactic transpositions* and *the paradigm of questioning the world* are used, whereupon *praxeology* has the greatest role in the analysis and categorization of data material from both development and interviews.

An important point to emphasize in this study is the desire to have a mathematical foundation in the introduction of text-based programming. The realization took place in a first year high-school class, and from the same group of pupils we find the participants who took part in focus group interviews after the implementation.

My study shows, among other things, that having a very open teaching plan, even within a topic such as programming in mathematics and its many possible solutions, can be counterproductive. In addition, I have seen that many students express that the prior knowledge they have through block programming feels of little use in the transition to text-based programming. A central part of the further development was therefore to remove some of the openness, without taking away the possibility of further exploration and rather including extra methods for solving the same problem through specific tasks. The transition between block and text-based programming also played a bigger role in the further development than I had originally envisioned, but based on the interviews this was undoubtedly an important element to highlight.

Forord

Denne masteroppgaven innen matematikkdidaktikk markerer slutten av min tid som student. Det har vært et lærerikt og variert studieløp, jeg har trivdes veldig godt med. Jeg ser nå fram til å tre inn i yrkeslivet, og ta i bruk den faglige, pedagogiske og didaktiske kompetansen jeg sitter igjen med etter endt lektorutdanning i realfag ved NTNU.

Jeg ønsker å takke min veileder, Vegard Topphol, for gode diskusjoner, nyttige tilbakemeldinger og et generelt trivelig samarbeid. Videre ønsker jeg å takke min mor for gode matematikkdidaktiske diskusjoner, samt skole og faglærer som lot meg teste ut undervisningsopplegg og intervjue deltakere til studien.

Avslutningsvis ønsker jeg også å takke samboer, medstudenter og venner for en flott studietid i Trondheim by!

Trondheim 30. Mai 2024

Marius Føske Danielsen

Innhold

Kapittel 1

Introduksjon

1.1 Bakgrunn for studien

Til tross for at vi nå befinner oss i 2024 er programmering som en del av fellesfagene på videregående, og ungdomsskole, fortsatt relativt nytt. Inntoget gjennom Kunnskapsløftet i 2020 kan en utvilsomt si har skapt muligheter, glede og frustrasjon. Hovedfokus i denne studien er dog ikke å vurdere programmering som en del av læreplanen, men heller hvordan introdusere og ikke minst relatere programmering i matematikk. Dette skal skje gjennom vurdering og videreutvikling av et interaktivt undervisningsopplegg, ment som introduksjon av tekstbasert programmering i matematikk 1T, og senere et hjelpemiddel inn mot repetisjon, videre studier av programmering, og lignende. I denne studien er det hovedsakelig hvordan elever fra en 1T klasse opplever undervisningsopplegget som er fokus, mens repetisjonselementet heller er et utestet ønske med opplegget. Vurdering og videreutvikling tar i bruk teoretiske rammeverket antropologisk teori for det didaktiske eller ATD (Chevallard, 2019), samt analyseverktøyene prakseologi og didaktisk ingeniørvirksomhet. Med dette rammeverket blir det også naturlig å se på de didaktiske transposisjonene eller endringer og forenklinger originalkunnskapen har gjennomgått fra akademisk nivå til det en finner i skolen som institusjon (Bosch & Gascón, 2014). Ettersom jeg var ferdig med videregående opplæring i 2016 opplevde jeg aldri LK20, og ikke hadde programmering på grunnskolen. For meg gjør dette transposisjonsprosessene og temaet ekstra interessante, da jeg i utgangspunktet har lært programmering på universitetsnivå, og kun undervist dette temaet på videregående skole.

Ønsket om å gjennomføre denne studien kommer både fra interesse og praksis. Interesse i form av et ønske om å skape en så god læringssituasjon som mulig innenfor matematikk, og programmering har jeg selv opplevd som veldig nyttig, visuelt og med få begrensninger tilknyttet hvor langt en kan gå. Denne interessen og nytteverdien er noe jeg vil skal komme fram i klasserommet gjennom undervisningsopplegget som skal være produktet av denne studien. Praksisdelen av motivasjonen for denne studien bygger på egne erfaringer fra praksis, samtaler med andre mattelærere, og vikariat hvor jeg har oppdaget at det er veldig stor variasjon i hvordan og hva som gjennomføres og vektlegges innenfor programmering. Da dette er en matematikkdidaktisk studie, og faget programmering er lagt hovedsakelig til

matematikk – spesielt grunnleggende opplæring – er det et sterkt ønske fra min side å kunne relatere programmeringen til matematikk.

Undervisningsopplegget som ble testet ut i denne studien ble utviklet i forbindelse med faget semesteroppgave i RFEL3100 ved NTNU (Danielsen, 2023). I den oppgaven var riktig nok ikke fokuset på å teste det ut i klasserommet, men diskusjon med medstudenter som har hatt programmering i videregående gjennom intervjuer. Etter endt semesteroppgave ble det gjennomført noen få endringer før jeg i januar 2024 var ute i en matematikk 1T klasse og testet det diskuterte undervisningsopplegget. Gjennom analyse av utvikling, gjennomføring og begrensninger tilknyttet opplegget ved bruk av didaktisk ingeniørvirksomhet sikter jeg meg inn på videreutvikling med forankring i ATD og de to innledende fasene av didaktisk ingeniørvirksomhet (Strømskag, 2020a).

1.2 Tidligere forskning

Av tidligere forskning er det gjennomført en god del studier på både sammenhengen og overgangen mellom blokk- og tekstbasert programmering, i tillegg til relasjonen mellom matematikk og programmering. Jeg skal i dette delkapittelet omtale nøkkelelementer fra fire artikler som er inne på en del av de samme temaene som omtales i denne studien.

Det er ikke alltid sammenhengen mellom blokk- og tekstbasert programmering er så tydelig for elever, og i artikkel «To block or not to block, that is the question» (Weintrop & Wilensky, 2015) ser de på hva som gjør at blokkprogrammering er «enkelt» å bruke, samt hvilke forskjeller elever i videregående skole opplever i overgangen til tekstbasert programmering. I tillegg har jeg sett på en studie som oppsummerer 15 år med introduksjon av programmering i skolen (Szabo et al., 2019), hvor de blant annet ser på plasseringen av de ulike programmeringsspråkene og metodene tilknyttet klassetrinn i ulike land. Det er verdt å nevne at Norge, eller studier som omhandler programmering i norsk skole, ikke omtales her, men at blant annet Sverige og Finland omtales. Et av de interessante momentene fra denne studien er hvordan plasseringen av spesielt «Scratch» og «Python» basert på klassetrinn, sammenfaller med modellen som virker å være implementert i Norge (Szabo et al., 2019, s. 4-5).

De to siste artiklene jeg skal nevne eksplisitt her, er studier som foregår på lavere trinn enn det som er hovedfokus i denne studien. «Bridgeing primary programming and mathematics» (Benton et al., 2017) har søkelyset på aldersgruppen 9-11 år, med blokkprogrammering som teknikk. Det er dog fem «E'er» som presenteres i forbindelse med programmering og etablering av en bro («bridgE» - en av «E'ene») mellom programmering og matematikk som

kan anvendes og oppleves som like relevant også senere i utdanningsløpet. Den siste artikkelen er en case-studie fra universitet i Østfold, som ser på læring av programmering og matematikk i svensk grunnskole (Stigberg & Stigberg, 2020). Denne studien er tredelt i den forstand at de ser på hvordan programmering undervises i matematikkundervisningen, hvilke utfordringer lærerne opplever tilknyttet programmering innenfor matematikk, og elevers forståelse av programmering og dens relasjon til matematikk.

1.3 Problemstilling og forskningsspørsmål

I denne studien har hovedfokus vært å produsere, teste ut, analysere og videreutvikle et undervisningsopplegg med ønske om å kunne relatere introduksjon av tekstbasert programmering i matematikk 1T til det matematiske innholdet i faget. Dette ønsket med tilhørende produksjons- og analysefase er oppsummert gjennom en overordnet problemstilling og to forskningsspørsmål som tar tak i de tre ulike delene av studiene – nemlig *utvikling*, *testing/realisering* og *videreutvikling*. Ettersom analyse er en sentral del av samtlige deler av studien, vil hver av dem havne under lupen med utgangspunkt i prakseologi (Chevallard, 2006) og didaktisk ingeniørvirksomhet (Strømskag, 2020a) som analyseverktøy. Den overordnede problemstillingen lyder som følger:

> *Hvordan introdusere tekstbasert programmering i 1T forankret i matematikk?*

Tilhørende problemstillingen er de to forskningsspørsmålene, og sammen legger de grunnlaget for diskusjon av studien i kapittel 7:

(1) Hvordan har jeg utviklet, testet og samlet inn data tilknyttet undervisningsopplegget om programmering innenfor matematikk, og hvordan opplevde elevene dette undervisningsopplegget?

(2) Hvordan kan en videreutvikle undervisningsopplegget basert på analyse av realisering og datainnsamling?

1.4 Profesjonsrelevans og bærekraft

Etter kunnskapsløftet i 2020, og innføring LK20 er programmering blitt en større del av matematikkundervisningen på både ungdoms- og videregående skole. I denne studien er hovedfokus på matematikk 1T med tilhørende kompetansemål (Utdanningsdirektoratet, 2020):

> *«Formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsingsstrategier, digitale verktøy og programmering.»*

Programmering er med andre ord blitt en sentral del av profesjonen lektor i matematikk for 8.-13. trinn. Det handler dog ikke bare om å skulle kunne dekke kompetansemålene, men en sentral del av det å være en profesjonsutøver er å holde seg oppdatert på samtiden og ikke minst slik det kommer fram av den overordnede delen av læreplan; «*Faglig dømmekraft forutsetter også jevnlig oppdatering» (Kunnskapsdepartementet, 2017, s. 18)*.

En viktig del av bærekraftig utvikling som presentert i overordnet del av læreplan er teknologi (Kunnskapsdepartementet, 2017, s. 14). Programmering faller utvilsomt innenfor denne kategorien av viktige aspekter ved bærekraftig utvikling. Spesielt blir teknologisk utvikling, samt kompetanse innenfor sammenhengene mellom denne utviklingen og de sosiale, økonomiske og miljømessige elementene av bærekraftig utvikling dratt fram som en essensiell del av tverrfaglige utdanningen skolen og du som lærer skal kunne tilby (Kunnskapsdepartementet, 2017, s. 13-14).

Hva angår FN sine bærekraftsmål anser jeg spesielt mål fire og ni, nemlig *god utdanning* og *industri, innovasjon og infrastruktur* som spesielt relaterbare innenfor denne studien (FN, 2023). Gitt det mer og mer digitaliserte samfunnet vi lever i, er det å opparbeide seg kunnskap om programmering, i dette tilfellet gjennom matematiske anvendelser, utvilsomt en nyttig egenskap for kommende generasjoner i arbeidsliv og etter hvert som generasjonene får en større rolle i samfunnsutviklingen. Spesielt nevnes digitale ferdigheter i FNs rapport (FN, 2023, s. 21) innenfor *god utdanning*, mens under *industri, innovasjon og infrastruktur* blir vedlikehold av vekstvilkår for teknologiske bedrifter, samt tilgang på internett over hele verden påpekt som viktige punkter for videre utvikling (FN, 2023, s. 31).

1.5 Oppbygning

Studien presenteres gjennom en nokså ordinær struktur, hvor jeg introduksjonsvis skal presentere det teoretiske rammeverket i kapittel 2. Innenfor dette rammeverket finner vi den

antropologiske teorien for det didaktiske (ATD), med tilhørende sentralt innhold som å stille spørsmål til verden, prakseologier og didaktiske transposisjoner. Videre ser jeg på teori for den didaktiske situasjonen for å legge grunnlaget for presentasjon av didaktisk ingeniørvirksomhet. Kapittel 3 omhandler metodologi, gjennom prinsipper for forskning, datamateriale, grunnlag for videreutvikling, utvalg og etikk. Videre fra metodologi følgerr analyse av undervisningsopplegget, kapittel 4, fra start (utvikling) til slutt (begrensninger og validitetsanalyse). Etter analyse av undervisningsopplegget kommer analyse av hovedkilden til datamateriale, nemlig intervjuene som ble gjennomført i kapittel 5. Med analyse av både undervisningsopplegg og data på plass, går jeg videre til hovedmålet med studien, å videreutvikle det utprøvde undervisningsopplegget i kapittel 6 før diskusjon og konklusjon i kapittel 7 og 8.

Kapittel 2

Teoretisk rammeverk

Kapittel to omhandler det teoretiske rammeverket for denne oppgaven. Antropologisk teori for det didaktiske (2.1) med tilhørende underdeler i form av å stille spørsmål til verden (2.2), prakseologi (2.3) og didaktiske transposisjoner (2.4) er utgangspunktet for dette rammeverket. I tillegg introduseres didaktisk ingeniørvirksomhet (2.6) gjennom å først se på teori for den didaktiske situasjon (2.5), som et ekstra analyseverktøy i forbindelse med design og realisering av et undervisningsopplegg.

2.1 Antropologisk teori for det didaktiske

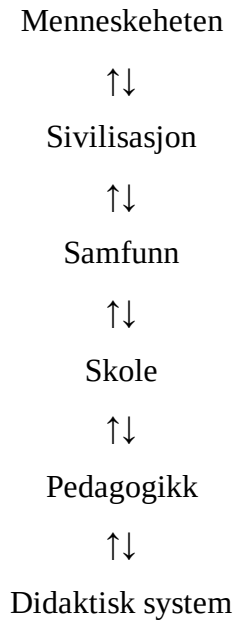
Det teoretiske rammeverket for denne oppgaven kalles antropologisk teori for det didaktiske – eller «antropologichal theory of the didactics» (heretter ATD). Selve rammeverket er utviklet hovedsakelig av Yves Chevallard, og ble først omtalt på 1980-tallet, med utgangspunkt i det å kunne studere matematikkdidaktiske fenomen, og spesielt de didaktiske transposisjonene som foregikk fra for eksempel universitet til videregående skole (Chevallard & Bosch, 2020b). I senere tid har ATD blitt videreutviklet og etablert seg som et rammeverk en kan bruke også utenfor matematikken, for eksempel gjennom generell kategorisering av menneskelige handlinger ved bruk av prakseologier. Et overordnet mål med ATD er å kunne ta tak i og se didaktiske situasjoner i så mange læringssituasjoner som mulig, ikke bare i skolen, men også i samfunnet generelt (Chevallard & Bosch, 2020a).

ATD bygger blant annet på det en omtaler som didaktiske situasjoner, gjennom didaktisk innhold/arbeid (*didactic stake*) og didaktiske handlinger (*didactic gesture*). De to kan forstås som henholdsvis hva som skal studeres og hvilke handlinger som er planlagt eller utført for å hjelpe i denne situasjonen (Chevallard, 2019). Videre kan en samle situasjonen og dens innhold og handlinger i et system – kalt det didaktiske systemet. Notasjonen for et slik system i ATD er $S(X, Y, \kappa)$ hvor S er selve systemet, X er de elevene, studentene eller bare menneskene generelt som skal studere eller lære noe om det didaktiske innholdet. Y er i den sammenheng hjelperne i det didaktiske systemet, det vil si hovedkilden til de didaktiske handlingene, og vil i en tradisjonell skolesituasjon typisk kunne være en eller flere lærere. Siste bokstavnotasjon er κ, selve kunnskapen som er målet for det didaktiske systemet ønsker å nå (Chevallard, 2015). Søken etter å belyse denne kunnskapen er noe av det som kjennetegner didaktikk som begrep innenfor ATD. Kunnskap i seg selv omtales veldig bredt,

og didaktikk som en vitenskap burde ha som mål å bevisstgjøre den studerende deltaker på alle stegene og transposisjonene kunnskapen har vært gjennom før den når mottaker – og på den måten åpne for en videre og dypere forståelse (Bosch Casabo, 2018). I denne søken for en dypere forståelse gjennom belysning av transposisjoner og bevisstgjøring på hvilken måte en kan skape videre engasjement dukker også begrensninger (*constraints*) og betingelser (*conditions*) opp som naturlige faktorer. Innenfor ATD omtales disse som hemmende faktorer for den didaktiske situasjonen, enten for en lærende person eller institusjon dersom det er noe som ikke kan endres innenfor de rammene og den tiden en har til rådighet (Chevallard, 2015). Et eksempel som ofte brukes når det er snakk om slike begrensninger er blant annet kompetansemål tilknyttet læreplaner. Grunnen til at disse i ATD sammenheng blir en begrensning er at de tidvis inneholder klare holdepunkter på når en elev har lært «nok», mens en i ATD ønsker og la elevene komme nærmere originalkunnskapen (Bosch & Gascón, 2014). Hvorvidt noe er en begrensning eller betingelse handler hovedsakelig om hvilket nivå innenfor for eksempel didaktisk medbestemmelse (se figur 2.1) en befinner seg, da kompetansemål ikke kan endres på skolenivå, mens på samfunnsnivå vil det være en betingelse ettersom det er her disse etableres.

Begrensninger og betingelser knyttes opp mot det didaktiske systemet i ATD gjennom seks ulike nivåer for didaktisk medbestemmelse. Blant disse seks nivåene finner vi det didaktiske systemet på bunn, mens menneskeheten er det øverste. Nedover fra menneskehet følger sivilisasjon, samfunn, skole og pedagogikk, før nevnte didaktisk system.

Figur 2.1



Figur 2.1: De ulike nivåene av didaktisk medbestemmelse i henhold til Bosch Casabo (2018, s. 4037), oversatt fra engelsk av meg.

Disse nivåene i ATD er både en presentasjon av hvor begrensninger kan komme fra, og hvorfor de kan være så vanskelige å skulle unngås eller endres. Det er i tillegg en annerkjennelse av at didaktiske systemer, pedagogikk og helt opp til skoler vil inneholde betingelser og forskjeller avhengig av samfunnet en som didaktiker opererer i (Chevallard, 2019). Et klart eksempel her vil være forskjellene mellom grunnskoleutdanning og hvor tidlig valg i retning arbeidsliv tas i land som Tyskland kontra Norge.

2.2 Stille spørsmål til verden

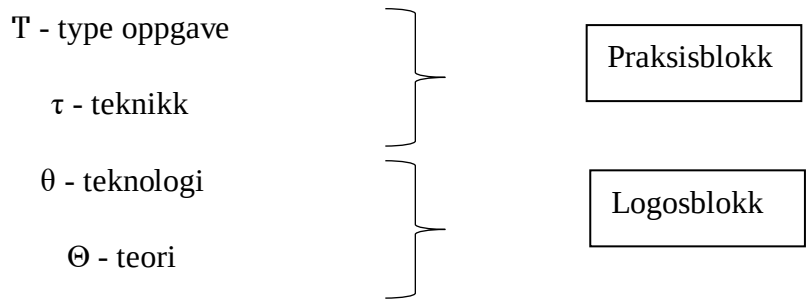
Et sentralt poeng i ATD er å bevege seg vekk fra det som omtales som det gamle paradigmet, eller det å besøke arbeider. En mye brukt metafor innenfor ATD og det gamle paradigmet er at kunnskap omtales som monumenter. Et av hovedargumentene for at en skal tre inn i et nytt paradigme handler om forståelsen av disse monumentene, og at istedenfor å besøke arbeider skal en studere disse. Elever i det gamle paradigmet er dessuten forventet å beundre monumentene de besøker – uten egentlig å ha grunnlag for å forstå den kunnskapen de faktisk observerer (Chevallard, 2015). Læreren blir i denne sammenheng omtalt som en guide, som hjelper elevene fra monument til monument, mens en i ATD ønsker å stoppe opp og ta seg tid til spørsmålene, hvorfor er den her eller hvordan oppsto denne? Disse spørsmålene kan kanskje virke underlige i en didaktisk sammenheng, men med monument metaforen frisk i minne, kan en tenke på disse spørsmålene som en form for dybdelæring. Et av

hovedargumentene som blir brukt mot nettopp det å besøke monumenter er at det i for stor grad kommer med en forventning om akkurat hva en skal lære. Det forventes at lærer eller guide presenterer et monument, og når denne presentasjonen er ferdig har en som elev lært det en skal om den presenterte kunnskapen. Sentralt i ATD er derfor tanken om at elever må studere κ selv, samt under lærerens veiledning, slik at endelig kunnskap faktisk avhenger av eleven sin tilnærming til læringsprosessen. Det handler altså i større grad om å skape en undersøkende læringskultur, hvor en åpner opp for spørsmål og dypere forståelse av monumentene, istedenfor å bare skulle nyte og beundre dem slik de fremstilles av en enkeltperson eller i litteratur (Chevallard, 2015).

2.3 Prakseologi

Tilegnelse, distribusjon og produksjon av kunnskap omtales i ATD sammenheng som en helt vanlig menneskelig aktivitet. ATD foreslår derfor en generalisering av disse aktivitetene kalt prakseologier (Bosch & Gascón, 2014). Chevallard peker på prakseologi som en grunnenhet for analyse av menneskelige handlinger, samt et verktøy for modellering av all type kunnskap, med de to grunnstenene praksis og logos (Chevallard, 2006). De to blokkene kan forstås som kunnskap om hvordan noe gjøres (*Praksis*) og hvorfor det gjøres på denne måten eller hvorfor noe fungerer (*Logos*). Praksis og logos kan igjen deles inn i to underdeler, nemlig type oppgaver (T) og teknikker (τ) for praksis, samt teknologi (θ) og teori (Θ) for logos (Bosch & Gascón, 2014). Settet [T, τ, θ, Θ] står sentralt i videre bruk av prakseologi som modell for analyse, og jeg vil derfor ta for meg hver enkelt komponent i noe større grad.

Figur 2.2



I praksisdelen av prakseologi finner vi nevnte T og τ. Type oppgaver kan her forstås på flere måter, men i denne studien vil den tolkes som gjentagende oppgaver innenfor en institusjon med en gitt metode for å løses. T trenger ikke nødvendigvis å være noe smått, og en kan enda mer generelt si at en aktivitet kan deles opp i spesifikke små oppgaver t, som igjen er en del av T. Den nevnte løsningsmetoden eller teknikken (*techné*) er det som omtales som τ. I ATD handler forståelsen av hva en type oppgave kan være, om at enhver handling eller oppgave

som krever en form for teknikk i utførelsen kan kategoriseres innenfor praksis blokken. Med andre ord vil enhver menneskelig handling med et mål, kunne kategoriseres ved bruk av de to «T-ene». Alt fra det å regne ut nærmest trivielle matteoppgaver til høyere ordens differensiallikninger, samt noe så hverdagslig som å ta ut av oppvaskmaskinen anses som slike handlinger innen ATD (Chevallard, 2019). Ser en enda mer spesifikt på praksisblokken i en typisk matematisk oppgave eller handling, vil T i matematikk 1T sammenheng for eksempel kunne være at elevene blir gitt en funksjon, $f(x)$, og blir bedt om å finne $f(3)$. Dette er en type oppgave de har sett og gjort mange ganger før, og det vil sannsynligvis være en rimelig standardisert, eller institusjonalisert, måte denne løses på. Hvis en derimot gir elevene en graf uten å spesifisere den tilhørende funksjonen kan en åpne opp for ulike teknikker (τ). De fleste elever er sannsynligvis relativt institusjonaliserte her også, men en teknikk vil være en grafisk løsningsmetode, samtidig som de kan ta en mer algebraisk tilnærming og finne utrykket for funksjonen.

Logos blokken forbindes med det folk flest vil kalle kunnskap (Chevallard, 2019), og har begrunnelse som et fellestrekk, i form av at teknologi legitimerer og begrunner teknikken, mens teori omhandler legitimering og nytteverdi tilknyttet spørsmål som hvorfor en gjør en handling eller oppgave, samt hvorfor prakseologien i seg selv gir mening. Teknologidelen av logos forstås som hvordan og hvorfor en teknikk fungerer, og i prakseologi vil det derfor være en klar kobling mellom τ og θ i den forstand at en kan velge en teknikk på bakgrunn av sin kunnskap om teknologien, slik terminologien forstås innenfor rammeverket. Teoridelen kan i den sammenheng forstås som grunnlag og støtte for en teknologisk forståelse, i og med at Θ skal underbygge drøftingen og valg av hvilke teknikker som løftes fram (Bosch & Gascón, 2014). Ser en tilbake på eksempelet som avrundet avsnittet om praksisblokken, kan en si at teknologidelen av logosblokken er valget mellom en grafisk og en algebraisk løsningsmetode. Hvis en utvider eksempelet og sier at deloppgave (a) er å finne $f(3)$ for en gitt graf, mens i oppgave (b) snur en på det og ber de finne $f(x) = 3$, før en avslutningsvis i en tenkt deloppgave (c) ber om nullpunktene til $f(x)$. Valget av teknikk gjennom teknologidelen kan da avhenge av hvorvidt grafen inneholder de ønskede verdiene, ekstraopplysninger som polynomgrad og lignende. Teoridelen kommer så inn som nevnt støtte for teknologi, eksempelvis dersom en kan løse (a) grafisk, mens (b) og (c) krever informasjon som ligger utenfor det grafiske utsnittet som er gitt. Med andre ord gjennom et grunnlag (teori) kan en argumentere for en algebraisk løsningsmetode allerede i første deloppgave (teknologi) for å velge riktig teknikk for å komme i gang med (b) og (c).

Avslutningsvis innenfor prakseologi skal det etableres et skille mellom spesielt begrepene teknikk og «didaktisk teknikk», hvor hver av disse tilhører prakseologier, men med ulik hovedperson – hvor sistnevnte er teknikker underviser bruker for å støtte læringen av et tema. Skille går mellom det jeg har valgt å kalle «didaktisk teknikk» og den mer generelle definisjonen av teknikk som presentert av Chevallard (2019). Teknikker med eleven som hovedkarakter, innebefatter de mer klassiske teknikkene en ser i et klasserom, som løsningsmetoder og koblinger mellom kjente strategier og den relativt nye programmeringen. Didaktisk teknikk har læreren og undervisningssituasjonen som hovedkarakter, i form av at det er teknikker som i større grad går på hvordan den didaktiske situasjonen kunne vært gjort annerledes i retrospekt, eller hvilke grep en kan ta underveis eller i forkant for å opprettholde engasjement, vise nytteverdi, og lignende (Gueudet et al., 2022). Noe mer generelt omtales en didaktisk prakseologi som arbeid mellom en kunnskapsinstitusjon og en didaktisk institusjon, med fokus på hva som skal tas med fra kunnskapsinstitusjonen og hva som beholdes i den didaktiske institusjonen (Artaud, 2019). Spesielt omtales didaktisk medbestemmelse (Bosch Casabo, 2018) i arbeidet med disse institusjonene, men noe mer utradisjonelt kan en også trekke paralleller mot didaktiske transposisjoner (Bosch & Gascón, 2014).

2.4 Didaktiske transposisjoner

En didaktisk transposisjon defineres som hvilke endringer kunnskap går igjennom fra dens originale form, og helt fram til den til slutt undervises på en bestemt institusjon (Chevallard & Bosch, 2020b). Transposisjonsprosessene deles tradisjonelt inn i fire ulike nivåer av kunnskap med tilhørende institusjoner. Øverste nivå, og det nærmeste en kommer kunnskapen sin originale form, er den akademiske kunnskapen. Relevante institusjoner for denne formen for kunnskap er universitet, forskningsinstitusjoner og andre steder der akademikere jobber med kunnskap så nært dens opprinnelse som mulig (Chevallard & Bosch, 2020b). Det er her viktig å være klar over at det kan være transposisjonsprosesser denne formen for kunnskap har vært gjennom også, men steget fra kunnskapen sin opprinnelse til akademisk kunnskap har ikke en egen plass i kategoriseringen. Eksempelvis vil rammebetingelser og antagelser om teoretiske systemer kunne tolkes som en didaktisk transposisjon fra kunnskapen sin mest generelle form, og allikevel regnes som akademisk kunnskap.

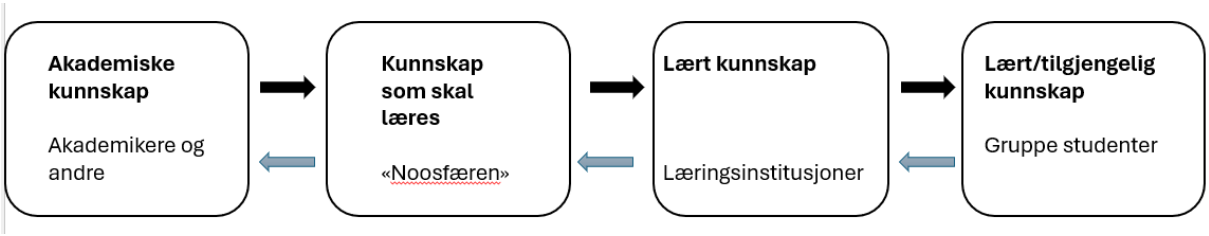
«Noosfæren» er det nest øverste nivået i denne modellen, og kan beskrives ved at det er her en finner lærere, pensumforfattere, læreplanutviklere og lignende. Denne formen for kunnskap har fått navnet *kunnskap som skal læres*, hvor «noos» da oversettes til å tenke – en befinner seg altså i sfæren med de som tenker på kunnskapen som skal læres, med læreplaner

og lærebøker som eksempler på produkter som stammer fra «noosfæren» (Bosch & Gascón, 2014). Det er i denne transposisjonsprosessen sentralt at relasjonen mellom akademisk kunnskap og neste nivå *undervist kunnskap* opprettholdes, for å ikke fjerne koblingen og potensialet for videre arbeid med kunnskapen. Chevallard og Bosch (2020b) peker her på viktigheten av å skape en autentisk illusjon, hvor en unngår at transposisjonsprosessene kommer tydelig fram.

Det neste nivået befinner seg på det som omtales som læringsinstitusjoner, og inneholder nevnt *undervist kunnskap*. Kunnskapen har nå vært gjennom to, potensielt tre (Chevallard & Bosch, 2020b), didaktiske transposisjonsprosesser fra dens originale form. Det er denne kunnskapen som presenteres og/eller studeres i en læringssituasjon. Den siste transposisjonen skjer mellom *undervist* og *lært kunnskap*. I den sammenheng er studentene eller elevene ansett som institusjonen, og ettersom en kan anse lærebøker og andre eksterne ressurser som en del av læringsinstitusjonen utviklet i «noosfæren» trenger ikke det didaktiske systemet og inneholde en Y. Selvstudier i form av Y som en tom mengde (Chevallard, 2015), kan også være et didaktisk system og selv med Y ulik en tom mengde er det ikke en selvfølge at den *underviste kunnskapen* er den som faktisk læres av X i systemet – derav en siste transposisjonsprosess.

Figur 2.3

Didaktiske transposisjonsprosesser



Illustrasjon av de didaktiske transposisjonsprosessene oversatt fra originalspråk av meg (Chevallard & Bosch, 2020b, s. 214)

2.5 Teori for den didaktiske situasjonen

Selv om ATD i all hovedsak vil fungere som teoretisk rammeverk for denne oppgaven, med tanke på hva kunnskap er og hvordan denne opererer i ulike instutisjoner, er teori for den didaktiske situasjonen i matematikk (forkortelse TDS) også viktig for å kunne forklare spesielt didaktisk ingeniørvirksomhet. Rammeverket så sitt lys på 1970-tallet med Guy Brousseau i spissen, og har vært under stadig utvikling siden den gang (Strømskag, 2020b).

TDS omtales først og fremst som et verktøy for utvikling og analyse av undervisningssituasjoner innenfor matematikk, i tillegg til personlig og profesjonell utvikling som lærer. Prinsipielt handler TDS om innføringen av målkunnskap som en optimal løsning av et gitt problem, og et ønske om at læreren skal klare å overføre problemet til elevene på en meningsfull måte. Tanken her er at når elevene kjenner på nytteverdien av å løse et slik problem blir målkunnskapen også meningsfull og ikke minst nyttig (Mangiante-Orsola et al., 2018).

2.6 Didaktisk ingeniørvirksomhet

Didaktisk ingeniørvirksomhet er en metodologi for design av didaktikk innenfor TDS, og er en kvalitativ analyseform bestående av fire faser. De fire fasene er *forberedende analyse, design og a priori-analyse*, *realisering, observasjon, datainnsamling og in vivo-analyse*, og *a posteori-analyse og validitetsvurdering* (Strømskag, 2020a).

I den første fasen av didaktisk ingeniørvirksomhet ønsker en å analysere den matematiske kunnskapen som er i fokus, eller målkunnskapen fra et TDS perspektiv. Denne analysen har igjen tre underkategorier, i form av epistemologisk, institusjonell og didaktisk analyse. Hensikten med en slik oppdeling er å bedre kunne analysere hvilke forhold og begrensninger som ligger til grunn, og hvordan dette kan påvirke det didaktiske systemet. Videre vil en basert på *forberedende analyse* bevege seg inn i neste fase, som blant annet innebefatter å designe en undervisningssituasjon. *A priori-analyse* baserer seg da på hvilke forventninger en har knyttet til realisering av design, og dermed vil en kunne etablere hypoteser i denne fasen (Strømskag, 2020a).

Tredje fase, *realisering, observasjon, datainnsamling og in vivo-analyse*, innebærer at personene som utøver forskningen tar et steg tilbake, og i all hovedsak fungerer som observatører. *In vivo-analyse* vil da si analyse underveis i realisering av design, gjennom observasjoner og faktisk innsamling av data knyttet til gjennomføring. Denne tredje fasen er med andre ord sentral i opptakten til fjerde fase, da en gjennom *a posteriori-analyse og validitetsvurdering* skal teste hypotesene en etablerte *a priori* opp imot de innsamlede dataene. Tolkning av avvik mellom hypotese og data er en sentral del av denne fasen, hvor validiteten til den didaktiske ingeniørvirksomheten i all hovedsak baserer seg på korrelasjon mellom hypotese og *a posteriori-analyse* (Strømskag, 2020a).

Kapittel 3

Metodologi

I dette kapittelet er fokuset på metodologien brukt i studien, og hvordan det teoretiske rammeverket knyttes opp mot de ulike delene av nevnt studie. I starten av kapittelet er det forskningsdesign (3.1) innenfor det kvalitative paradigme som omtales og kontekstualiseres sammen med de analyseverktøyene som er presentert gjennom det teoretiske rammeverket. Videre følger et delkapittel om datainnsamling og datamateriale (3.2), før et om hvordan bruke prakseologi og didaktisk ingeniørvirksomhet som metode for videreutvikling (3.3). Siste delkapittel av metodedelen i denne studien omhandler utvalg og etikk (3.4).

3.1 Forskningsdesign

Studien befinner seg innenfor det kvalitative forskningsparadigme, med intervju som hovedkilde til dataene som ble samlet inn, og en analyse delt i to. Grunnen til denne oppdelingen er skille mellom undervisningsopplegget som ble utviklet i forkant av studien, intervjuene som ble gjennomført etter oppstarten og videreutviklingen av nevnt opplegg på bakgrunn av intervju, observasjon og tilbakemeldinger. Sentralt i analysen av disse delene – utvikling, gjennomføring og videreutvikling – står prakseologi og didaktisk ingeniørvirksomhet. Videre i dette avsnittet skal vi se på hovedsakelig utvikling og gjennomføring, som to av tre deler av analysen, og hvordan hver av disse skal angripes senere i teksten. Videreutvikling skal diskuteres ytterligere i eget delkapittel 3.3.

Innenfor utvikling og begrunnelse, eller med andre ord grunnen til at undervisningsopplegget ser ut som det gjør, er det spesielt didaktisk ingeniørvirksomhet og didaktiske transposisjoner som står sentralt. Ettersom didaktisk ingeniørvirksomhet fungerer som en helthetlig metode for analyse og design av didaktiske situasjoner (Strømskag, 2020a), vil det i første omgang være *forberedende analyse* og *design og a priori-analyse* som anses relevant for å bedre forstå utviklingsprosessen. Videre vil det være naturlig å diskutere hvilke didaktiske transposisjoner som er gjort i forbindelse med utviklingen, og ikke minst hvordan disse skal legge til rette for å stille spørsmål til verden og beholde en form for originalitet i kunnskapen som studeres (Chevallard & Bosch, 2020b).

Selve gjennomføringen vil analyseres med forankring i prakseologi, samt tredje fase didaktisk ingeniørvirksomhet - *realisering, observasjon og in vivo-analyse*. Chevallard (2006) anser som nevnt prakseologi som en grunnenhet for analyse av menneskelige handlinger, i tillegg til

at aspektet med distribusjon og tilegnelse av kunnskap kommer inn i denne delen, henholdsvis knyttet opp mot hvordan undervisningsopplegget ble mottatt og hvilke strategier som ble benyttet for å faktisk ta til seg kunnskapen. Det er først og fremst praksisblokken som skal benyttes til å analysere gjennomføringen av undervisningsopplegget, med innslag av teknologi. Grunnen til at det først og fremst er praksisblokken som benyttes er at dataene som samles inn i denne delen baserer seg på observasjon av elever i praksis, men når det er sagt vil elementer innen teknologi og teori kunne komme fram gjennom for eksempel diskusjon eller spørsmål. Ettersom temaet tekstbasert programmering virket på faglærer å være nytt for de aller fleste elevene, ville det vært rart å forvente at teoridelen står sterkt, men jeg vil naturlig nok fortsatt se etter denne typen datamateriale også. Type oppgaver, teknikker og til dels hvordan argumentere for teknikken en har brukt (teknologi) er meget relevant analyse av gjennomføringen. *Realisering, observasjon og in vivo-analyse* har her, som i forrige avsnitt om utvikling, en naturlig plass som del av en metodologi. Innenfor denne tredje fasen finner vi selve realiseringen av undervisningsopplegget, observasjoner fra klasserommet og analyse, samt tanker og refleksjoner som dukket opp underveis i gjennomføringen. *Datainnsamling* som også er en del av denne tredje fasen innenfor didaktisk ingeniørvirksomhet ble på mange måter et eget segment, samtidig som observasjoner og analyse underveis absolutt kan anses som en form for datainnsamling. Ettersom datainnsamling og datamateriale er neste delkapittel vil derfor denne delen av fasen behandles der, mens i analyse av gjennomføring er det de tre andre delene av denne fasen som er i fokus.

3.2 Datainnsamling og datamateriale

Innsamling av datamateriale i denne studien har hovedsakelig blitt uført i form av observasjoner tilknyttet gjennomføring av undervisningsopplegg og intervjuer i etterkant av undervisningsopplegget. Observasjonsmetoden som ble brukt i forbindelse med gjennomføring av undervisningsopplegget var uformell observasjon, som skiller seg fra formell observasjon gjennom mangel på struktur og en større mulighet til å ta tak i det som måtte være interessant der og da (Robson & McCartan, 2016, s. 322). Grunnen til at dette virket som en naturlig måte å angripe observasjoner som innsamlingsmetode på, er i forbindelse med usikkerheten knyttet til hvilke interessante momenter som oppstår i en undervisningssituasjon – og det var derfor ønskelig å opprettholde en stor grad av fleksibilitet.

Selve intervjuprosessen var semi-strukturert, noe som vil si at jeg hadde en intervjuguide med spørsmål jeg forholdt meg til, men at jeg kunne variere rekkefølge og tidsbruk tilknyttet de ulike spørsmålene og temaene. Hvert intervju ble gjennomført med fokusgrupper på tre

deltakere og fire grupper som utgangspunkt. Det å bruke fokusgrupper ble valgt på grunn av at intervju som metode for datainnsamling er en tidkrevende prosess, og på bakgrunn av de fordeler og ulemper som presentert av Robson og McCartan (2016, s. 299-300). Det var spesielt tre fordeler og ulemper som ble hovedfokus i denne vurderingsprosessen (Robinson, 1999):

(1) Innsamling av kvalitativ data fra flere deltakere av gangen.
(2) Det er enklere å oppdage delte meninger innad eller felles for gruppen.
(3) Muligheten deltakerne har til å komme med egne meninger, samtidig som andre sine innspill kan virke stimulerende i deres tanke- og refleksjonsprosesser.

På den andre siden var det spesielt ulemper eller begrensninger tilknyttet følgende:

(1) Justering av antall spørsmål og hvorvidt en kan gå i dybden.
(2) Konflikter mellom deltakere.
(3) Hvordan antall deltakere og gruppedynamikk påvirker konfidensialitet i form av hva deltakerne ønsker å dele.

De tre fordelene og ulempene er hentet ut fra en utvidet liste presentert av Robinson (1999, s. 909-910, sitert i, Robson & McCartan, 2016, s. 299-300).

Når det gjelder spørsmålene som ble stilt i intervjuprosessen ble søkelyset på følelser, tro og holdninger i større grad en fakta. Det er naturlig nok et skille her mellom spørsmålene som omhandler deltakerne og deres faktiske erfaringer, kontra deres opplevelse av undervisningsopplegget og programmering som del av matematikkundervisningen. Denne typen data omtales som vanskelig og kompleks, men også utrolig viktig kilde for egenevaluering (Robson & McCartan, 2016, s. 286). I forbindelse med den semi-strukturerte intervjuformen fungerte en intervjuguide som støtte for datainnsamlingen, med fokus på spørsmål som ikke trenger å komme i samme rekkefølge for hver fokusgruppe, samt at en som intervjuer skal kunne ta tak i interessante momenter som dukker opp hos den enkelte fokusgruppe. Denne guiden ble utviklet på bakgrunn av noen etablerte sjekkpunkter, som at en ønsker å unngå i utviklingen av spørsmål er lange, todelte, vanskelig formulerte, ledende og forutinntatte spørsmål (Robson & McCartan, 2016, s. 288).

Videre er rekkefølgen og typen spørsmål noe en som intervjuer bør være bevisst på, og en naturlig rekkefølge presenteres av Robson og McCartan (2016, s. 290) som introduksjon, oppvarming, hoveddel, nedkjøling og avslutning. I denne studien ble introduksjon foretatt før

opptaker ble skrudd på, og støttet av et samtykkeskjema deltakerne så over og signerte. Oppvarmingen var den samme for alle gruppene, da de fikk spørsmål om hva slags programmering de har hatt før, mens hoveddelen i større grad handlet om deres følelser knyttet til undervisningsopplegget og delvis plasseringen og relevansen av programmering i matematikk. Intervjuguiden ble her brukt som en slags sjekkliste for at alle de fire gruppene skulle ha et lignende sett med spørsmål. Rekkefølgen, ordlyden og tiden brukt på hvert spørsmål varierte riktignok fra gruppe til gruppe. Dette virket naturlig, da gruppene hadde ulik innfallsvinkel på de forskjellige spørsmålene, slik at vi kunne komme inn på et nytt tema fra ulike utgangspunkt. For å gjøre intervjuformen mer samtaleorientert ble det derfor viktig å gå videre med det temaet deltakerne hadde vært innom, istedenfor å holde seg til strukturen i intervjuguiden.

Nedkjøling ble noe forskjellige for de ulike gruppene, da den semi-strukturerte formen gjorde at hoveddelen ble avsluttet på litt ulike stadier. Hovedfokus under nedkjøling var dog det samme for de ulike gruppene, i form av oppklaring og oppsummering dersom det var noe spesielt som ble plukket opp eller sagt underveis i hoveddelen. Avslutning skjedde også uten opptaker, hvor det viktigste var å ta vare på deltakerne i form av å svare på eventuelle spørsmål og repetere hvilke muligheter og rettigheter de har som del av studien.

3.3 Prakseologi og didaktisk ingeniørvirksomhet som grunnlag for videreutvikling

En sentral del av denne studien er videreutvikling av det utprøvde undervisningsopplegget, slik at det ideelt sett kan være en ressurs for både elever og lærere i introduksjon av tekstbasert programmering i 1T. Rent metodisk skal denne videreutviklingen bygge på en prakseologisk modell, gjennom isoleringen av «de fire t'ene», i tillegg til at didaktisk ingeniørvirksomhet blir et viktig analyseverktøy i tolkningen av det opprinnelige undervisningsopplegget, samt prediksjon og analyse av hvordan videreutviklet opplegg vil bli mottatt og bedre løser ulike problemer knyttet til didaktiske situasjoner, som hvordan elever forstår oppgaver, hvilke muligheter som finnes for å gå videre med et spesifikt tema, oppstart av oppgaver, og lignende. Spesifikt for didaktisk ingeniørvirksomhet blir altså dens rolle i analyse av de didaktiske situasjonene forbundet med uttesting av opplegget. De fire fasene for analyse, fra *forberedende* til *a posteriori-analyse og valididetsvurdering* skal benyttes i dette arbeidet, hvor også *observasjoner* og *datainnsamling* kommer inn som en naturlig del av denne analysemetoden (Strømskag, 2020a). I prosessen med videreutviklingen vil det derimot

kun være *forberedende analyse* og *design og a priori-analyse* som blir tatt i bruk. Dette virket naturlig da uttesting av videreutviklet opplegg ikke er tiltenkt.

Den prakseologiske delen av videreutviklingen bygger som nevnt på «de fire t'ene», med bakgrunn i de antropologiske postulatene som ligger til grunn for prakseologi. Hvis en tenker seg et startpunkt tilknyttet en aktivitet (a) i en institusjonalisert sammenheng, for eksempel i et klasserom, deler Chevallard (2019) opp disse aktivitetene i ulike oppgaver (t – «task» på engelsk). Videre kan en da representere aktiviteten på en matematisk form ved:

$$a = t_1 \wedge t_2 \wedge t_3 \wedge \ldots \wedge t_n$$

Lille t, eller en spesifikk oppgave, er nå en del av store T, som vi kjenner som type oppgaver fra **2.3**, og etter hvert som en gjør en slik type oppgave flere ganger utvikles teknikker for å løse den typen. Dette er den nevnte praksisblokken innenfor prakseologi, og kan i en denne spesifiserte sammenhengen relateres til både problemløsningsstrategier og algoritmisk tenkning slik kompetansemål for programmering presenteres i læreplan for matematikk 1T (Utdanningsdirektoratet, 2020). Videre i denne utviklingsprosessen er det viktig å være bevisst på hvilke prakseologier som tas i bruk, samt hvilke verktøy og teknikker som trengs i forbindelse med disse (Bosch & Gascón, 2014). Her skiller en mellom hvem som er hovedkarakter i det didaktiske systemet S(X, Y, κ), er det elevene (X) eller læreren (Y)? Et tydelig skille her vil for eksempel være verktøyene en elev trenger for å utforske en programmeringsoppgave, og verktøyene en lærer trenger for å kunne legge til rette for den didaktiske situasjonen.

Sentralt for arbeidet med videreutvikling var dessuten et ønske om å skape noe bedre, og ikke minst understreke at selv om et opplegg fungerer, er det gode muligheter for å kunne ta det ett steg videre. Prakseologi og didaktisk ingeniørvirksomhet er metoder for design og analyse, slik at de kan hjelpe med å avdekke og ta bort mulige vanskelige situasjoner i en didaktisk sammenheng. Samtidig vil de være til hjelp i utviklingen, gjennom å bevisstgjøre utvikler på hvilke situasjoner en ønsker skal oppstå, og på en systematisk måte kunne sikte seg inn mot det en kan kalle en utopi, nemlig at alle deltakere i et klasserom engasjeres, får hjelpen, utfordringene og har tilgang på alle ressursene de trenger for et ideelt utbytte av læringssituasjonen.

3.4 Utvalg av deltakere og etikk

I denne studien er det et klart skille i utvalget av deltakere, nemlig mellom deltakerne som kun var med på undervisningsopplegget, og de som også tok del i intervjuprosessen etter endt

undervisningsopplegg. For gruppen som kun var med i klasserommet var det viktig for opprettholdelse av god forskningsetikk å unngå observasjoner som kunne lede til brudd på personvern, og i den forbindelse opprettholde en viss generalitet i det som ble notert av observasjoner. Jeg hadde ingenting med utvalg av den enkelte intervjudeltakere å gjøre, men valgte istedenfor å benytte meg av faglæreren, og la den ta en prat med klassen. Det var viktig for meg å ikke presse eller velge ut elever jeg ville ha med, på grunn av at en vil unngå å legge føringer for hvilke resultater som skal oppnås og ikke minst at det er frivillig å delta (NESH, 2021).

Blant de forskningsetiske retningslinjene en skal jobbe etter i tråd med den nasjonale forskningsetiske komité for samfunnsvitenskap og humaniora, finner en blant annet *hensyn til personer,* som en av fem sentrale retningslinjer (NESH, 2021). Ettersom hovedkilden til datamateriale i denne studien er intervju og observasjon av personer har jeg valgt å trekke frem spesielt denne. Når det gjelder utdypning av denne retningslinjen finner en blant annet skille mellom etisk og juridisk vern av personen, samt at samtykke og frivillighet er helt essensielle aspekter i ønske om å skape trygghet og bygge tillitt mellom forsker og deltaker. Her blir også indirekte deltakere omtalt, og viktigheten av å også ivareta disse, gjennom informasjon vedrørende studien og hva som faktisk kommer til å skje (NESH, 2021).

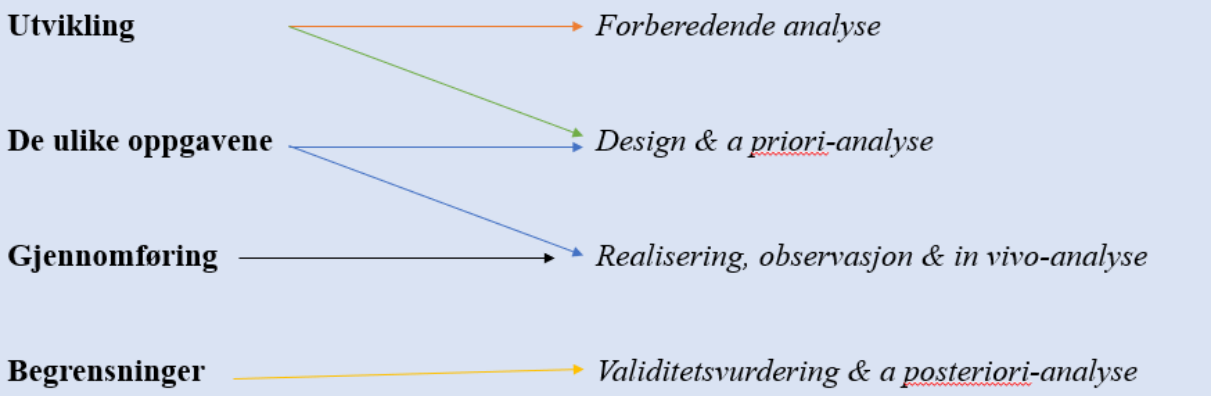
Under den forskningsetiske retningslinjen, *forskerfellesskapet*, finner en det som omtales som *fordreining og fortielse*. Ettersom jeg i denne studien arbeider med datamateriale i form intervjuer og observasjoner er et viktig forskningsetisk perspektiv at disse må håndteres og tolkes deretter. Det å unngå tolkninger eller fjerne data som ikke underbygger ønsket utfall er naturlig nok dårlig forskningsetisk skikk (NESH, 2021). Altså vil det å være bevisst på at min subjektive kategorisering av data skal være så objektiv og omfattende som kreves for at meningene og poengene som er ytret ivaretas, noe som blir essensielt for validiteten i studien. Måten jeg ønsker å forsikre meg om at dette er tilfellet er gjennom å inkludere sitater fra transkripsjon, samt være tydelig på hva som er mine tolkninger og hva som hentes direkte fra datamateriale.

Kapittel 4

Undervisningsopplegget

Det fjerde kapittelet i denne oppgaven handler om undervisningsopplegget som på mange måter er grunnlaget for datainnsamlingen, og ikke minst essensielt med tanke på videreutvikling. Hovedfokuset vil være utvikling, oppgavetyper, gjennomføring og begrensninger med didaktisk ingeniørvirksomhet som verktøy for analyse av den didaktiske situasjonen. I analysen av denne didaktiske situasjonen vil søkelyset være på koblingen mellom de ulike delene av prosessene fra ide og utvikling til endt gjennomføring og refleksjoner opp mot de fire fasene av didaktisk ingeniørvirksomhet, som illustrert i figuren under.

Figur 4.1:



Figur 4.1: Illustrasjon av koblingene mellom de ulike delene av utvikling og gjennomføring av undervisningsopplegg, og de fire fasene av didaktisk ingeniørvirksomhet

Som en kan se av figuren er det noe overlapping mellom de ulike delprosessene og fasene, dette skal forklares nærmere i delkapitlene som følger. Ettersom analysen av undervisningsopplegget foregår ved bruk av didaktisk ingeniørvirksomhet vil det være naturlig å etablere en målkunnskap i forbindelse med den didaktiske situasjonen som analyseres. I en noe institusjonalisert forstand kan en enkelt og greit si at å nå et tilstrekkelig nivå, som beskrevet av kompetansemålet tilknyttet temaet kan fungere som målkunnskap. Problemene oppstår da når kompetansemålet blir en begrensning for hvor dypt inn i et tema elevene kan komme, jamfør Chevallard (2015) sin metafor om å besøke monumenter istedenfor å studere dem. På en annen side er dette undervisningsopplegget ment som en introduksjon og et hjelpemiddel til videre studier av tema programmering, så et altfor stort

fokus på generalisering og dypdykk inn i monumentene sin historie kan kanskje virke mot sin hensikt. Målkunnskapen for dette undervisningsopplegget ble derfor formulert som:

«Elevene skal kunne forklare og bruke grunnleggende komponenter i matematisk programmering.»

Begrepene *grunnleggende* og *matematisk programmering* handler i denne sammenhengen om ferdigheter og teknikker for førstnevnte, for eksempel å kunne håndtere betingelser og vilkår. *Matematisk programmering* er en større grad en poengtering, da programmeringen og oppgavene elevene skal holde på med skal ha forankring i faget matematikk, istedenfor å eksempelvis skulle utvikle en nettside. Et viktig poeng med denne målkunnskapen er at selv om ordet *grunnleggende* blir brukt, skal undervisningsopplegget ikke stanse elever fra å ville gå videre – tvert imot. Målkunnskapen kan i den forstand ses som et idealisert minstemål på den kunnskapen som skal oppnås. Videre er søkelyset på *matematisk programmering*, da basiskunnskaper innenfor programmering i matematikk-faget og et mer tradisjonelt programmeringsemne kan være nokså forskjellig.

Jeg kommer til å ta i bruk observasjoner fra realiseringen av undervisningsopplegget, spesielt i delkapittel 4.2 og 4.3, da både *de ulike oppgavene* og *gjennomføring* er koblet opp mot tredje fase av didaktisk ingeniørvirksomhet som inneholder nettopp observasjon. Til tross for at observasjonene er en del av datamaterialet virket det naturlig å inkludere disse her, da kategorisering av egne observasjoner gjennom en selvetablert prakseologi virket mindre relevant, enn å koble de mot situasjonen observasjonene kom fra.

4.1 Utvikling av undervisningsopplegget

Det naturlige startpunktet når det snakk om et undervisningsopplegg blir her utviklingsprosessen, og hvilke tanker og refleksjoner som gikk inn i nettopp dette. Fra figuren over kan en se at en på mange måter kan relatere denne utviklingen til de to første fasene av didaktisk ingeniørvirksomhet, nemlig *forberedende analyse* og *design & a priori-analyse*. Grunnen til at begge disse fasene er satt til utviklingsprosessen er at selv om design kan virke som det mest nærliggende begrepet, så er forberedelser og analyse av hva en forventer blir interessant og utfordrende noe en kan bruke i utviklingen.

Som kjent fra avsnittet om didaktisk ingeniørvirksomhet, deles *forberedende analyse* inn i tre underdeler, nemlig epistemologisk, institusjonell og didaktisk. Hovedformålet med en slik analyse er avdekke forhold og begrensninger som vil påvirke det didaktiske systemet (Strømskag, 2020a). I forbindelse med undervisningsopplegget ble denne tredelte analysen en

sentral del i arbeidet med å sette seg inn i hvilken kunnskap som skal studeres, hvordan den oppnås og hvor den kommer fra som den epistemologiske analysen. Her kan en se en tydelig kobling mellom innledende analyse i didaktisk ingeniørvirksomhet, og kjennetegn ved det didaktiske systemet i ATD. Kjennetegn som å stille spørsmål til verden, opprettholde en form for originalitet, bevisstgjøring knyttet til didaktiske handlinger og hvordan disse handlingene kan hjelpe med å oppnå kunnskap. Den institusjonelle delen av analysen handlet om å sette seg inn i og forstå hvilken kunnskap som kreves gjennom kompetansemål, læreplan, eksamensveiledning og lignende – også her ser en en kobling til ATD, hvor nettopp kompetansemål omtales som en «constraint» eller begrensning (Chevallard, 2015; Strømskag, 2020a). Den tredje delen av *forberedende analyse*, nemlig didaktisk analyse, handlet i hovedsak om å forsøke å forutse ulike situasjoner som kunne oppstå, og å kunne skape en form for kontroll av hvilke didaktiske «under-situasjoner» som oppstår. Begrepet didaktisk «under-situasjon» er her konstruert kun for å skille mellom den store didaktiske situasjonen, nemlig selve undervisningsopplegget, og de mange mulige «under-situasjonene» som oppstår mens en jobber med dette opplegget. Eksempler på slike er hvis en eller flere elever står fast, trenger en utfordring eller ikke ser nytteverdien av verktøyet programmering.

Innledningsvis i designdelen av utviklingen var det spesielt to spørsmål som dukket opp; i hvilken rekkefølge skal temaene introduseres, og hvilken målkunnskap skal en ha for gruppen som helhet? Svaret på de to spørsmålet ble funnet henholdsvis gjennom vurdering, erfaring med andre læringsarenaer og logikk for det førstnevnte spørsmålet. Vurdering og erfaring med andre læringsarenaer handler her om innhold, altså hva som skal inkluderes, mens logikken tar for seg rekkefølgen – da noen temaer kan bygge videre på andre temaer, som vilkår og løkker. Spørsmål nummer to, om målkunnskapen, kom i en noe institusjonalisert forstand enda tydeligere fram gjennom ny eksamensveiledning (Utdanningsdirektoratet, 2024b). Ut fra nevnt eksamensveiledning kan det nå virke som elevene skal ta steget fra blokkprogrammering eller lignende til tekstbasert programmering i 10. klasse.

Eksamensveiledning 1T:

> *Ved skriftlig eksamen kan kandidaten velge å bruke programmering som løsningsstrategi for å løse et matematisk problem. Det kan også være oppgaver hvor et matematisk problem er representert som en programkode. Kandidaten skal vurdere innholdet i koden og bruke denne til å løse det matematiske problemet. I matematikk 1T er programmering beskrevet eksplisitt. Derfor kan det bli gitt oppgaver i matematikk 1T hvor kandidaten blir bedt om å levere program som svar (Utdanningsdirektoratet, 2024b).*

Eksamensveiledning 10. trinn:

> *Kandidaten kan velge å bruke programmering som løsningsstrategi for å løse et matematisk problem. Det kan også være oppgaver hvor et matematisk problem er representert som en programkode. Kandidaten skal vurdere innholdet i koden og bruke denne til å løse det matematiske problemet. Kandidaten skal vurdere innholdet i koden og bruke denne til å løse det matematiske problemet (Utdanningsdirektoratet, 2024a).*

Det virket allikevel naturlig med en rekkefølge som ikke nødvendigvis bygger videre på det de skal ha sett på ungdomsskolen, og derfor starte med betingelser og noe variabelforståelse. Videre er rekkefølgen for hele undervisningsopplegget som følger:

(1) Betingelser
(2) Vilkår
(3) Lister
(4) Funksjoner
(5) Løkker
(6) Ekstra eksempler og oppgaver
(7) Plotting

Hvordan håndtere betingelser og variabler i programmering, og ikke minst hvordan skille mellom de to står sentralt i veldig mange av de andre delene av programmering, for eksempel løkker og vilkår, og virket derfor som et naturlig sted å starte.

Videre ble det valgt å fokusere på vilkår, hvor det spesielt er if-setningen som er sentral. Denne delen av programmering viderefører i stor grad betingelser inn i en mer praktisk sammenheng. Fra et prakseologisk standpunkt kan en si at en etter å ha besøkt logosblokken til betingelser, gjennom fokuset på hvorfor vi bruker betingelser, tar med seg denne inn i

praksisblokken, ved å se på mer praktisk relevante bruksområder enn å skulle bruke «bool()» til å teste betingelser gjennom «true/false». Neste post i undervisningsopplegget ble valgt å være lister, grunnen til dette var at håndtering av lister og ulike verktøy for å kjapt konstruere større lister med tall kan være en fin vei inn til å se nytteverdien av programmering som verktøy. Hvis elevene sitter og finner et og et tall, når programmering gir mulighet til å gjennomføre flere hundre operasjoner på sekunder, var det tanken at dette skulle motivere bruken av programmering kontra et program som geogebra. LIST-oppgaver, det vil si oppgaver med lav inngangsterskel og stor takhøyde, kan utvilsomt være en motiverende faktor (Wæge & Nosrati, 2021). Ved å la inngangsterskelen være et og et eller tre og tre tall, mens takhøyden kan være å gjøre flere operasjoner med ti-talls variabler og parametere.

Etter vilkår og lister fulgte funksjoner, som i seg selv er et begrep mange elever nok må gjenoppdage i denne sammenhengen, ettersom funksjoner i programmering har en noe bredere betydning enn den de er vant til fra matematikk. Her kan elevene lage funksjoner som de kjenner fra mattefaget, men også bruke samme oppsett til å lage en kode som kan gjenbrukes flere steder eller gjentas med ulike parametere. Etter funksjoner beveget fokuset seg over på løkker, i form av for- og while-løkker. Selve konseptet med løkker som gjentas over henholdsvis ett intervall (for-løkke) eller til en betingelse er oppfylt (while-løkke) åpner for å kunne gjøre veldig mye forskjellig, og i boken «Programmering i skolen» blir det pekt på som noe veldig logisk, men også som noe av det mest utfordrende å bli vant til (Haraldsrud et al., 2020, s. 47).

Etter løkker fulgte et sammensatt eksempel, hvor planen var å ta i bruk flere av komponentene de hadde vært gjennom, og la elevene se hvordan de kunne kombineres til noe større. I tillegg ble det inkludert en blokk om plotting av grafer, da dette kan være en interessant måte for elevene å visualisere det koden de skriver returnerer. Som nevnt med lister, er det også noe uklart hvorvidt plotting trenger og være en del av målkunnskapen i matematikk 1T (Utdanningsdirektoratet, 2020). Motivasjonen for å allikevel ta det med i undervisningsopplegget bygget derfor på nettopp det å kunne visualisere, og ikke minst bruke styrken til programmering i større grad enn hvis en kun skal bruke print-funksjonen og håndtere et og et tall eller sett med tall.

Når det gjelder hvilke forventninger som var tilknyttet realisering av design, altså a priori-analyse (Strømskag, 2020a), var det spesielt mengden informasjon tilgjengelig og måten den ble presentert på som var de store spørsmålene. På grunn av skolen sitt valg av editor, MU istedenfor Anaconda Navigator og dermed Jupyter Notebook, kom det med en forventning om

at brukergrensesnittet ikke kom til å være optimalt, da det ikke finnes samme muligheter for å skille tekst, kode, oppgaver og ulike temaer i blokker. I tillegg fulgte forventninger om at åpenhet i oppgaver og eksempelkode, samt en alternerende teori/oppgave struktur, skulle kunne gjøre det enklere for elevene å komme i gang med oppgaver. Kommentarer i koden, begrepsliste og en tydelighet på at selve opplegget skulle kunne brukes senere som et hjelpemiddel, var også et viktig aspekt med tanke på forventninger tilknyttet nytteverdien som ble opplevd og hvordan de håndterte åpenheten i oppgaver. Oppsummert var det altså forventninger tilknyttet brukergrensesnitt, åpne oppgaver, struktur og nytteverdi, som igjen skal under lupen i *a posteriori-analyse*.

4.2 De ulike oppgavene

Som nevnt tidligere følger dette undervisningsopplegget en alternerende struktur med forklaring, eksempelkode og oppgaver i den rekkefølgen. Tanken bak nettopp den rekkefølgen var at elevene skal kunne ha forklaring og eksempler å lene seg på når de løser oppgaver selv eller sammen. Det at de fortløpende tester ut enkeltkomponenter med få begrensninger til hvordan, hadde også som baktanke at skulle styrke deres forståelse og ikke minst gi de muligheten til å gjøre oppdagelser selv. Når det gjelder de ulike typene oppgaver har jeg valgt å oppsummere dem i fire kategorier; kodeforståelse, matematisk forståelse, diskusjonsoppgaver og det å lage kode. De fire hovedkategoriene vil naturlig nok ha en form for overlapping, og i noen tilfeller handler det om å forstå små deler av en kode eller plukke opp viktige elementer ved oppbygning av koden de ser på. For å understreke forskjellene mellom de fire ulike oppgaveformene presenteres fire eksempler og tilhørende avsnitt under.

4.2.1 Kodeforståelse

I første kategori av oppgaver er det først og fremst forståelse av struktur og enkeltkomponenter i koden som står i fokus. Begrepet kodeforståelse kan også inkludere flere underkategorier som feilsøking, logikk, syntaks og lignende. I eksempelet under er tanken at elevene skal forstå, og potensielt undersøke med å kjøre koden, hvorfor en må enten redefinere «a» og «b» til ønsket verdi eller ha egne variabler («A» og «B») etter at while-løkken som står øverst i koden er kjørt.

Bilde 4.1: Celle om løkker

```
#LØKKER
import numpy as np

a=7
i=0
b=53

#a+=1 betyr det samme som a=a+1, og ny a vil oppdateres for hver runde i løkka
while a<b:#while er en løkke som gjentas helt til en betingelse er møtt
    a+=1 #for hver runde i løkka legges det til en i a
    #print(a)
    i+=1 #"i" kan i dette tilfellet f.eks være en måte å telle antall runder løkka kjører


A=7
B=53
k=0
for j in range(A,B): #For er en annen type løkke hvor variabel j får et intervall (in range(start, stop, steg)) fra A til B
    k+=j #k+=j vil si at alle tall mellom A og B summeres sammen
    #print(k)

#SPØRSMÅL: Hvorfor må jeg ha egne store A og store B, kunne jeg ikke brukt lille a og lille b fra forrige?
```

Bilde 4.1: Viser hvordan løkker presenteres i de originale undervisningsopplegget.

En noe dypere tanke bak oppgaven er at elevene skal jobbe med forståelse av både variabler, løkker og hvordan en while-løkke påvirker variablene som er en del av dens betingelse. Hvis en hadde fjernet «A» og «B», og heller brukt «range(a,b)» ville «b» fortsatt vært 53, mens «a» ville hatt den verdien som bryter med betingelsen i løkken – i dette tilfelle 53. Ser en da på for-løkken, ville denne fortsatt kjørt, men istedenfor et intervall fra 7 til 53 ville en kun hatt elementet 53. I tillegg til at kodeforståelse er et viktig steg mot det å kunne lage egen kode, er det også en del av det elevene skal vurderes i (Utdanningsdirektoratet, 2024b), samt at en i matematikk R1 tar denne delen av programmering et steg videre med forståelse som hovedfokus (Utdanningsdirektoratet, 2024c).

4.2.2 Matematisk forståelse

Til forskjell fra kodeforståelse handler matematisk forståelse i større grad om å forstå de matematiske problemene og utfordringene som oppstår når en koder i matematikk. For mange elever kan dette være en ny måte å måtte tenke på, da programmeringsspråket Python også leverer feilmeldinger i forbindelse med ugyldige matematiske operasjoner. Når det gjelder de matematiske problemene elevene skal møte i dette undervisningsopplegget har hovedfokus vært på å gjøre de gjenkjennbare fra pensum, samtidig som det skal være relevante matematiske temaer i forbindelse med programmering som et verktøy for å bedre kunne forstå og løse problemet. I eksempelet under er det lagd en funksjon som utnytter angredagsformelen, eller ABC-formelen som den ofte kalles, til å bestemme hvor mange nullpunkter et polynom av grad to har og hvilke tilhørende x-verdier disse har.

Bilde 4.2: Eksempel om nullpunkter fra generell andregradsfunksjonen

```python
#Under ser vi et annet eksempel på hvordan en funksjon kan se ut ved bruk av vilkår og betingelser
def andregrad(a,b,c):
    rot = b**2 - 4*a*c #Rotutrykket i ABC-formelen regnes ut
    if rot<0: #Hvis rotutrykket er mindre enn null er det ingen nullpunkter
        return print('Funksjonen har ingen nullpunkter')
    elif rot==0: #Hvis rotutrykket er lik null er det ett nullpunkt
        x_1 = (-b)/2*a
        return print('Funksjonen har ett nullpunkt, x=', x_1)
    else: #Ellers, altså hvis rotutrykket er større enn null, har vi to nullpunkter
        x_1 = (-b+np.sqrt(rot))/2*a
        x_2 = (-b-np.sqrt(rot))/2*a
        return print('Funksjonen har to nullpunkter, x=', x_1, '& x=', x_2)

#andregrad(1, -4, 2)
```

Diskusjonsoppgave

Hvorfor regner vi "rot" uten å ta kvadratroten? Kan dette forklares matematisk?

Dette kan gjøres for hånd og i geogebra også, men når kan det være nyttig å ha slik funksjon?

Fungerer funksjonen for lineære funksjoner?

Bilde 4.2: Eksempelkode med tilhørende diskusjonsoppgaver, som tar i bruk funksjoner og vilkår for å finne antall nullpunkter og tilhørende x-verdi for andregradsfunksjoner.

Som en kan se her er det en overlapp mellom kodeforståelse, matematisk forståelse og diskusjonsoppgaver, men ved å se spesielt på «Hvorfor regner vi «rot» uten å ta kvadratrot?» og «Fungerer funksjonen for lineære funksjoner?» har begge en forankring i den matematiske forståelsen til elevene. En kan selvfølgelig argumentere for at disse oppgavene også passer under kategorien kodeforståelse og diskusjon, men her er det valgt å skille mellom disse oppgavetypene. Ser en på det første spørsmålet handler det naturlig nok om at en skal kunne ta inn alle typer andregradspolynomer og få returnert antall nullpunkter. Uttrykket «rot» kan være større null, lik null eller mindre enn null, og hvis sistnevnte skjer vil en få et negativt rotuttrykk dersom en inkluderer kvadrat i innledende utregning. Denne formen for argumentasjon og tenkning, hvor en kombinerer matematikk og kodestruktur er den andre kategorien av oppgaver i undervisningsopplegget.

Ser en på det andre nevnte spørsmålet, hvorvidt funksjonen fungerer for lineære funksjoner, skulle en kanskje tro det ligger i navnet at den kun fungerer for andregradsfunksjoner. Hvis en derimot har lyst til å ta et steg vekk fra språklig argumentasjon, og heller se på det matematiske i koden, ser en at «rot» regnes ut som normalt hvis parameteren «a» er null. Ettersom $b^2 \geq 0$ unngår en å få ut at det ikke finnes noen nullpunkter, noe som i seg selv er greit

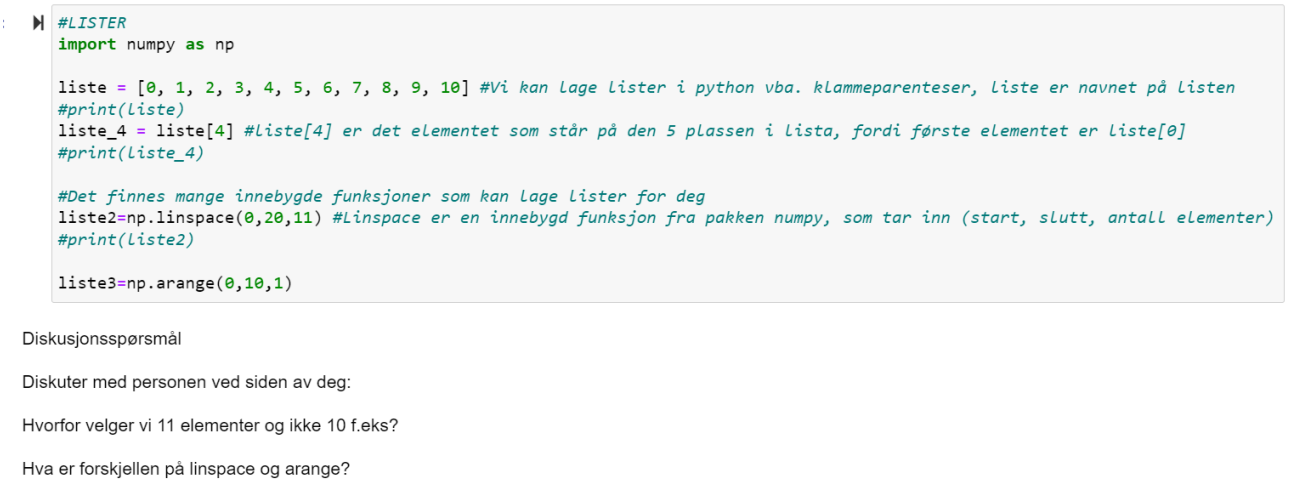
for lineære funksjoner over den reelle tallmengden, men det første store utropstegnet burde dukke opp når en skal dele på «2a». En skal med andre ord dele på null, med det kan elevene argumentere matematisk for at denne koden ikke fungerer for lineære funksjoner.

I tillegg til den matematiske forståelsen er det fullt mulig å utvide spørsmålet, slik at kodeforståelse blir en større del av diskusjonen for denne eksempelkoden. En kan argumentere for at «rot» er en naturlig mellomregning for å etablere en naturlig betingelse for if-setningen som senere bestemmer antall nullpunkter basert på betingelsen. En kan diskutere utvidelser av koden som hadde gjort det mulige å regne ut nullpunkt for lineære funksjoner også, det er med andre ord mange muligheter og selv om kodeeksempelene som presenteres i en spesifikk oppgave-kategori finnes det flere bruksområder.

4.2.3 Diskusjonsoppgaver

Vi har sett på oppgavene tilknyttet forståelse, både innenfor faktisk kode og det matematiske aspektet, og sett at det er en form for overlapp mellom de to nevnte og den tredje kategorien kalt diskusjonsspørsmål. En kan naturlig nok løse oppgaver kategorisert som kodeforståelse og matematisk forståelse gjennom diskusjon, men hovedfokus for diskusjonsspørsmål er logikk, endringer og anvendelse. Dette høres kanskje ut som komponenter som kan være en del av alle oppgavekategorier, men forståelsesoppgavene er faktiske oppgaver som kan komme på en eksamen i henhold til eksamensveiledning (Utdanningsdirektoratet, 2024b), mens diskusjonsoppgaver først og fremst er konstruert for klasserommet uten en direkte kobling til institusjonalisert vurderingssituasjon. Der en med kodeforståelse gjerne har en kode som fungerer, og oppgaven er å forklare hvorfor den fungerer og hva den gjør, eventuelt motsatt, hvorfor den ikke fungerer og hva en må gjøre for at den skal fungere, kan diskusjonsspørsmål åpne for utforskning av flere teknikker, og med det kan elevene arbeidet med logosblokken, eller i hvert fall grunnlaget for logos, i sin praksis (Bosch & Gascón, 2014).

Bilde 4.3: Celle om lister med tilhørende diskusjonsspørsmål



```
#LISTER
import numpy as np

liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] #Vi kan lage lister i python vba. klammeparenteser, liste er navnet på listen
#print(liste)
liste_4 = liste[4] #liste[4] er det elementet som står på den 5 plassen i lista, fordi første elementet er liste[0]
#print(liste_4)

#Det finnes mange innebygde funksjoner som kan lage lister for deg
liste2=np.linspace(0,20,11) #Linspace er en innebygd funksjon fra pakken numpy, som tar inn (start, slutt, antall elementer)
#print(liste2)

liste3=np.arange(0,10,1)
```

Diskusjonsspørsmål

Diskuter med personen ved siden av deg:

Hvorfor velger vi 11 elementer og ikke 10 f.eks?

Hva er forskjellen på linspace og arange?

Bilde 4.3: Eksempelkoden tar for seg hvordan en kan manuelt lage lister, og hente ut enkeltelementer, samt to innebygde funksjoner som genererer ønskede lister basert på ulike parametere.

Fra bilde 4.3 kan vi se et eksempel på diskusjonsoppgaver tilknyttet lister. Disse oppgavene kan en si omhandler logikk i hovedsak, men med et innslag av anvendelse og matematisk forståelse, spesielt tilknyttet antall elementer og steglengde. De er nokså lukkede, da en i bunn og grunn kun ser på forskjellen mellom «linspace» og «arange». Den logiske delen av oppgaven bygger på at elevene enkelt og greit skal reflektere over at fra 0 til 20, så er det 21 heltallige elementer, noe som kan virke trivielt, men som sammen med at første element i en liste står på plass null er en slags vanesak i programmering. Forskjellen på å velge 10 og 11 elementer for «linspace» mellom 0 og 20 er altså om du vil ha [0, 2.2222, 4.4444, …, 17.7778, 20] eller [0, 2, 4, …, 18, 20]. Når det gjelder neste deloppgave er det et par sentrale ulikheter som bygger på en logikk som elevene sannsynligvis vil møte igjen mens de programmerer. Der «linspace» gir deg en liste på intervallet [start, stopp] med et spesifisert antall elementer fungerer «arange» slik at du har et intervall [start, stopp) med en spesifisert steglengde. Med andre ord vil eksempelet fra bilde 4.3 gi listen [0, 1, 2, 3, …, 9]. Motivasjonen bak å skulle diskutere disse liste-verktøyene er altså både å gi elevene ulike innebygde funksjoner til å kunne kjapt lage større lister, samtidig som logikken bak hva disse funksjonene gjør er overførbare til videre arbeid med programmering.

4.2.4 Å lage kode

En annen sentral del av hva som forventes av elever i matematikk 1T, i henhold til eksamensveiledning er at de skal kunne lage kode selv (Utdanningsdirektoratet, 2024b). I

tillegg til at det er en forventing i vurderingssammenheng er det også naturlig når en snakker om programmering som et verktøy innenfor matematikk at en faktisk kan anvende dette verktøyet gjennom å kunne lage kode selv. På bilde 4.4 vises et eksempel på hvordan å lage kode oppgaver blir presentert i undervisningsopplegget. Ettersom opplegget er tiltenkt en rolle ved oppstart av temaet, som introduksjon og oppslagsverk får elevene en eksempelkode tilknyttet den koden de skal lage selv. Det at elever da gjenbruker kode fra eksemplene eller tidligere oppgaver er for meg helt greit og forventet, da oppgavene er lagt opp slik at eksempelkoden og tidligere oppgaver må endres i større eller mindre grad for å kunne løse det nye problemet – eventuelt at tidligere oppgaver bare er en del av en senere oppgave. Hvorvidt slik gjenbruk av kode fremmer læring eller bare benyttes som minste motstands vei av elevene, må naturlig nok avveies. Så lenge oppgavene er slik at elevene må forstå hvordan de kan gjenbruke deler av tidligere kode, og ikke minst hvordan de må bygge opp resterende kode rundt, har det potensiale til å fremme nytteverdi og legitimere tidligere arbeider.

Bilde 4.4: Celle om vilkår og tilhørende oppgave

```
#VILKÅR
import random as rnm #Her importeres pakken random, og gis forkortelsen rnm ved bruk av as.
#Grunnen til at vi ofte gir pakker forkortelser er for å effektivisere bruken

a=22
b=rnm.randint(0, 50) #variabel b defineres som et tilfeldig tall mellom 0 og 50, dette tallet vil endre seg hver
                     #..gang du kjører koden

if a<b: #hvis a er mindre enn b, gjør dette
    print('a er mindre enn b') #print er en innebygd funksjon som brukes til å returnere tekst, tall, etc.
elif a>b: #eller hvis a er større enn b, gjør dette
    print('a er større enn b')
else: #ellers, altså hvis ingen av de andre vilkårene stemmer, gjør dette. Hvilke tilfeller er dette?
    print('a er lik b')
```

```
a er større enn b
```

```
#OPPGAVE VILKÅR
#Lag en kode som ved bruk av if-setningen bestemmer om et tall er et partall eller oddetall.
```

Bilde 4.4: Øverste delen er et eksempel med kommentarer, tenkt som introduksjon av vilkår. «a er større enn b» er resultatet fra å ha kjørt koden, mens nederst finner vi en oppgave innenfor temaet vilkår.

Rent matematisk er ikke oppgaven på et spesielt høyt nivå, da det som kreves av elevene er at de vet forskjellen på odde- og partall. Tanken med oppgaven er at de skal bruke «randint», som introdusert i eksempelkoden, til å velge et tilfeldig tall på det intervallet de selv ønsker. Deretter er eneste krav til utførelsen at elevene bruker if-setningen til å bestemme hvorvidt det tilfeldige tallet er et odde- eller partall. Her er det flere muligheter, enten gjennom bruk av modulo betingelsen «%», som vil returnere 0 eller 1 for henholdsvis partall og oddetall

dersom en tar modulo 2. En annen mulighet er å sjekke om det tilfeldige tallet delt på 2 avrundet til et heltall er det samme som utgangspunktet. Til høyre ser vi to mulige løsningsmetoder for denne oppgaven.

```
tall=rnd.randint(0,100)
if tall%2==0:
    print(tall, "er et partall")
else:
    print(tall, "er et oddetall")

if tall/2==int(tall/2):
    print(tall, "er et partall")
else:
    print(tall, "er et oddetall")
```

4.2.5 Relasjon til didaktisk ingeniørvirksomhet

Illustrert i figur 4.1 blir de ulike oppgavetypene relatert til både fase to og tre av didaktisk ingeniørvirksomhet. Grunnen til dette er at oppgavene er blitt designet på bakgrunn av en forberedende analyse, samtidig som de er en sentral del av realiseringen av undervisningsopplegget og observasjoner tilknyttet denne realiseringen. Det virket derfor naturlig å foreta både en *a priori-analyse* og en *in vivo-analyse* i forbindelse med de ulike oppgavene. I avsnittene som følger vil de to analyseformene fra henholdsvis fase to og tre bli presentert.

Det var vanskelig å skulle ha klare forventninger til realiseringen av de ulike oppgavene, men en har naturlig nok et håp om at hvert fall noen av tankene og refleksjonene bak hvorfor opplegget ser ut som det gjør kommer fram. En forventning var riktig nok at eksempelkode skulle kunne hjelpe elevene med å komme i gang med oppgaver, enten ved at de ser hvordan det kan brukes eller ved at de gjenbruker og endrer den eksisterende koden. I tillegg var ulik grad av selvstendighet og håndtering av nytt verktøy en forventet skjevhet innad i klassen. Selvstendighet handler da om både hvordan en ønsker å løse oppgaver – alene, med medelever, hjelp fra læreren, osv. – men også hvordan håndtere problemer du ikke har sett før. Det ble også forventet at en må bruke tid på gjennomgang og oppsummering av de ulike oppgavene, både for å kunne ta tak i eventuelle uklare momenter, men også for at de som ikke fikk gjort seg helt ferdig skulle kunne se en måte å gjøre det på. Tar en steget med å fullføre eller vise fram oppgaver enda litt lenger, kan en også si at det er essensielt for at elevene skal få muligheten til å ha et fullverdig og nokså likt oppslagsverk til senere.

Analyse underveis i realiseringen av designet er noe som skal bli sett enda nærmere på i neste delkapittel (4.3), men i dette avsnittet vil hovedfokus være på realisering av oppgavene, og hvilke observasjoner som ble gjort i forbindelse med dette. En innledende observasjon var at det for mange elever var vanskeligere å komme i gang med spesielt det å kode selv, enn forventet. Her hadde du naturlig nok en stor spredning innad i klassen, hvor noen var meget raskt i gang, mens andre brukte lenger tid. Det kom først fram i datainnsamlingen, intervjuene, at eksempelkode og kommentarer var godt mottatt fra flere leirer – dette var noe

vanskelig å observere. Videre ble det observert at flere elever hadde problemer med å akseptere de enkleste løsningene, for eksempel at den hadde skrevet en fullverdig kode ved bruk av if-setningen, men viste ikke helt hvordan den skulle få inn et tall den kunne bruke. Grunnen til at det her omtales som å akseptere en enkel løsning, er at en her kan skrive eksempelvis «a=6» og teste koden, og så fort elevene fikk høre at dette var en mulighet, var det en slags «selvfølgelige kan en det» reaksjon fra flere. Dette kan naturlig nok være et resultat av en tenker at programmering er noe nytt og vanskelig, så da kan det ikke være så lett, men uansett var det en interessant opplevelse.

4.3 Gjennomføring av undervisningsopplegget

Realisering av undervisningsopplegget kom naturlig nok med et par utfordringer spesielt i oppstarten. Når en går direkte inn i en klasse, kan det være vanskelig å vite akkurat hvilke programmer de har lastet ned, hvorvidt de er vant til å bruke programmet og i denne sammenhengen om de hadde riktige ekstra nedlastninger som kreves for filene de skulle jobbe med. En stor del av oppstarten ble derfor å hjelpe elevene i gang, finne riktige filer og hjelpe de som ikke hadde nødvendige pakker med å få installert disse. Mange lærere vil nok kjenne seg igjen i at «ting tar tid», og introduksjon av et relativt nytt digitalt verktøy er ikke noe unntak.

Etter hvert som oppstarten var gjennomført, og alle elever hadde fått på plass riktig program med tilhørende pakker og filer startet den interaktive og alternerende formen for undervisning. Interaktiv i den forstand at elevene jobbet i akkurat samme filer som ble visst fram, og alternerende som nevnt i forbindelse med strukturen til opplegget, teori → eksempel(er) → Oppgave(r). Tidsaspektet kom igjen inn i strukturen på opplegget, da det tok noe lenger tid å få med så mange som mulig enn antatt, men ettersom strukturen bygget på ønske om at elevene skulle kjenne på mestring underveis og tilknyttet hvert enkelt undertema ble det naturlig å tillate at det tok litt lenger tid.

Underveis i realiseringen dukket det opp noen interessante observasjoner av en noe mer generell karakter enn de nevnt 4.2.5. Blant annet virket det som at friheten som ble gitt i løsning av oppgaver kunne virke mot sin hensikt, og for noen elever heller fremme usikkerhet knyttet til oppbygning og struktur i de innebygde funksjonene de skulle bruke. Eksempelvis kom det et spørsmål om hvorvidt if-setningen var begrenset til tre betingelser, i form av «if», «elif» og «else». Dette var i seg selv et veldig fint spørsmål, og det er selvfølgelig flott å kunne ta tak i slike usikkerheter, men det var allikevel interessant at med

programmeringsverktøyet framme falt valget på å spørre istedenfor å teste selv. Det trenger ikke ligge noe mer i det, enn at eleven ønsket å spørre, og at det var mer nærliggende enn å teste selv. Riktignok er problemløsningsstrategier og algoritmisk tenkning en del av kompetansemålet innenfor programmering i matematikk 1T (Utdanningsdirektoratet, 2020), og det som ble observert kan peke mot at dyrke disse komponentene i enda større grad kan være et steg mot å forenkle overgangen til programmering, i tillegg til at programmering i seg selv kan hjelpe i utviklingen av disse (Sevik, 2016, s. 13).

Ser en nok en gang tilbake til figur 4.1 er naturlig nok gjennomføring koblet opp mot tredje fase av didaktisk ingeniørvirksomhet, nemlig *realisering, observasjon og in vivo-analyse*. Datainnsamling er utelatt i denne fasen fordi den foregikk separat, og vil være fokusområde i neste kapittel. *In vivo-analyse* er allerede nevnt i forbindelse med de ulike kategoriene av oppgaver, men er igjen relevant for analyse av gjennomføring. Ser en tilbake på den *forberedende analyse* og *a priori-analyse* som grunnlaget hvilke begrensninger og forventninger som var tilknyttet realiseringen av undervisningsopplegget er det noen ulike aspekter og koblinger som dukker opp. Spesielt *a priori*, og forventninger presentert i delkapittel 4.1 om brukergrensesnitt, åpne oppgaver, struktur og nytteverdi står nå sentralt *in vivo*. Begrensninger knyttet til brukergrensesnitt ble som forventet, og førte med seg tilhørende utfordringer som nevnt tidligere med import av pakker og filbehandling. I tillegg ble selve opplegget seende noe mer rotete ut, da en ikke kunne bruke tekstblokker på samme måte som i Jupyter Notebook. Dette ble dog mottatt nokså greit av elevene, og det ble blant annet kommentert at det så ut som mer komplisert enn det faktisk var, på grunn av tekstmengde og at kommentarer i koden rettet opp. De åpne oppgavene og friheten til å utforske ble utslagsgivende på en noe annerledes måte enn forventet, da kommentarer i koden, eksempler og åpenhet i større grad ledet til fine spørsmål og diskusjoner enn hjelp med oppstart. Et eksempel på dette var et spørsmål tilknyttet hvor mange «if'er» eleven kunne bruke, eller om det var begrenset til «if», «elif» og «else». Istedenfor å sette i gang og prøve selv, ville noen helst få forklart litt mer. Dette varierte naturlig nok innad i elevgruppen, men sammenlignet med forventningen noteres det som et lite avvik. Strukturen i undervisningsopplegget virket på elevene som meningsfull, og den naturlige progresjonen i opplegget virket forståelig. Forventet nytteverdi er noe mer vanskelig å skulle svare på, da en gjerne skulle ha sjekket opp i hvor mange som tar i bruk opplegget ved en senere anledning. Forhåpentligvis kan elevene ta det i bruk på nytt, samt at både underveis og gjennom

intervjuene kom det fram at flere var positive til nettopp dette og satt pris på jobben som var gjort i forbindelse med filene – men det er vanskelig å si sikkert.

4.4 Begrensninger

Fra den innledende figuren (4.1) som relaterer de ulike stadiene av undervisningsopplegget sin utviklingsprosess til didaktisk ingeniørvirksomhet, er vi nå kommet til fjerde fase, nemlig *validitetsvurdering og a posteriori-analyse*. Grunnen til at delkapittelet er kalt begrensninger, handler om et overordnet mål om å kunne videreutvikle undervisningsopplegget, og derfor legge hovedvekten på elementer som har virket begrensende. Disse begrensningene kan en kjenne igjen som «constraint» fra ATD, og til dels som en transposisjonsprosess mellom «taught knowledge» og «learned/available knowledge» innenfor didaktiske transposisjoner (Chevallard, 2015; Chevallard & Bosch, 2020b).

Strømskag (2020a) omtaler *a posteriori-analyse* som en sammenligning mellom hypotesene en kom med i *forberedende analyse* og dataene fra *a priori-analyse*. *Validitetsvurderingen* til det didaktiske ingeniørvirksomheten baseres dermed på korrelasjonen mellom hypoteser og data/analyse, som nevnt i delkapittel 2.6. Hypotesene, eller forventningene, som jeg nevner i 4.1 er tilknyttet temaene brukergrensesnitt, åpne oppgaver, struktur og nytteverdi. Brukergrensesnitt var ikke noe jeg fikk gjort noe med, og selv om det ikke var optimalt, så løste det seg fint. Jeg forventet en viss grad av «wow-effekt» ettersom beskrivelser, kode og oppgavetekst ble rimelig kompakt, men det var ikke et stort hinder selv om jeg står fast på at Jupyter Notebook ville gitt en bedre opplevelse av undervisningsopplegget som helhet. Min hypotese tilknyttet de åpne oppgavene var det som overrasket meg mest, og er absolutt noe jeg kommer til å ta tak i under videreutvikling av opplegget. Naturlig nok er det stor spredning innad i klassen, basert på mine observasjoner, men for en del fler enn forventet var åpenheten et hinder mer enn en mulighet for å gjøre egne valg. I retrospekt kan det være en ide å øke graden av åpenhet etter introduksjon, eventuelt ha en form for nivådeling på hvor mye hjelp eleven ønsker med å komme i gang.

Struktur og nytteverdi henger noe sammen, da strukturen på opplegget er tiltenkt å ha en naturlig alternerende «gjennomgang → eksempel → prøve selv», og i tillegg kunne fungere som et oppslagsverk senere. På denne måten kan en si at elevene skulle se noe av nytteverdien i opplegget gjennom strukturen – dette ble også påpekt innledningsvis i gjennomføringen. Det var vanskelig å observere hva elevene syntes om strukturen på opplegget, med unntak av et par observasjoner hvor jeg så at de utnyttet seg av eksempelkode. Det kom derimot fram i

intervjuene at et flertall av elevene likte måten opplegget var strukturert på, mens noen helst ville jobbe uten gjennomgang, se flere eksempler, og lignende. Mine hypoteser traff altså til en viss grad, med spesielt åpne oppgaver som unntak. Når det er sagt, så er like opplevelser av et undervisningsopplegg innad i en elevgruppe veldig vanskelig å skulle oppnå. Du vil nesten alltid ha noen som helst skulle hatt det på en annen måte, så basert på at den didaktiske situasjonen er analysert med et ønske om å kunne videreutvikle opplegget – anser jeg validiteten i den didaktiske ingeniørvirksomheten som opprettholdt.

Kapittel 5

Analyse av intervjuer

Hovedkilden for datamateriale tilknyttet undervisningsopplegget er som nevnt intervjuer i fokusgrupper. I dette kapittelet skal disse dataene analyserer gjennom en prakseologisk referansemodell, hvor utsagn, diskusjon og kommentarer kategoriseres innenfor praksis- og logosblokken. Avslutningsvis kommer en noe løsere form for kvalitativ kategorisering, hvor fokuset er på å plukke opp og sette sammen dataene som ikke faller naturlig inn i referansemodellen, men som allikevel er relevant for videreutvikling og analyse av undervisningsopplegget.

5.1 Prakseologisk modell for analyse av intervjuer

Fra delkapittel 2.3 kjenner vi prakseologier som en generalisering av menneskelige handlinger i forbindelse med tilegnelse, distribusjon og produksjon av kunnskap (Bosch & Gascón, 2014). I denne studien står samtlige tre aspekter sentralt, ettersom det er blitt distribuert et undervisningsopplegg, elevene har tilegnet seg kunnskap og gjennom intervjuer og analyse skal et videreutviklet undervisningsopplegg produseres. Det er med andre ord datamateriale og egne refleksjoner tilknyttet gjennomføring som er grunnlaget for videreutvikling, med det overordnede målet å lage et undervisningsopplegg som møter variasjoner innad i en klasse, samt engasjerer og legitimerer plasseringen av programmering innenfor matematikk.

Modellen tilknyttet analyse av intervjuer vil fokusere på å knytte utsagn, svar og diskusjoner deltakerne har hatt opp mot prakseologi i form av praksis- og logosblokken. Gjennom tolkning av intervjudataene skal jeg kategorisere det som ble sagt innenfor type oppgaver (T), teknikker (τ), teknologi (θ) og teori (Θ). Det blir naturlig nok en god del tolkninger i den forbindelse, og jeg har derfor valgt å etablere forklaringer tilhørende de to blokkene for å kunne ha en tydelig modell og utgangspunkt for kategorisering. De fire forklaringene tar utgangspunkt i slik Chevallard (2019) omtaler de ulike delene av en prakseologi, og er systematisert i tabell 5.1 på neste side.

Tabell 5.1

KATEGORI	BESKRIVELSE
TYPE OPPGAVER	Kjente oppgaver/problemer med tilhørende teknikker
TEKNIKKER	Ulike måter å løse en oppgave eller et problem
TEKNOLOGI	Hvordan og hvorfor ulike teknikker fungerer
TEORI	Legitimering av motivasjon, teknologi og andre aspekter tilknyttet prakseologien

Tabellen over inneholder parafraserte beskrivelser av «de fire t'ene» basert på Chevallard (2019).

I delkapittelet om prakseologi så vi på et eksempel, tilknyttet hvordan oppgaver kan legges opp for å utfordre elever på deres prakseologiske ståsted. Hvorvidt en elev kjenner en teknikk tilhørende et tema, eller en spesifikk oppgavetype snakker en om type oppgaver. Hvis eleven ser flere måter å løse en oppgave på, eventuelt ulike problemløsningsstrategier for å komme i gang, er en innenfor teknikk kategorien. Beveger en seg over til logosblokken er teknologi diskusjon av ulike teknikker, det vil si at eleven i praksis skal kunne begrunne hvordan og hvorfor teknikken fungerer. Til slutt har vi teori, som begrepet er noe mer svevende, men er her legitimering av både faglige, filosofiske og motiverende faktorer tilknyttet prakseologien. Elever kan fort finne på å diskutere disse faktorene mindre direkte enn med for eksempel teknikker, ettersom hvilken nytteverdi og hvilke tanker hver enkelt elev har eller opplever kan være meget subjektiv. Det jeg derimot ønsker å se etter innenfor teori er faglig begrunnelse for hvorfor en teknikk passer bedre i en gitt situasjon enn en annen, for eksempel at det vil hjelpe deg i de to neste deloppgavene, selv om det er en noe mer tungvint teknikk å starte med.

5.2 Datamateriale tilknyttet praksisblokken

I dette delkapittelet skal jeg se på utsagn, diskusjoner og svar tilknyttet praksisblokken av prakseologi. Under følger en del om type oppgaver og en om teknikker, som blant annet vil inneholde utsnitt fra de transkriberte intervjuene. Som nevnt innledningsvis i kapittelet innebærer kategoriseringen av datamateriale en form for tolkning, da besvarelsene ikke alltid har en tydelig relasjon til en kategori som beskrevet i tabell 5.1. Der det er nødvendig vil jeg derfor fokusere på å få fram hvordan jeg tolker det som er sagt, og argumentere for hvorfor dataene passer i akkurat den kategorien.

5.2.1 Type oppgaver

Som en del av oppvarmingen i intervjuprosessen valgte jeg å ha søkelyset på hva slags forkunnskaper elevene hadde med programmering. Her kom det fram at et fellestrekk var kjennskap til «scratch» og «microbit» fra ungdomsskolen, men at denne typen oppgaver ikke ble ansett som noe de kunne bruke inn mot tekstbasert programmering. Under ser vi to eksempler fra transkripsjon av intervjuene som peker på skille mellom blokk- og tekstbasert programmering.

> *«Jeg har litt blokkprogrammering gjennom scratch, men det er ikke helt programmering på samme måte som python.»*

> *«Blokkprogrammering går jo ut på å finne mønster og informasjon ut fra blokker. Python har ikke samme mønster, du må finne alle stegene selv – så det er ikke like selvforklarende.»*

Blokkprogrammering kontra tekstbasert skal omtales mer i 5.2.2, da elevene uttrykte at som teknikker var de to rimelig forskjellige. Som oppgavetyper ble blokkprogrammering omtalt som mønstergjenkjenning med færre valgmuligheter, mens tekstbasert føltes mer som et eget språk hvor de måtte bygge blokkene selv. I tillegg hadde noen av deltakerne vært på kodekurs, i form av et ekstratilbud, men konsensus her for de det gjaldt var at de ikke husket altfor mye.

Matematikken i programmeringsoppgavene havnet også innenfor denne kategorien, og ble blant annet omtalt som «enkel» - det var med andre ord en kjent løsningsmetode og oppgavetype.

> *«Det var ganske enkle mattestykker i seg selv, det var samtidig nytt, men tror fortsatt det var veldig lærerikt.»*

Noen deler av programkoden ble også utpekt som kjent, da det var en tydelig kobling mellom kjent matematikk og hva en ville oppnå med programmet, for eksempel funksjoner. En naturlig vei videre fra type oppgaver innenfor matematikk, var å undersøke deltakerne sin aksept ovenfor oppgaver i programmering de like gjerne kunne gjort for hånd eller med geogebra. Disse oppgavene er nokså spesielle innenfor programmering i matematikk, da de kan være velkjente og «enkle» med allerede innarbeidede teknikker, mens en kan få større utfordringer med å løse de med et programmeringsverktøy. Jevnt over var aksepten for denne typen oppgaver nokså unison, i form av at deltakerne ga uttrykk for at å løse kjente

matematiske problemer kan være god trening i programmering. En gruppe sa direkte at det ville vært en dårlig oppgave, men også der ble det omtalt som god øving:

«Spørsmål: Er det demotiverende å få oppgaver du like gjerne kunne gjort på andre måter enn programmering?

Da føler jeg nok at det er et dårlig spørsmål, men jeg bare aksepterer det.

Det er god øving i hvert fall.»

Beveger en seg mot krysningen mellom type oppgaver og teknikker ble spesielt samarbeid og interaktivitet tatt fram som kjente arbeidsmåter – og til dels teknikker, mer om det i 5.2.2. Grunnen til at jeg velger å omtale samarbeid og interaktivitet under type oppgaver er at gjennom disse to virket det på deltakerne som at en kunne senke inngangsterskelen, spesielt i forbindelse med oppgaver som ligner på noe en har gjennomgått felles og at det medfølger en stor grad av frihet. Denne friheten baserer seg på å kunne gjøre endringer, samt rett og slett eksperimentere med en egen kodefil, for så å se hva som skjer.

«Jo, fordi det er veldig enkelt å gå tilbake hvis du ikke har fått med deg noe, fordi det liksom står i feltet på en måte.»

«Jeg likte at vi fikk prøve selv, og skrive selv. Det tror jeg hjelper, så det likte jeg.»

Denne lekenheten og utforskende naturen elevene ga uttrykk for, gjør at jeg kan argumentere for at selv om elevene mangler kunnskap om teknikkene som kreves, kan denne kunnskapen utvikles gjennom kategorien type oppgaver. I dette legger jeg spesielt fokus på praksis, i og med at det å arbeide med teknikker gjennom oppgaver, kan virke noe omvendt av hva en vanligvis ville gjort, altså å lære seg en teknikk som en så anvender i praksis. Det kan derimot hjelpe elevene med å komme i gang, og ikke minst føre til spørsmålet; *hvorfor funket dette?*

5.2.2 Teknikker

Spesielt for denne delen av analysen er at jeg velger å ta i bruk skille mellom to ulike teknikker i tråd med hvem som er hovedkarakter i situasjonen med tilhørende prakseologi (Bosch & Gascón, 2014). Skille går mellom det jeg har valgt å kalle «didaktisk teknikk», basert på Gueudet et al. (2020), og den mer tradisjonelle definisjon av teknikk fra tabell 5.1. Grunnen til at dette er interessant innenfor analysen generelt, men kanskje spesielt teknikker, er bevisstgjøring på skille mellom læreren sine teknikker for å skape en meningsfull læringssituasjon og elevene sine teknikker i forbindelse med spesifikke oppgaver og problemløsning mer generelt.

Den første overordnede teknikken som ble omtalt var tilknyttet elevene sitt eksisterende forhold til programmering gjennom blokkprogrammering. Denne typen programmering var en kjent teknikk for de fleste, men som nevnt i 5.2.1 så ikke elevene noen spesiell sammenheng eller bruksområde som knyttet denne teknikken opp mot tekstbasert programmering. Videre var det å gjenbruke og/eller endre kode en teknikk som ble omtalt både i forbindelse med eksemplene, men også tilknyttet koden elevene lagde selv – som illustrert av sitatet under.

> *«Det tar en del tid å sette opp, men når du først har det kan det jo brukes igjen senere.»*

Når det er snakk om å gjenbruke kode kan det være nyttig å se på hvorvidt dette er en teknikk som fremmer læring eller bare er enkleste utvei. For å skulle kunne svare på dette valgte jeg å konstruere to oppgavesituasjoner hvor denne teknikken kan være anvendbar. Den første situasjonen handler om en tilsvarende oppgave som koden er brukt til allerede, i denne situasjonen risikerer en at elever «bare vet at den funker» - uten egentlig å forstå hvordan eller hvorfor. Det kan derfor være lurt å være bevisst på hvilken kode og hvilke oppgaver elevene har tilgjengelig og har jobbet med fra før. På denne måten kan en ta steget inn i den andre tenkte situasjon som handler om å gjenbruke kode en større oppgave eller endre eksisterende kode for å passe til oppgavespesifikasjoner. Denne situasjon stiller helt andre krav til forståelse og ferdigheter tilknyttet teknikken gjenbruk – og hvordan den kan brukes hvis eleven står fast eller har glemt noe, som illustrert under.

> *«Jo, fordi det er veldig enkelt å gå tilbake hvis du ikke har fått med deg noe, fordi det liksom står i feltet på en måte.»*

Innledningsvis etablerte jeg et skille mellom omtalte teknikker basert på hvem som var hovedkarakter i prakseologien. Samarbeid og feilsøking var to teknikker som ble omtalt innenfor begge «definisjonene» av teknikker. Samarbeid ble omtalt som teknikk av en gruppe i forbindelse med oppstart og usikkerhet, da de mente det skulle mer til for å stå fast hvis en var flere som jobbet sammen på en oppgave.

> *«Å jobbe sammen, for hvis jeg jobber alene kan det være vanskelig å forstå, men hvis man jobber i grupper og diskuterer kan det være lettere å forstå.»*

Som en didaktisk teknikk handlet tilbakemeldingene om å legge enda mer til rette for samarbeid, og ikke minst være enda klarere på at det er fullstendig tillatt. Feilsøking hadde en nokså lik omtale som samarbeid, da det ble nevnt som en fin teknikk for å etablere forståelse for hva ulike deler av eksisterende eller konstruert kode gjør.

> *«Finne ut hvorfor det blir feil, og fikse feilen, hvis det bare var et likhetstegn eller et kolon som var feil og bare fikse det.»*

En slags framprovosert feilsøking, hvor en endrer på deler av en eksisterende kode for å se hva som skjer kunne derfor vært interessant som en enda tydeligere form for oppgave. I tillegg ble feilsøking kommentert som noe jeg kunne tatt i bruk i enda større grad inne i undervisningsopplegget, som en slags teknikk for å arbeide med forståelse av struktur, nøyaktighet og syntaks.

Spesielt tilknyttet didaktiske teknikker diskuterte vi i intervjuprosessen bruk av såkalte «unplugged» aktiviteter, det vil si arbeid med programmering uten et dedikert programmeringsverktøy (Brackmann et al., 2017, s. 65). Flytskjema, pseudokode og andre mer fysiske visualiseringer av kodeprinsipper ble møtt med interesse, og en annerkjennelse av det kan hjelpe med forståelse.

> *«Jeg tenker at sånne flow-charts kan være veldig nyttig for å vise fram hvordan tenkingen funker.»*

Det er dog flere spørsmål her om både tidsbruk, institusjonaliserte krav inn mot eksamen og med det trinn-plassering. Slike aktiviteter omtales som en fin introduksjon til programmering, med fokus på grunnleggende konsepter innenfor programmering og utvikling av algebraisk tenkning (Brackmann et al., 2017). Selv om hovedfokus i studien presentert av Brackmann et. al fokuserer på elever fra trinn under videregående skole, kan det uansett være noe å ha i bakhånd, dersom for eksempel løkker blir vanskelig å visualisere inne i programmeringsverktøyet.

5.3 Datamateriale tilknyttet logosblokken

Som for datamateriale tilknyttet praksisblokken deles dette delkapittelet opp i teknologi og teori. Logosblokken kan dog være noe mer diffus å hente ut datamateriale tilknyttet, og spesielt siden undervisningsopplegget er introduksjon av et relativt nytt tema. Ser en tilbake på beskrivelsene av teknologi og teori fra tabell 5.1, handler det om diskusjon og begrunnelse av valgt teknikk(er). Selv om jeg ikke forventet diskusjon av teknikker (teknologi) med sterkt begrunnet teori, var det allikevel interessant å se på hvordan elevene indirekte diskuterte grunnlag for teknologi og teori gjennom spesielt oppbygning av oppgaver, syntaks og andre elementer fra undervisningsopplegg og programmeringsverktøy.

5.3.1 Teknologi

Et av de klareste eksemplene på en diskutert teknikk som kom fram i intervjuene var hvorfor elevene følte at blokkprogrammering ikke var altfor relevant i forbindelse med oppstart av tekstbasert programmering. En spesiell forskjell som ble nevnt var at blokkprogrammering ble omtalt som en form for mønstergjenkjenning, mens tekstbasert i større grad var et språk hvor du måtte bygge blokker selv. Vi er med andre ord tilbake der at mange av elevene ikke helt fant koblingen mellom blokk- og tekstbasert programmering – noe som kanskje ikke er rart i forbindelse med syntaks, men oppbygning og struktur i kode samt generell kunnskap om hvordan for eksempel løkker fungerer, så kunne kunnskapen vært overførbar.

> *«Spørsmål: Dere har kanskje holdt på med scratch?*
>
> *Alle: Ja!*
>
> *Føler dere at det har hjulpet dere?*
>
> *Ikke egentlig*
>
> *Det var litt annerledes, siden der kan du sette inn brikker, mens her må du skrive kode.*
>
> *Mhm*
>
> *Det har vært en overgang?*
>
> *Alle: Ja!*
>
> *Det føles egentlig som noe helt nytt»*

> *«Blokkprogrammering går jo ut på å finne mønster og informasjon ut fra blokker. Python har ikke samme mønster, du må finne alle stegene selv – så det er ikke like selvforklarende.»*

Ser en litt større på hva som er målet med undervisningsopplegget, så er arbeid med teknologi, samt diskusjon av og forståelse for hvilken teknikk elevene skal ta i bruk, noe jeg mener er helt essensiell for legitimering og introduksjon av tekstbasert programmering. Spørsmål som hvordan kan du «dele opp» matematikken og implementere den på en måte som gjør at koden du lager fungerer på en god måte kan anses som en form for teknologi, da det krever kjennskap til teknikker innenfor matematikk og matematisk programmering, som så må diskuteres og velges. For å kunne diskutere ulike teknikker for implementering er det en fordel å i første omgang ha kunnskap om syntaks, tilknyttet programmeringsspråket, men også hvordan matematikk skrives. Dette ble uttalt som en utfordring og noe nytt av elevene.

«Det er ikke noe feil med det, men noe jeg syntes er vanskelig er hvordan man bruker symboler på en annen måte enn i matte.»

«Programmering blir jo kalt et språk, fordi det har mange regler og struktur. Fleksibiliteten du har er ny for meg, og de små delene av koden som gjør en spesiell ting. Ulik struktur kan gjøre at koden gjør noe helt annet, men siden du har så mange variabler og tall syntes jeg det går veldig mye innenfor matte.»

I tillegg finnes det veldig mange løsningsmetoder innenfor programmering, skal du velge å lage en funksjon eller et frittstående script? Hvilke innebygde operasjoner passer inn? Løkker, betingelser, vilkår, liste-funksjoner, og lignende. Denne forståelsen av både den underliggende matematikken og strukturen i en velfungerende kode ble omtalt av fokusgruppene på litt ulike måter. Eksempelet under viser hvordan begrepet brukes i forbindelse med oppstart av oppgaver, og at manglende forståelse gjør det vanskeligere å sette sammen delene av en oppgave til en fungerende kode.

«Spørsmål: Er det oppgaver eller eksempler som er vanskelig?

Jeg tror det er mer å forstå.

Putte ting sammen til en oppgave.

Vite hvor man skal begynne?

Ja.»

Videre ble spesielt begrepet forståelse brukt sammen med programmering, og gjerne i en kontekst som omhandlet bruksområder tilknyttet matematikk. Med andre ord ble ofte teknologi dratt fram som grunnlaget for å kunne utnytte verktøyet programmering på en bedre måte – og da spesielt matematisk.

5.3.2 Teori

Som med teknologi er det noe vanskelig å skulle forvente altfor mye innenfor teori underveis i et undervisningsopplegg som er ment som introduksjon. Jeg har derfor valgt å fokusere på relasjoner innad i logosblokken, og utsagn som omtaler hvordan elevene har arbeidet med og kan arbeide med teori. Fra tabell 5.1 kjenner vi teori som legitimering av motivasjon, teknologi og generelt hvorfor vi gjør som vi gjør. Starter vi helt overordnet ble nytteverdien av teknikkene innenfor programmering satt i sammenheng med et sterkt fremtidsfokus, og at det å mestre programmering kan være nyttig for videre studier, arbeidsliv, osv.

> *«Det er jo mer og mer som blir databasert, så programmering kan sikkert være fint for å forstå det som ligger bak mye av det du finner på nettet og sånn. Og kan kanskje hjelpe deg i framtida med å bedre forstå.»*

Sammen med teknologi utgjør teori grunnlaget for forståelse for hvordan og hvorfor vi gjør en handling, samt refleksjoner og tanker bak nytten av denne handlingen. En oppgavetype hvor dog teori i form av nytteverdi til dels er på plass, er de hvor elevene i bunn og grunn lager små kalkulatorer ved bruk av funksjoner. Her vil struktur og operasjoner sett i en programmeringssammenheng være mindre utfordrende, samtidig som matematikken er kjent. Gjennom denne typen oppgaver ønsker jeg at elevene kan beholde det kjente i større grad, og på den måten legge til rette for at teorien er til stede – til tross for at programmering er nokså nytt, og viktigheten av å innarbeide en motivasjon for hvorfor vi driver med programmering vil da i stor grad falle på læreren, når elevene ikke har et etablert forhold til matematisk programmering.

Tilbake til relasjoner innad i logosblokken, er det utvilsomt en sammenheng mellom kunnskapen og forståelsen elevene har tilknyttet programmering, og hvorvidt teknologi og teori er til stede. Den forståelsen som blir omtalt kan settes inn i både teknologi og teori, med et tydelig skille på hvor praktisk utførelse eller valg av teknikk er. En forståelse med forankring i teknologi kan for eksempel gi en god praktisk løsning gjennom valg av fornuftig teknikk. Forståelse innenfor teori vil da for eksempel være aksepten elevene viste tilknyttet oppgaver de hadde teknologisk grunnlag for å løse med andre teknikker enn programmering, men allikevel velger å gå løs på med programmering som verktøy.

> *«Vi jobbet jo med matematiske funksjoner nå, rett før vi kom inn hit, og satt egentlig og lagde små kalkulatorer. Og jeg føler jo at koding relaterer veldig til matematikk sånn generelt, og når du gjør koding bare for å bruke det i matte-sammenheng, så er det jo enda mer relatert.»*

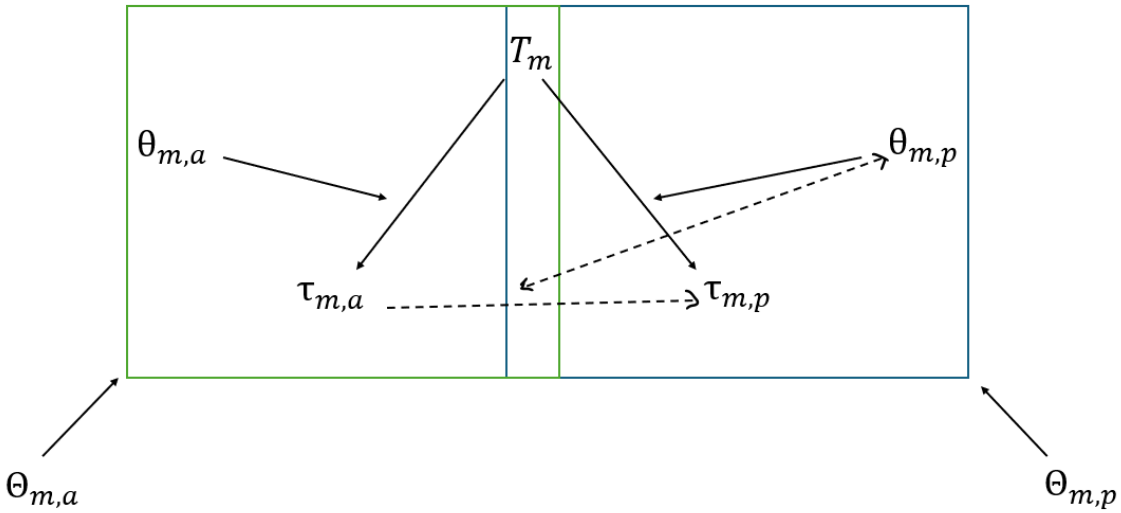
Fra sitatet over ser vi at til tross for at eleven har en fint fungerende kalkulator, og sannsynligvis kunne løst oppgaven med den eller for hånd aksepteres det at programmering skal benyttes. Den nytteverdien og ikke minst relasjonen til matematikk virker å være med på å underbygge aksepten, og ikke minst legitimere oppgaven fra et teoretisk ståsted.

5.4 Korrelasjon mellom praksis- og logosblokk

Fra 5.2 og 5.3 har vi sett på de ulike delene av praksis- og logosblokken som utgjør en prakseologi. Her har vi sett koblinger innad i de to blokkene, men også sett og diskutert

koblinger på tvers – for eksempel teknikk og teknologi. I dette delkapittelet skal vi se nærmere på denne korrelasjonen mellom og innad i blokkene, samt presentere en modell som representerer disse sammenhengene. Modellen er tenkt som et visuelt hjelpemiddel for å bedre koble sammen utsagn fra intervjudeltakerne, men er også sentral i den prakseologiske videreutviklingen av undervisningsopplegget.

Figur 5.1



Figur 5.1: En modellert beskrivelse av relasjonen mellom type oppgaver, teknikker og teknologi med utgangspunkt i en matematisk oppgave (m – matematisk). De to resterende prefiksene betegner henholdsvis algebraisk (a) og programmering (p).

Fra figur 5.1 ser vi den tenkte korrelasjonen mellom praksis- og logosblokken. Komponent for komponent er utgangspunktet en matematisk oppgave (T_m), som springer ut to teknikker hvorav en er algebraisk ($\tau_{m,a}$) og en tilknyttet programmering ($\tau_{m,p}$). I dette steget fra oppgavetype til teknikk finner vi teknologi, som igjen er tilknyttet den som skal gjøre oppgaven sin evne til å svare på hvordan og hvorfor den gjør oppgaven, og velge teknikk(er) basert på dette. Her er det nok en gang naturlig å skille mellom algebraisk teknologi ($\theta_{m,a}$) og den tilknyttet programmering ($\theta_{m,p}$) – og spesielt i en situasjon hvor det er snakk om introduksjon av tekstbasert programmering, og elevene vil da sannsynligvis ha et annet teknologisk grunnlag i forbindelse med algebraiske teknikker. Andre del av logosblokken, teori, finner vi tilknyttet hele handlingen eller oppgaven. Teori legitimerer oppgaven sin relevans og nytteverdi, og kan sånn sett være sentral i det første valget – hvorvidt en velger algebraisk teknikk og teknologi, eller går rett på programmering. Videre er de stiplede linjene

illustrasjoner av tilknytninger på tvers av algebra og programmering, og vi finner disse mellom de to gruppene av teknikker, samt teknologi tilknyttet programmering i overgangen mellom algebraisk- og programmeringsteknikk. Grunnen til at den stiplede linjen fra programmeringsteknologi og inn til overgangen mellom de to teknikkene går begge veier, er at en i første omgang må kunne gjøre om den algebraiske teknikken til noe innenfor programmering, før en setter det inn i programmeringsteknikken. For eksempel hvis vi ser tilbake på bilde 4.2, og elevene skulle lage koden som sjekker nullpunkter for et andregradspolynom. Da vil en kunne si at det å bruke rotutrykket til å bestemme antall nullpunkter, for så å kunne regne ut disse, er den algebraiske teknikken – mens «if-setningen» med tilhørende betingelser, og etablering av en funksjon dersom ønskelig er programmeringsteknikken. For å kunne utnytte den algebraiske teknikken i programmering, må elvene kunne diskutere plassering av uttrykk og hvordan utnytte «rot» som betingelse – derav koblingen fra $\tau_{m,\,a}$ til $\theta_{m,p}$. Deretter følger så å diskutere og utføre resten av nødvendig kode, illustrert av modell som $\theta_{m,p}$ til $\tau_{m,\,p}$.

5.5 Sammenligning av fokusgruppene

Et nevnt fellestrekk blant fokusgruppene var programmering i form «scratch» og «microbit», samt en unison enighet om at koblingen mellom denne blokkprogrammeringen og den tekstbaserte varianten «python» ikke var altfor tydelig for dem. Her var det spesielt oppbygning og struktur som ble pekt på, når de snakket om hvorfor det var så ulikt. Når det er sagt startet allikevel fokusgruppene, og deltakerne innad, med nokså forskjellige utgangspunkt ettersom tre av fire grupper hadde en eller flere deltakere som hadde vært med på eller hatt undervisning/frivillige kodekurs for videregående skole.

Et annet interessant fellestrekk var aksepten for programmering sin plassering i matematikkfaget. Hvorvidt dette er aksept i form av at elevene skjønner at dette er noe de ikke får velge selv eller ei, kan naturlig nok diskuteres. Det som dog underbygger at de så nytteverdien av denne plasseringen var at alle gruppene omtalte programmering som et potensielt nyttig verktøy innenfor matematikk. Unntaket var en elev som rett ut uttalte at den ikke likte matematikk, men syntes programmering var artig. Begrepet forståelse kom spesielt i fokus når vi diskuterte verktøyet programmering og hvordan de opplevde spesielt oppgavene i undervisningsopplegget. Samtlige grupper nevnte forståelse, og to av fire grupper valgte heller å sette søkelys på manglende forståelse enn å kalle oppgavene/opplegget vanskelig eller utfordrende. Dette kan tolkes til at deltakerne her selv tror at det er oppgaver de skal få til etter hvert som de har arbeidet noe mer med temaet, men de samme gruppene hadde et ønske

om større grad av hjelp med oppstart for å kunne bistå i arbeidet med forståelse. En annen gruppe kalte oppgavene vanskelig, men understrekte at de fikk til det meste – så det kunne virke som de følte seg utfordret, men at det ikke var for vanskelig. Den siste gruppen, hvor to av tre deltakere hadde hatt en del mer programmering enn sine medelever omtalte vanskelighetsgraden som fin repetisjon.

Det å kunne skrive kode selv, endre på eksisterende kode og ha alt tilgjengelig var også noe som ble omtalt på en eller annen måte av de fire gruppene. Tilknyttet forståelse som nevnt i forrige avsnitt, var dette noe elevene satt pris på – da flere deltakere påpekte at det å kunne gjøre endringer i eller fikse kode selv var en fin måte å jobbe mot ønsket nivå av forståelse. Matematikken som var til stede i undervisningsopplegget virket å være på et nivå elevene var vant med, noe som var tiltenkt, da jeg ønsket at de skulle se sammenhengen mellom matematikken og verktøyet programmering. Felles for gruppene var at den matematiske kunnskapen som krevdes for å løse oppgavene ikke hindret fremgangen. Dette var ønskelig fra min side, i utviklingen av opplegget, ettersom det er introduksjon av tekstbasert programmering innenfor matematikk 1T, og derfor virket det naturlig å holde det matematiske nivået slik at elevene i det aller minst har kjennskap til det fra før.

5.6 Relasjon til matematikk

Ettersom dette er en matematikkdidaktisk studie skal jeg i dette delkapittelet se enda nærmere på hvordan deltakerne følte opplegget var relatert til matematikk, og hvordan denne relasjonen kom fram gjennom oppgaver, eksempler, og lignende. I tillegg skal jeg se på hvordan verktøyet programmering omtales i intervjuene, og da spesielt som et verktøy i forbindelse med læring innenfor matematikk. Noe av det som blir tatt opp her vil også ha vært nevnt i den prakseologiske analysen av intervjuene, men hovedfokus i dette delkapittelet vil være omtale av matematikken i praksis og som en del av en matematisk programmeringsoppgave, samt overgangen mellom eksempelvis algebra og programmering som illustrert i figur 5.1.

Det var ganske naturlig et skille mellom introduksjon av grunnleggende ferdigheter som betingelser og vilkår, og de noe mer avanserte funksjonene og løkkene når det kom til deltakerne sin opplevelse av relasjonen til matematikk. Ettersom funksjoner er et begrep de kjente godt til fra matematikken ble dette omtalt av flere som svært relevant, og det kan da være viktig å understreke at funksjoner i programmering ikke trenger å være akkurat det samme som i matematikk. Betingelser og vilkår ble ikke omtalt på samme måte, og det kan ha

noe med at matematikken i disse oppgavene og eksemplene kanskje virket uinteressant og meget grunnleggende – for eksempel å sjekke regnestykker gjennom returnering av «true/false». For meg ble dette understreket av mangelen på tilbakemeldinger tilknyttet disse temaene, samt uttalelsen under.

> *«Det var ganske enkle mattestykker i seg selv, det var samtidig nytt, men tror fortsatt det var veldig lærerikt.»*

Spesielt med verktøyet programmering, og hvordan en kan motivere forståelse for og bruk av dette verktøyet innenfor matematikk, var at alle fokusgruppene uttalte at de ser nytten av verktøyet og at det har bruksområder innenfor matematikk. Spesielt refleksjoner tilknyttet hva deltakerne kan oppnå etter hvert som de når et høyere kunnskapsnivå innenfor programmering virket som en felles oppfatning. En videreføring av verktøyet sin relevans innenfor matematikk bygger på selve matematikken som skal anvendes når deltakerne bruker programmering. Det var en form for aksept for de fleste typer matematiske innhold og oppgaver, selv de som like gjerne kunne vært løst på andre måter. Dette kan ha noe med at deltakerne selv var bevisst på at introduksjon av programmering kan være greit å gjøre i trygge rammer. Samtidig uttalte spesielt en gruppe at oppgavene og matematikken som tas i bruk burde være relevant for verktøyet programmering, samtidig som den generelle aksepten for også disse typene oppgaver kom fram.

> *«Spørsmål: Er det demotiverende å få oppgaver du like gjerne kunne gjort på andre måter enn programmering?*
>
> *Da føler jeg nok at det er et dårlig spørsmål, men jeg bare aksepterer det.*
>
> *Det er god øving i hvert fall.»*

Dette er for så vidt noe jeg er helt enig i, men det er interessant å observere at de mindre relevante matematikkoppgavene innenfor programmering aksepteres i så stor grad, uten nødvendigvis å reflektere over hvordan en kunne løst oppgavene på en annen måte – det var ingen av fokusgruppene som selv spurte hvorfor de skulle gjøre akkurat dette i programmering, når det er enklere med en annen teknikk. En siste relasjon som ble nevnt mellom matematikk og programmering er oppbygningen som et slags eget språk med tilhørende logikk. Denne uttalelsen er inne på det som kalles metakognisjon, og er i Sevik (2016) et argument for hva programmering kan hjelpe elevene med å utvikle på et mer tverrfaglig nivå. Det å være bevisst på egen tankeprosess, og ikke minst utvikle hvordan en angriper nye situasjoner og evaluerer egen innsats, refleksjoner, og lignende, kan være et

fokusområde sammen med algoritmisk tenkning innenfor matematikk allerede før en tar fatt på programmering.

5.7 Direkte tilbakemeldinger på undervisningsopplegget

Avslutningsvis i kapittel 5 om analyse av intervjuer har jeg valgt å inkludere et delkapittel om de mer direkte tilbakemeldingene på undervisningsopplegget. Tanken bak er at disse uttalelsene kan fort falle utenfor den prakseologiske analysen og trenger heller ikke handle om relasjonen mellom matematikk og programmering. Det er dog min mening at når målet med studien er å kunne videreutvikle undervisningsopplegget på en god måte, er det viktig å også inkludere de mer generelle oppfatningene og ikke bare de med analytisk eller teoretisk verdi med grunnlag i teoretisk rammeverk og metodologi. Det vil naturlig nok være en viss grad av subjektivitet i noen av tilbakemeldingene, men mitt inntrykk er at elever sjeldent er alene om slik tilbakemelding, selv om personen ved siden av eller i fokusgruppen mener noe annet.

Det var generelt sett positive tilbakemeldinger på hvordan opplegget var strukturert, og spesielt likte deltakerne at en kunne prøve selv i samme fil som eksempelkode og gjennomgang foregikk. Jeg tror fortsatt undervisningsopplegget ville fungert bedre i «Jupyter Notebook», basert på de estetiske mulighetene en har til å skille tekst, kode og bilder. Utseende på selve filen ble med andre ord en utfordring på grunn av overgangen fra «Jupyter Notebook» til «MU», og at de ulike filene jeg måtte inndele temaene i så ut som litt mye, fordi skille mellom forklarende tekst, kommentarer i koden og faktisk kode gled over i hverandre. Videre var det noen deltakere som ønsket flere eksempler, mens andre ønsket flere oppgaver – dette kan være tilknyttet utfordringene med åpenhet og oppstart, det vil si den åpenheten jeg tenkte skulle gi frihet i oppstarten, men endte opp heller som et sted for spørsmål og refleksjon. Antall eksempler og oppgaver kan også være relatert til at et fellestrekk for tre av fokusgruppene var at de synes opplegget var utfordrende, selv om kun en av gruppene direkte brukte ordet vanskelig i sin beskrivelse av en eller flere oppgaver. Tips og hint knyttet til oppstart av oppgaver var også noe som ble nevnt, og noe jeg ønsker å ta tak i.

Kapittel 6

Videreutvikling av undervisningsopplegget

I dette kapittelet, videreutvikling av undervisningsopplegget, starter vi på diskusjonsdelen av oppgaven. Denne studien inneholder med andre ord to deler diskusjon, nemlig en konkretisert form som diskuterer videreutvikling av det utprøvde undervisningsopplegget, samt kapittel 7 som inneholder diskusjon av resultater fra studien. Resultat er et felles stikkord for de to diskusjonsdelene, da videreutviklingen er et resultat av arbeidet i studien og med datamateriale, på samme måte som diskusjon av data knyttet opp mot tidligere forskning, forskerspørsmål og problemstilling.

En essensiell del av denne studien er videreutvikling av det analyserte og testede undervisningsopplegget. Hovedmålet er å skape et best mulig opplegg, hvor et ønske om interaktivitet, engasjement og nytteverdi skal være synlig både for elever og lærer. Fra delkapittel 5.5 til 5.7 har vi sett på fellestrekk mellom fokusgruppe intervjuene, relasjoner mellom programmering og matematikk, samt de mer direkte tilbakemeldingene på undervisningsopplegget, mens spesielt i 4.2 og 4.3 har vi sett hvordan observasjoner fra klasserommet stemmer overens med tredje fase av didaktisk ingeniørvirksomhet. Med disse delkapitlene friskt i minne har jeg fått noen overordnede tilbakemeldinger på hvordan jeg kan oppnå en mulig bedre læringssituasjon. Jeg ønsker å ta tak i disse tilbakemeldingene innledningsvis, før en analyse med forankring i det teoretiske rammeverket ATD (Chevallard & Bosch, 2020a), samt *forberedende-* og *a priori-analyse* fra didaktisk ingeniørvirksomhet (Strømskag, 2020a), vil ta oss med enda dypere inn i videreutviklingen av studien sin hovedkarakter.

Fra de avsluttende delkapitlene av kapittel 5 har vi blant annet sett at elevene omtaler relasjonen mellom matematikk og programmering spesielt gjennom kjente begreper, som funksjoner. I tillegg har det vært en gjennomgående aksept for de fleste type oppgaver, løsbare med andre verktøy eller ei, plassering av programmering innenfor matematikk og et generelt fokus på at å lære mer kan gjøre nytteverdien større, både innenfor matematikk og mer fremtidsrettet aspekter, som studier, digital hverdag, og lignende. Matematikken i seg selv ble ikke omtalt som spesielt utfordrende, dette var til dels meningen, i alle fall innledningsvis – så kunne det vært veldig interessant å høre hva de syntes om de litt vanskeligere oppgavene som kommer mot slutten av opplegget. Det er dog noe jeg ønsker å ta tak i, gjennom å introdusere enda mer programmeringsrelatert matematikk tidlig i opplegget. Her trenger fortsatt ikke

matematikken å være spesielt utfordrende, men jeg ønsker at elevene enda tydeligere får se hvilke muligheter programmering gir og at utfordringen blir å sette i gang tankeprosessen rundt hvordan matematikken de allerede kjenner til kan anvendes innen programmering. Dette for å fange de elevene som sitter med følelsen av at å bruke programmering til å sjekke eller utføre et og et regnestykke virker lite meningsfullt.

Interaktivitet og strukturen i opplegget ble dratt fram som positivt, noe som var veldig gledelig da interaktiviteten og det faktum at elevene skulle ha hele kodefilen og muligheten for å bruke denne senere var noe av motivasjonen for å lage undervisningsopplegget på denne måten. I tillegg var det interessant at begrensningene knyttet til det visuelle innenfor editoren MU ble tatt fram, noe jeg mener underbygger bruker av Jupyter Notebook når en gjennomfører også det videreutviklede undervisningsopplegget.

6.1 Didaktiske transposisjoner

Så fort det er snakk om introduksjon av et tema er det naturlig at det har skjedd flere didaktiske transposisjoner (Chevallard & Bosch, 2020b), og da ikke bare fra originalkunnskap, men sannsynligvis også fra målkunnskapen en ønsker og oppnå. Ettersom programmering fører med seg veldig mange løsningsmetoder av samme problem, innebygde funksjoner som ikke nødvendigvis stiller krav til å forstå alt som ligger bak og utallige muligheter til å gå videre mot noe mer utfordrende ligger det en god del valg bak utvikling og videreutvikling av et slikt undervisningsopplegg. I dette delkapittelet skal jeg se på hvilke didaktiske transposisjoner som er gjort i utviklingen og videreutviklingen, og argumentere for hvorfor disse får være med videre eller må endres/fjernes.

6.1.1 Betingelser og vilkår

Et av de første valgene som ble tatt i forbindelse med undervisningsopplegget var hvordan jeg ønsket å starte det. Valget falt på betingelser og vilkår noe som virket naturlig fordi disse bygger på en logikk som står sentralt i programmeringsarbeid – for eksempel hvis dette er oppfylt, gjør dette, hvis ikke, gjør dette. Den didaktiske transposisjonen fra de helt generelle definisjonene og bruksområdene for betingelser og vilkår, og rett inn i talleksempler var et valg som ble gjort på bakgrunn av at matematikken skulle stå sentralt i opplegget. Selv om relasjonen til matematikk er noe jeg mener må være tilstede når en programmerer i matematikk, er det å stille spørsmål til verden en sentral del av ATD (Chevallard, 2015). Etter å i tillegg ha fått spørsmål om en var begrenset til én «if» når en arbeider med «if-setningen» valgte jeg derfor å inkludere en tekstcelle med litt mer generell informasjon om spesielt vilkår.

Bilde 6.1: Videreutviklet celle om vilkår

Bilde 6.1: Generell informasjon om oppbygning av vilkår i form av «if-setningen».

Tanken bak å holde det så generelt som «..gjør dette» var å åpne for at elevene skulle kunne stille spørsmål til verden og reflektere over andre bruksområder, også innenfor programmering. Kanskje noen elever klare å tenke at vilkår kan sannsynligvis brukes på nettsider for å bestemme hva som skjer når du klikker på et spesifikt sted. Det er med andre ord relasjonen mellom programmering og matematikk som skal stå sterkest i undervisningsopplegget, men jeg ønsker allikevel ikke å frarøve elevene muligheten til å utforske andre deler av programmeringskunnskapen.

6.1.2 Lister

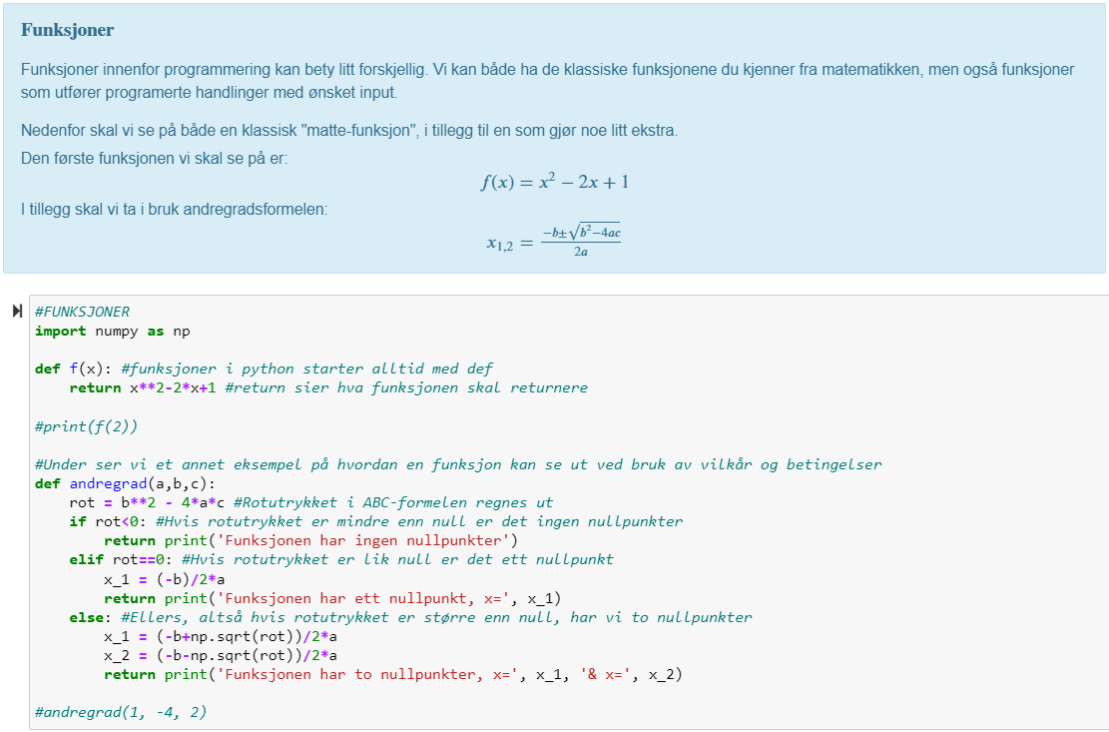
Det å inkludere lister som del av undervisningsopplegget er i seg selv et valg jeg har gjort på bakgrunn av at det gir muligheter for å kunne jobbe med relevante matematiske problemer innenfor programmering. Gjennom å kunne etablere lister med mye datamateriale og håndtere dette på kort tid, åpner en opp muligheter for å kunne kjenne på beregningskapasiteten til selv en standard datamaskin. Innenfor lister har jeg gjort flere didaktiske transposisjoner i utviklingen av originalt opplegg, som er blitt med i videreutviklingen. Disse kommer i form av blant annet å introdusere de innebygde funksjonene «linspace» og «arange» tidlig, istedenfor et større fokus på å lage sine egne lister. Jeg har også gjort et valg om å droppe lagring i lister, utvidelser av lister, med mer, da tanken bak å inkludere liste-begrepet er å gi elevene et verktøy som forhåpentligvis kan hjelpe med å se nytteverdien av programmering innenfor matematikk. Når det er sagt inkluderes hvordan de kan lage en liste med klammeparenteser, og hvordan hente ut et enkeltelement fra lister – men bortsett fra disse er fokus i større grad på bruksområder enn generell kunnskap.

Lister kan også omtales innenfor den didaktiske transposisjon som den rekonstruksjonen av kunnskap vi kjenner som kompetansemål og lignende innenfor skoleverket, og spesielt plassering i akkurat 1T. Fra egen erfaring med undervisning av programmering og diskusjon med andre faglærere er lister en større del av programmering innenfor matematikk R1, men basert på kompetansemålene er det vanskelig å si akkurat hvor lister er tiltenkt en rolle. En snakker altså her om et valg som skjer i overgang mellom «Noosfæren» og læringsinstitusjonene (Chevallard & Bosch, 2020b), men min oppfatning er at ved å inkludere lister på denne måten – gjennom hovedfokus på bruk i praksis – kan en styrke prakseologien gjennom teknikker for håndtering av datasett, i tillegg til teori gjennom nytteverdi og legitimering (Chevallard & Bosch, 2020a).

6.1.3 Funksjoner

Et sentralt didaktisk moment når det er snakk om funksjoner, spesielt innenfor matematisk programmering, er skille mellom funksjonsbegrepet fra matematikk og hva en funksjon kan være innenfor programmering. Transposisjonen handler da om hvor mye fokus en ønsker og ha på tilfeller som ikke er like ligner på funksjoner elevene har sett før, og funksjoner som bygger på problemer innenfor programmering – og ikke nødvendigvis matematikk. I videreutvikling ble en sentral del å understreke akkurat dette, at funksjoner innenfor programmering kan være de klassiske funksjonene, som $f(x) = x^2 - 2x + 1$, men også andre prosesser og handlinger som kan være interessante og bruke igjen senere.

Bilde 6.2: Videreutviklede celler om funksjoner



Funksjoner

Funksjoner innenfor programmering kan bety litt forskjellig. Vi kan både ha de klassiske funksjonene du kjenner fra matematikken, men også funksjoner som utfører programerte handlinger med ønsket input.

Nedenfor skal vi se på både en klassisk "matte-funksjon", i tillegg til en som gjør noe litt ekstra.

Den første funksjonen vi skal se på er:

$$f(x) = x^2 - 2x + 1$$

I tillegg skal vi ta i bruk andregradsformelen:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```python
#FUNKSJONER
import numpy as np

def f(x): #funksjoner i python starter alltid med def
    return x**2-2*x+1 #return sier hva funksjonen skal returnere

#print(f(2))

#Under ser vi et annet eksempel på hvordan en funksjon kan se ut ved bruk av vilkår og betingelser
def andregrad(a,b,c):
    rot = b**2 - 4*a*c #Rotutrykket i ABC-formelen regnes ut
    if rot<0: #Hvis rotutrykket er mindre enn null er det ingen nullpunkter
        return print('Funksjonen har ingen nullpunkter')
    elif rot==0: #Hvis rotutrykket er lik null er det ett nullpunkt
        x_1 = (-b)/2*a
        return print('Funksjonen har ett nullpunkt, x=', x_1)
    else: #Ellers, altså hvis rotutrykket er større enn null, har vi to nullpunkter
        x_1 = (-b+np.sqrt(rot))/2*a
        x_2 = (-b-np.sqrt(rot))/2*a
        return print('Funksjonen har to nullpunkter, x=', x_1, '& x=', x_2)

#andregrad(1, -4, 2)
```

Bilde 6.2: Den blå cellen er generell informasjon om funksjoner og eksempelet som kommer. Eksempelet inneholder nå en vanlig matematisk funksjon, og nullpunkt-eksempelet fra originalt undervisningsopplegg – for å understreke at funksjoner kan opptre på ulike måter innen programmering.

Fra bilde 6.2 ser vi hvordan jeg har valgt å presentere funksjoner, både som en klassisk $f(x)$ og som en programmert handling. Funksjonen *andregrad* er utvilsomt relatert til matematikk, men ser en på selve utførelsen og hva som returneres er noe annerledes enn hva elevene er vant med. Det generelle ønske om å skape enda større kontraster mellom informasjon i form av tekst og kode kommer også inn her, gjennom å inkludere den blå cellen med tekst og funksjoner/formler på den tradisjonelle formen elevene er vant med.

6.1.4 Løkker

Innenfor temaet løkker er valg tilknyttet hva du skal inkludere nokså låst, da *for*- og *while-løkker* er sentralt ikke bare i matematikk 1T – beste kontrollen av dette er å se på gitte eksamensoppgaver, men også tenke at en skal legge grunnlaget for videre studier av matematisk programmering, spesielt innenfor numeriske metoder. Det en riktignok kan velge er hvordan og hva en presenter innenfor løkker, noe som igjen leder oss til transposisjonsprosessene mellom «Noosfæren», læringsinstitusjoner og elevgrupper. Ser en på

prosessen fra start til slutt påvirkes altså «Noosfæren» av både utdanningsdirektoratet, i form av eksamensoppgaver og kompetansemål, samt at en også på universiteter bruker disse løkkene, så i et studieforberedende løp virker det som en naturlig del av undervisningen. Beveger en seg så inn i læringsinstitusjonen, og ser på hva jeg har lagt opp til at elevene skal få se på og/eller undervist, er det nok en gang praksis og logikk som står sentralt. Praksis i form av å vise fram grunnleggende kjennetegn ved løkker, og bruksområder som å summere en tallrekke og å finne nullpunktene til en funksjon av grad tre (se «sammensatt eksempel», vedlegg 2).

Bilde 6.3: Videreutviklede celler om løkker

Løkker

Løkker brukes i programmering for å lage en gjentagende prosess. Der vilkår fungerer sånn at hvis dette gjør det, hvis noe annet gjør det og ellers gjør sånn, fungerer løkker slik at så lenge $a < 100$ gjenta dette eller så lenge $a \epsilon [0, 100]$ gjenta dette.

Det er i hovedsak to typer løkker vi bruker, **while** og **for**, hovedforskjellen her at den første tar i bruk en betingelse - dvs. så lenge $a < 100$, mens den andre bruker et intervall.

```
#LØKKER
import numpy as np

a=7
i=0
b=53

#a+=1 betyr det samme som a=a+1, og ny a vil oppdateres for hver runde i løkka
while a<b:#while er en løkke som gjentas helt til en betingelse er møtt
    a+=1 #for hver runde i løkka legges det til en i a
    #print(a)
    i+=1 #"i" kan i dette tilfellet f.eks være en måte å telle antall runder løkka kjører


A=7
B=53
k=0
for j in range(A,B): #For er en annen type løkke hvor variabel j i et intervall (in range(start, stop, steg)) fra A til B
    k+=j #k+=j vil si at alle tall mellom A og B summeres sammen
    #print(k)
```

Bilde 6.3: Blå celle er generell informasjon om løkker og tilhørende eksempel, mens selve eksempelet er beholdt fra det utestede undervisningsopplegget.

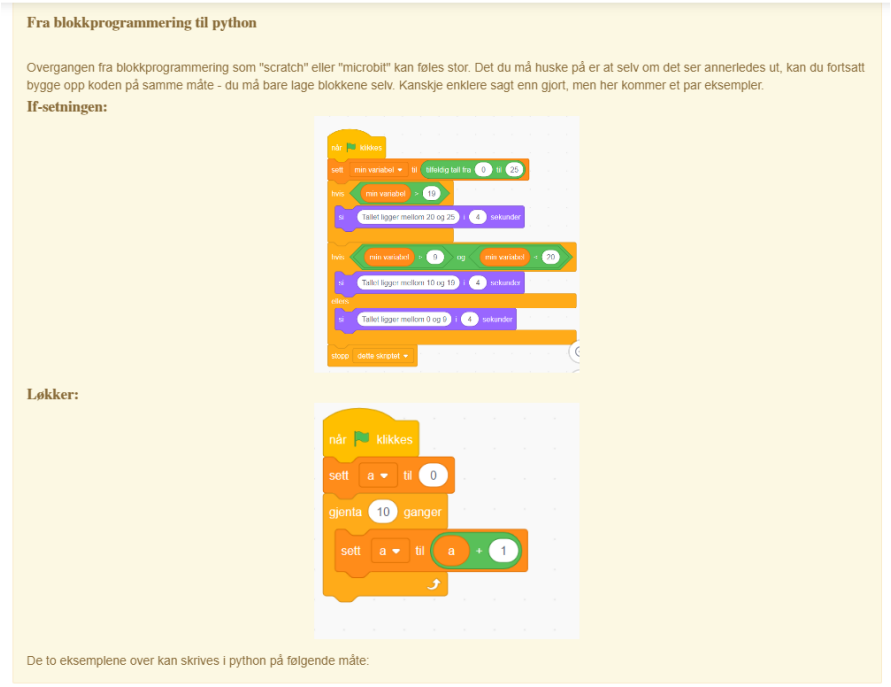
Fra bilde 6.3 ser vi at oppsettet er nokså likt som for funksjoner, og ettersom løkker for mange er noe mer utfordrende rent logisk [kilde], starter jeg veldig greit ved å illustrere den repeterende gangen i en løkke gjennom å legge til 1 for hver runde i løkken. Fokuset er rett og slett på å bli vant med løkker, tilhørende variabel (i/j) og hva som faktisk skjer runde for runde i løkken. Det er med andre ord et ganske grunnleggende utbytte jeg ønsker at elevene skal lære eller ha tilgjengelig (se figur 2.3), noe jeg mener er godt forenelig med et undervisningsopplegg ment som introduksjon, og ikke minst med praktisk bruk innenfor matematikk som kanskje den viktigste komponenten for opplegget.

6.2 Type oppgaver (T)

Når det gjelder type oppgaver var det spesielt hvordan utnytte forkunnskaper, begrepet forståelse og innarbeiding av teknikker gjennom oppgaver som ble hovedfokus i videreutvikling. Utnyttelse av forkunnskaper stammer fra en unison enighet i intervjuprosessen at elevene ikke følte blokkprogrammering var relevant eller nyttig når de skulle ta steget inn i den tekstbaserte verdenen. Forståelse som begrepet, er noe svevende og kan bety så mangt, men basert på kategoriseringen jeg introduserte under utvikling av undervisningsopplegget (kapittel 4) er dette begrepet noe som går igjen i intervjuene og har koblinger til de ulike oppgave kategoriene. Det er da spesielt kodeforståelse som oppgavetype, jeg mener mange av intervjudeltakerne refererer til som løsningen på hvordan de skal få enda mer ut av programmering og se flere bruksområder i matematisk sammenheng. Det siste punktet jeg nevnte innledningsvis er innarbeiding av teknikker gjennom oppgaver, noe elevene ga uttrykk for at de satt pris på – det å få prøve selv – og da er vi inne på en teknikk som sannsynligvis innarbeides aller best i praksis, nemlig hvordan komme i gang med oppgaver.

Det å skulle utnytte forkunnskaper, og i større grad bygge bro mellom blokkprogrammering og tekstbasert programmering var noe som virket veldig interessant for videreutviklingen av undervisningsopplegget. En konsekvens av å forsøke og løfte fram blant annet de strukturelle og logiske likhetene fra kjente oppgaver med innarbeidede teknikker i form av blokkprogrammering, kan jo bli at overgangen og oppstarten blir noe mykere for elevene. Måten jeg valgte å gjøre det på, var å inkludere to eksempler fra «Scratch», for så å vise hvordan disse kunne vært gjort i «Python», samtidig som jeg skrev litt om hva som er overførbart og hva som blir ulikt.

Bilde 6.4: Eksempel på blokkprogrammering inkludert i videreutvikling



Bilde 6.4: Inkludert eksempel fra blokkprogrammering gjennom editoren «Scratch».

Selv om elevene kanskje ikke er helt vant med begrepene vilkår og løkker, er dette eksempler som forhåpentligvis føles gjenkjennbare ut, med en editor og et innhold de skal ha sett før. De to bildeeksemplene følges opp med følgende:

Bilde 6.5: Blokkprogrammering eksempel skrevet om til tekstbasert kode

```
#If-setningen:
import random as rnm

min_variabel=rnm.randint(0,25)

if min_variabel>19:
    print("Tallet ligger mellom 20 og 25")
elif min_variabel>9 and min_variabel<20:
    print("Tallet ligger mellom 10 og 19")
else:
    print("Tallet ligger mellom 0 og 9")

#Løkker:

a=0
i=0
while i<10:
    a=a+1
    i=i+1
    print(i)

print('a=',a)
```

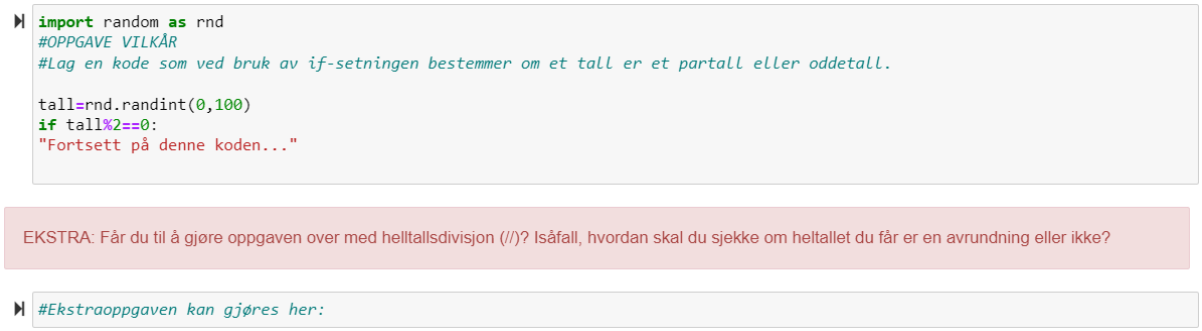
Her ser vi at strukturen er nokså lik, men at du i python må bygge blokkene slik at de gjør det du ønsker. I tillegg kommer python med en god del bibliotek eller pakker, som f.eks "random", disse må importeres for å kunne bruke innholdet i dem. Vi skal bli bedre kjent med flere pakker senere, som blant annet numpy. For løkke-eksemplene, eller gjenta som det heter i scratch, bruker vi her while-løkke og istedenfor å si gjenta 10 ganger - introduserer vi en "telle-variabel" som vi kaller "i", og så lenge denne er mindre enn 10 gjentas handlingen inne i løkken.

Bilde 6.5: Omskriving av eksempel fra bilde 6.4 til tekstbasert programmeringsspråk.

Bilde 6.5 viser hvordan eksemplene fra «Scratch» kunne vært gjort i «Python», med fokus på å bruke samme variabelnavn og forklare det jeg tror blir de første spørsmålene som dukker opp i den gule tekstcellen. Det var noe overaskende at ingen av elevene som deltok i intervjuene så deres forkunnskaper innenfor blokkprogrammering som en ressurs inn mot tekstbasert programmering, og det virker derfor naturlig å forsøke og gjøre de eksisterende likhetstrekkene enda tydeligere for elevene.

Forståelse innenfor T kan tenkes å være en forutsetning, når type oppgaver beskrives som et kjent institusjonalisert problem med tilhørende teknikk (Chevallard, 2019), må elevene naturlig nok ha et forhold til problem og teknikk – og da enten forståelse av den innarbeidede metoden eller faktisk forståelse av matematikken bak. Det er dog kobling mot de to oppgavekategoriene kodeforståelse (4.2.1) og matematisk forståelse (4.2.2) som skal være hovedfokus nå, mens andre aspekter ved begrepet som forståelse av teknikker blir mer relevant innenfor teknologi, eller når det er snakk om teori, forståelse i form av nytteverdi og legitimering. Kodeforståelse som en type oppgave vil forhåpentligvis kunne komme enda tydeligere fram gjennom forsøket på å utnytte forkunnskapene tilknyttet blokkprogrammering – spesielt med tanke på struktur og logikk. Dette er også oppgaver hvor tanken er å bygge opp under det siste punktet som ble nevnt innledningsvis, nemlig å innarbeide teknikker gjennom oppgaver. Matematisk forståelse derimot traff beskrivelsen som type oppgave noe bedre, da flere av intervjudeltakerne uttrykte at matematikken i seg selv ikke var det mest utfordrende, og dermed var disse oppgavene enklere å komme i gang med. Når det gjelder innarbeiding av teknikker gjennom oppgaver, ble en sentral del av videreutviklingen å finne en balansegang mellom å la elevene oppdage teknikker selv, gjennom eksempelkode og oppgaver, og å hjelpe dem i gang – slik at de nådde fram til denne oppdagelsen. Under er et eksempel på hvordan jeg har valgt å hjelpe elevene i gang, og ikke minst appellere til gjentatt bruk av teknikken, samt matematisk argumentasjon.

Bilde 6.6: Videreutviklet oppgave med inkludert start av kode



Bilde 6.6: Elevene gis to linjer med kode for å hjelpe dem i gang med oppgaven.

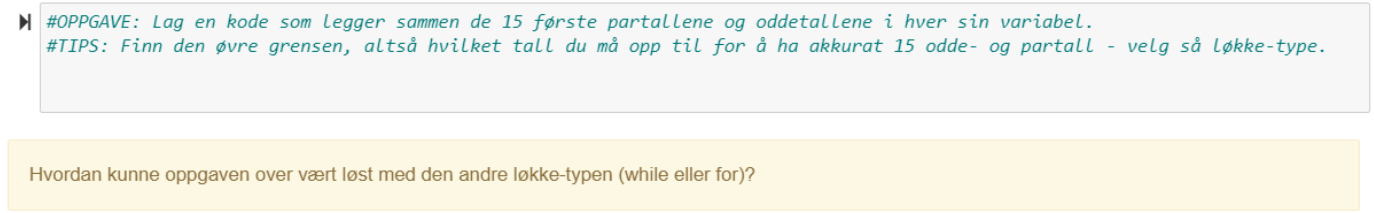
Tanken bak oppgaven på bilde 6.6 er å hjelpe elevene i gang med å utforske teknikker tilknyttet «if-setningen». Elevene må altså finne ut om $tall\%2 == 0$ betyr at det er snakk om et partall eller oddetall, for så å fullføre koden på ønsket måte. Ekstraoppgaven inneholder deretter en liten matematisk nøtt, som må utføres og kontrolleres på en hensiktsmessig måte. De skal altså løse akkurat samme oppgave, men med heltallsdivisjon istedenfor modulo som operator. Den matematiske nøtten leder også til en form for kodeforståelse, spesielt med tanke på struktur, da en må utføre en sjekk av heltallsdivisjon for å kunne ha en betingelse å bruke i «if-setningen».

6.3 Teknikker (τ)

Fra bilde 6.4 og praten om innarbeiding av teknikker gjennom oppgaver, bærer denne videreutviklingen preg av noe overlappende faktorer og grep som er tatt innenfor praksisblokken. En kan utvilsomt si at blokkprogrammering er en kjent teknikk, men i delkapittel 6.2 var det altså hvilke type oppgaver og hvordan hjelpe elevene med å bruke denne kjente teknikken til å løse problemer når tekstbasert programmering er teknikken i fokus.

En sentral del av videreutvikling ble å ta tak i problematikken tilknyttet oppstart av oppgaver som kom fram gjennom både observasjon og intervjuer. Jeg måtte med andre ord ta i bruk noen didaktiske teknikker, og da spesielt hva jeg kunne gjort i forkant, samt legge til rette for at elevene kunne ta grep underveis for å opprettholde engasjement og fortsette å se nytteverdien av oppgavene (Gueudet et al., 2022). Det var spesielt tre ulike didaktiske teknikker jeg valgte å ta i bruk, hvorav to var tilknyttet tips enten i oppstart eller hvis de skulle stå fast, og den siste handlet om å gi elevene starten av koden – og da ha mulighet til å utfordre dem senere, se bilde 6.6.

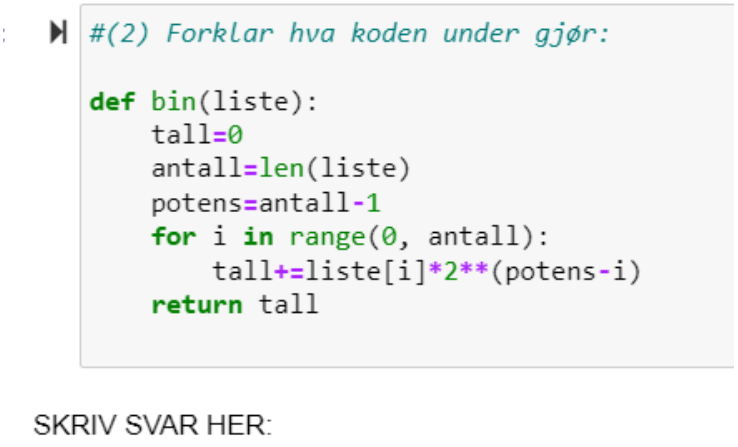
Bilde 6.7: Tips til oppstart



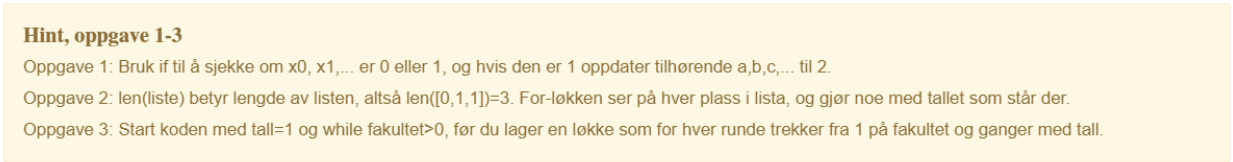
Bilde 6.7: Elevene gis tips til hvordan de kan starte tankeprosessen i oppbygningen av koden.

Originalt var det kun oppgaveteksten som var en del av dette problemet. Ettersom det er en oppgave tilknyttet «løkke-delen» av undervisningsopplegget er dette i fokus, og elevene skal allerede ha gjort en oppgave som hjelper de med å sortere partall fra oddetall. Gjennom valg av øvre grense, samt hvilken type løkke de vil bruke kan de her komme i gang, slik at utfordringen blir i større grad å oppdatere to variabler for hver runde av løkken tilknyttet korrekt betingelse – partall eller oddetall. I tillegg valgte jeg å legge til spørsmålet som kan ses i gul celle som en utfordring, men også for å hinte om at denne kan fint løses med både «for-» og «while-løkke».

Bilde 6.8: Tips underveis



Bilde 6.9: Hint, nederste celle i undervisningsopplegget



Bilde 6.8 og 6.9: Første bilde viser en av tre oppgaver som kommer mot slutten av undervisningsopplegget, mens bilde 6.9 inneholder hint elevene kan se på om de vil.

Oppgavene mot slutten av undervisningsopplegget har en egen celle med hint, se bilde 6.9. Denne cellen er plassert slik at den ikke ses når eleven først kommer til oppgavene, men det står skrevet at det finnes hint nederst i filen. Grunnen til at det er løst på denne måten er todelt, da det er ønskelig at eleven reflekterer og forsøker uten hint i første omgang. Det kan dog være ganske store forskjeller på hvordan elever i en klasse håndterer dette, og for de som enten syntes det ser veldig vanskelig ut og/eller prøver, men setter seg fast kan opplysningen om hint medføre at eleven faktisk ga det et forsøk først – i visshet om at hvis den sto fast kunne den se på hintene. Et annet tilfelle er de elevene som uten hint ikke engang ville forsøkt å løse oppgaven, disse kan velge å gå rett til hintene, og forhåpentligvis komme i gang på denne måten.

Bilde 6.10: Start på kode

```
Du kan finne hint for oppgave 1 til 3 nederst (under begrepsliste).
```

```
import numpy as np
#OPPGAVER
# (1) Lag en funksjon som regner ut verdien til en binær sekvens med seks plasser.
def binary(x0, x1, x2, x3, x4, x5):
    a=0
    b=0
    c=0
    d=0
    e=0
    f=0

    bin_val=a**5+b**4+c**3+d**2+e+f
    return bin_val
```

Bilde 6.10: Et annet eksempel på hjelp i form av start på kode, men denne gangen er elevene gitt start og slutt, og må arbeide på koden i mellom «f=0» og «bin_val».

Bilde 6.10 er også en del av oppgavene mot slutten av undervisningsopplegget, og har derfor i tillegg til at starten på koden er foreslått, hint som vist fram av bilde 6.9. Originalt inneholdt denne oppgaven kun teksten «Lag en funksjon som regner ut verdien til en binær sekvens med seks plasser», hvor binære sekvenser blir diskutert i en tekstcelle før oppgavene – se undervisningsopplegget som vedlegg. Essensen her er å ta bort deler av oppgaven for at elevene isteden kan fokusere på hvordan de håndterer variablene de er gitt, og hvordan de løser oppgaven, eksempelvis med bruk av vilkår og betingelser inne i funksjonen.

Bilde 6.11: Løsningsforslag binær sekvens med seks plasser

```
import numpy as np
#OPPGAVER
# (1) Lag en funksjon som regner ut verdien til en binær sekvens med seks plasser.
def binary(x0, x1, x2, x3, x4, x5):
    a=0
    b=0
    c=0
    d=0
    e=0
    f=0
    if x0==1:
        a=2
    if x1==1:
        b=2
    if x2==1:
        c=2
    if x3==1:
        d=2
    if x4==1:
        e=2
    elif x5==1:
        f=1

    bin_val=a**5+b**4+c**3+d**2+e+f
    return bin_val

print(binary(0, 1, 0, 0, 1, 0))
```

```
18
```

Bilde 6.11: Forslag til hvordan oppgave fra bilde 6.10 kan løses.

På bilde 6.11 ser vi et forslag til hvordan oppgaven kunne vært løst med repeterende bruk av «if», en teknikk de nå skal kjenne til. Det er spesielt to grunner til at jeg har valgt å gi elevene noe å starte fra i akkurat dette problemet, nemlig lokale kontra globale variabler og en mer matematisk felle i form av 0^0. Definering av a, b, c, osv. utenfor funksjonen vil resultere i en feilmelding tilknyttet de lokale variablene som er en del av funksjonen, dette er for så vidt et fint eksempel med tanke på å vise fram forskjellen mellom lokal og global. Videre er det sannsynlig at mange av elevene ville tatt i bruk formelen fra tekstcellen om binære sekvenser, og derfor skrevet $a^5 + b^4 + c^3 + d^2 + e^1 + f^0$, noe som ville resultert i at «f-leddet» alltid ble en. Ettersom elementer i binære sekvenser som er satt til null har verdien null, og potensen kun blir relevant dersom det står et ettall. Nok en gang er dette noe som er viktig å diskutere, både med tanke på klassiske feller innenfor matematisk programmering, men også for å arbeide med en grunnleggende logikk som utvilsomt er viktig innenfor programmering og matematikk.

Utover disse didaktiske teknikkene, var det spesielt koblingen mellom teknikken blokkprogrammering og tekstbasert programmering, som nevnt innledningsvis i 6.3 og mye omtalt i 6.2. Det ble her nokså kraftig overlapping mellom type oppgaver og teknikker, men ettersom koblingen mellom de to var såpass fjern for elevgruppen jeg intervjuet ble det naturlig å ta tak i dette under videreutviklingen for å bedre kunne utnytte de teknikkene elevene kjenner fra før gjennom både eksempler og oppgaver.

6.4 Teknologi (θ)

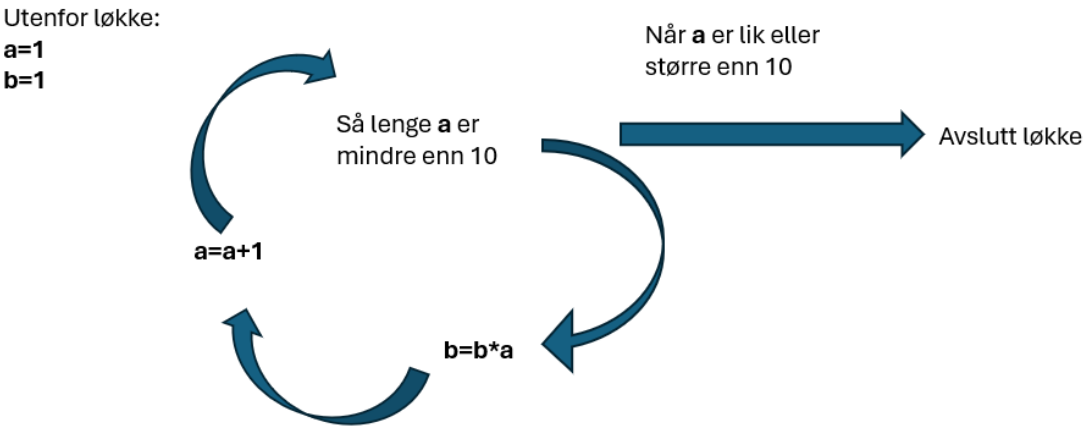
Et overordnet mål med strukturen på spesielt oppgavene i det originale undervisningsopplegget var å legge til rette for utforskende arbeid, og at elevene selv skulle kunne finne fram til at innen programmering i matematikk er det veldig mange mulige løsningsmetoder. Idealisert var altså tanken at elevene skulle kunne oppdage og diskutere ulike teknikker underveis, og gjennom diskusjonsoppgaver – med andre ord å arbeide med teknologi i praksis. Den praktiske biten med å skulle teste ut selv, prøve å lage egen kode med eksempler og kommentarer lett tilgjengelig ble strukturelt sett godt mottatt, men problemer og utfordringer tilknyttet oppstart ble nevnt som noe elevene gjerne ville ha mer hjelp med. I videreutviklingen ble det derfor naturlig å fjerne noen av løsningsmetodene, som for eksempel hvilken type løkke elevene skulle velge, men med et ønske om å opprettholde den utforskende delen av undervisningsopplegget. Fra bilde 6.7 ser vi et eksempel på hvordan jeg ønsker å ta vare på den utforskende delen av opplegget, samtidig som elevene får råd til hvordan de kan starte oppgaven. I tillegg inneholder den oppgaven en ekstraoppgave hvor elevene skal løse akkurat samme problem, men med den andre typen løkker – «while» eller «for». Denne ekstraoppgaven bygger på et ønske om at elevene skal kunne se og føle på forskjellene mellom de to teknikkene eller typene løkker, og på den måten være bedret rustet til å diskutere deres neste valg av løkker. Et annet eksempel med et tilsvarende ønske er fra bilde 6.6, men her er det i større grad diskusjon av matematiske teknikker utført som kode som står sentralt. Hvorvidt elevene setter mest pris på en løsning med rest etter divisjon (modulo) eller heltallsdivisjon, med en tilhørende sjekk av at heltallet de startet med kan oppnås ved å multiplisere med to, gjenstår å se. Poenget er dog at to ulike teknikker i form av ulike operatorer, kan med små forskjeller i strukturen og inneholdt i koden gi løsning på samme problem.

De to eksemplene fra bilde 6.6 og 6.7 kan også relateres til den lærersentrerte prakseologien som er omtalt i forbindelse med didaktiske teknikker. Hovedfokus i denne sammenheng er altså hvordan legge til rette for teknologisk utforskning, med diskusjon og legitimering av ulike teknikker tilknyttet det samme problemet som fokusområde. Samtidig er det viktig innenfor denne prakseologien at en som lærer er bevisst på at dette undervisningsopplegget er ment som introduksjon av tekstbasert programmering. I henhold til de utfordringene jeg møtte med å ha veldig åpne oppgaver må en derfor avveie hvorvidt dette går på bekostning av introduksjon av de grunnleggende ferdighetene som må til for å bedre kunne utforske senere, eller om det er nyttig å la elevene oppdage selv. Ser en på læreplan (Utdanningsdirektoratet,

2020) og paradigme med stille spørsmål til verden fra ATD (Chevallard, 2015) er utforskende arbeid som engasjerer og motiverer elevene til å se lenger enn kun hva som kreves for å løse en oppgave utvilsomt essensielt i en didaktisk situasjon. Poenget med å da skulle presentere ulike løsningsmetoder og teknikker i ulike steg, istedenfor å la dem oppgave disse selv blir et valg hvor jeg ønsker å treffe en så stor del av klassen som mulig. En side ved introduksjon som ikke må undervurderes er at mulighetene for å ta neste steg, og la elever som ønsker og kan gå videre utenfor det som er planlagt, er lett tilgjengelig i den forstand at de bare så vidt har startet sin reise med å studere monumentene slik Chevallard (2015) oppfordrer til.

Videre blir det naturlig å snakke om koblingen mellom blokkprogrammering og tekstbasert programmering også innenfor teknologi, da det er snakk om to teknikker som gjerne skulle vært diskutert og sammenkoblet på en tydeligere måte enn hva elevene ga uttrykk for. Fra bilde 6.4 har jeg forsøkt å gjøre koblingen enda tydeligere, gjennom å spille på den kjente teknikken blokkprogrammering og eksemplifisere hvordan dette kunne vært gjort med tekstbasert programmering. En form for teknologisk arbeid som kunne vært interessant å innlede undervisningsopplegget med er en «unplugged aktivitet» som flytskjema. Grunnen til at jeg omtaler dette som teknologisk arbeid er at ved å introdusere «unplugged» som en tredje teknikk, i tillegg til blokk- og tekstbasert programmering, vil elevene kunne diskutere ulike teknikker gjennom visualisering (flytskjema), det kjente (blokkprogrammering) og det ukjente (tekstbasert programmering). Et eksempel på hvordan et flytskjema kunne sett ut er presentert under.

Figur 6.1: Flytskjema for forklaring av fakultet ved bruk av «while-løkke»



Figur 6.1: Eksempel på flytskjema som kan benyttes i forbindelse med introduksjon av løkker, eller i overgangen mellom blokk- og tekstbasert programmering

Fra figur 6.1 ser vi hvordan en kunne illustrert en «while-løkke» som i hvert steg multipliserer en oppdatert **a** med en «lagringsvariabel» **b**. Det er mange ulike teknikker som kan diskuteres i denne sammenhengen, som hvordan kunne dette vært gjort med blokk- eller tekstbasert programmering? Hvilken matematisk operasjon er det egentlig vi utfører gjennom løkken? Per nå er ikke flytskjemaer inkludert i undervisningsopplegget, men det er absolutt noe jeg ser for meg kan være en ekstra ressurs tilknyttet koblingen mellom blokk og tekstbasert, eller for å forklare logiske operasjoner, plassering av matematiske uttrykk, oppbygning av kode, med mer.

6.5 Teori (Θ)

En ringvirkning av hint, tips, tydelighet på hvilken teknikk som tas i bruk og andre elementer som hjelper med oppstart av oppgaver handler om legitimering av verktøyet programmering og nytteverdien elevene ser i arbeidet de driver med. Det er riktig nok en balansegang også her, hvorpå noen elever kanskje vil slite med å se nytteverdi, og sånn sett ikke kunne legitimere bruken av programmering dersom oppgavene blir for banale. Banale oppgaver vil i denne sammenheng handle om de som kan karakteriseres som type oppgaver, hvor programmering ikke er den kjente teknikken. Det er med andre ord viktig for meg å la alle jobbe med datasett og noen type problemer som typisk ville tatt tid med penn og papir eller geogebra, nettopp for å starte legitimeringsprosessen hos elevene og ikke bare spille på en slags autoritær aksept – som kan eksistere i et klasserom. På den andre siden har du elever som kanskje trenger de kjente oppgavene og teknikkene, for å kjenne på mestring og sånn sett motiveres til bruk av programmering i enda større grad enn at de selv legitimerer bruken gjennom nytteverdien av å kunne løse tidligere tungvinte problemer.

Nok en gang blir det her relevant å prate om overgangen mellom blokk- og tekstbasert programmering. Delen av teori som handler om å motivere bruk og videre studier av et emne på bakgrunn av legitimering og nytteverdi, kan også her forsterkes av å tydeliggjøre sammenhengene mellom de to teknikkene innenfor programmering. Med andre ord snakker jeg her om et noe større bilde, som avgjør hvorvidt elevene føler at de starter med eller uten legitime og nyttige forkunnskaper. Målet er utvilsomt at de skal oppleve likheter i struktur og syntaks, og kunne bygge videre på disse når de selv skriver kode som tidligere har vært presentert som blokker. Bilde 6.4 og noe tilsvarende flytskjema i figur 6.1 er som nevnt tidligere hjelpemidler på veien mot en tydeligere sammenheng, hvor håpet er at gjennom å legitimere det elevene har arbeidet med før og sette det i en sammenheng hvor de

forhåpentligvis ser nytteverdien av det, kan gjøre etablering av tekstbasert programmering som et nyttig verktøy for elevene mer nærliggende.

Vi har sett at problemer tilknyttet oppstart hovedsakelig har blitt håndtert med tips, en kode elevene kan fortsette på eller lignende, men innenfor teori blir også dette med å fjerne løsningsmetoder viktig å ta tak i. Fra bilde 6.6 og 6.7 har vi sett på videreutviklede oppgaver som også introduserer ekstraoppgaver hvor elevene får forsøke seg med en annen metode eller teknikk enn den «hovedoppgaven» fokuserer på. Det å jobbe teoretisk med flere løsningsmetoder, slik at en legitimerer og fremmer nytteverdien av å kunne inkludere disse teknikkene i videre arbeid anser jeg som viktig for å kunne utnytte verktøyet programmering fullt og helt. Hvorvidt dette er bruksområder tilknyttet de ulike matematiske operatorene, hvilken løkke du skal bruke eller bare helt generelt være bevisst på at oppgaver kan løses på flere måter, men at effektivitet og ikke minst mengden arbeid som går inn de ulike metodene varierer.

Bilde 6.12: Sammensatt eksempel

```
#SAMMENSATT EKSEMPEL
#Under skal vi se på et eksempel som bruker funksjon, "for" og "if" for å prøve og finne nullpunktene til en funksjon
#..av grad tre.
import numpy as np

def nullpunkt(a, b, c, d, start, stop):
    def f(a, b, c, d, x):
        return a*x**3 + b*x**2 + c*x + d
    for i in range(start, stop):
        if f(a, b, c, d, i)==0:
            print('Nullpunkt i x=', i)
    return

#nullpunkt(1, -2, 4, 7, -5, 5)
```

SPØRSMÅL: Koden over kan kun brukes til å finne heltallallige nullpunkter, kan du se hvorfor?

Bilde 6.12: Sammensatt eksempel som tar i bruk funksjoner, vilkår og løkker.

Fra bilde 6.12 ser vi et eksempel på en kode som fungerer for formålet å finne heltallige nullpunkter for tredjegradspolynomer. Eksemplet er i hovedsak der for å vise en måte en kan kombinere flere funksjoner, løkker og vilkår i samme kode, men den kan også kobles opp mot dette med effektivitet og mengden arbeid som må til – hvis en ser på nytteverdien elevene potensielt vil føle at å gjøre som i eksempelet over eller finjustere koden så den takler løsninger som ikke er heltall. Det er utvilsomt mulig å spille videre på det sammensatte eksempelet, og for eksempel introdusere en steglenge innenfor *range()* med avrunding eller feilmargin i betingelsen tilhørende «if-setningen». Grunnen til at jeg allikevel har valgt å beholde oppgaven som den er, handler om å treffe en hel gruppe elever, også er det i tillegg en

fin utfordring for de som ønsker og ta det neste steget. Jeg tror dog at for mange vil diskusjon tilknyttet hvorfor en bare finner heltallige nullpunkter være nok til at de ser nytteverdien av et slikt sammensatt eksempel, mens numerikk i enda større grad kommer inn i R1 og R2.

6.6 Didaktisk ingeniørvirksomhet

Den didaktiske ingeniørvirksomheten vil i dette tilfelle inneholde to faser, nemlig de to første – *forberedende analyse* og *a priori-analyse*. Førstnevnte har ikke gjennomgått alt for store endringer siden den opprinnelige utviklingen av undervisningsopplegget, da både epistemologisk og institusjonell analyse fortsatt er gjeldende for et videreutviklet opplegg. I dette legger jeg at målkunnskap og institusjonaliserte rammer er de samme selv om jeg har forsøkt å ta undervisningen et steg videre. Den didaktiske analysen må dog gjennomføres på nytt, og i delkapittel 4.1 pratet jeg om å forutse ulike didaktiske «under-situasjoner», slik at når jeg nå har brukt mine observasjoner og elevene sine tilbakemeldinger tilknyttet slike «under-situasjoner» må analysen naturlig nok oppdateres. Jeg nevnte hvis en eller flere elever står fast, trenger en utfordring eller ikke ser nytteverdien av programmering som verktøy, som eksempler på slik «under-situasjoner» som burde planlegges. Disse tre er fortsatt relevante, men har i ulik grad allerede blitt tatt tak i, gjennom videreutvikling av undervisningsopplegget. Hjelp med oppstart, samt utfordringer i form av ekstraoppgaver som bygger på andre løsningsmetoder og teknikker, har vært en sentral del av videreutviklingen basert på de tilbakemeldingene jeg fikk. Når det er sagt vil ikke det si at alle mulige slike «under-situasjoner» er fjernet helt fra opplegget, men sammenlignet med opprinnelig undervisningsopplegg vil disse nå sannsynligvis oppstå senere i løsningsforløpet, samt at utfordringene kommer i større grad som en naturlig og tydelig del av oppgavene. Når det gjelder elever som utfordrer nytteverdien av programmering som verktøy, og kanskje ikke ser hvorfor de skal bruke tid på å lære akkurat dette, kan for eksempel lister og sjekk av nullpunkt som i bilde 6.12 være potensielle innfallsvinkler. Grunnen til at jeg anser lister som viktig her, er mulighetene de gir for å generere og håndtere større datasett på en hurtig og ryddig måte. Det sammensatte eksempelet gir et inntrykk av hvor kjapt en kan sjekke et større datasett mot en funksjon, og fører også med seg muligheten for å gå videre til tilnærming av nullpunkt gjennom numerikk.

A priori-analyse, eller hvilke forventninger jeg nå har til undervisningsopplegget vil naturlig nok også ha endret seg sammenlignet med opprinnelig opplegg. En forventning tilknyttet strukturen til opplegget er at flyten i overgangen mellom gjennomgang, eksempler og oppgaver blir noe bedre gjennom tips, hint og allerede oppstartet kode. Samtidig forventer jeg

at det vil komme spørsmål til disse tilleggsdetaljene, men ideelt sett skal disse hjelpe elevene i gang med enten å skrive koden eller diskutere seg fram til en start med medelever. Noe jeg ikke fikk testet opprinnelig, men som jeg gjennom videreutviklingen har enda større forventninger til er bruken av «Jupyter Notebook». På grunn av skolen sitt valg av program hadde jeg ikke muligheten til å teste dette, men gjennom videreutvikling av tekstceller og enda større visuelle forskjeller mellom ulike typer tekst, kode og andre elementer i undervisningsopplegget forventer jeg at når strukturen og hvordan opplegget var lagt opp i utgangspunktet ble godt mottatt, vil «Jupyter Notebook» virke forsterkende med tanke på dette inntrykket. Når det er sagt blir det naivt å ikke nevne at kjøring av «Jupyter Notebook» gjennom «Anaconda Navigator» krever noe mer av elevene enn for eksempel «MU», «Thonny» eller nettbaserte «Trinket». Det er med andre ord essensielt å ta seg tid til å installere og hjelpe elevene med å bli vant med «Jupyter Notebook», men når det er gjort vil deling av undervisningsopplegg og ikke minst automatisk systematisert lagring store bonuser ved å jobbe i denne editoren.

Ettersom jeg nå er inne i andre runde med didaktisk ingeniørvirksomhet innenfor *a priori-analyse* blir det naturlig å snakke om hvorfor jeg forventer at denne videreutviklingen skal fungere bedre enn originalt undervisningsopplegg, og da ikke kun i klassen jeg testet opprinnelig opplegg, men på generelt grunnlag. I retrospekt kan jeg være enig med de tilbakemeldingene som kom tilknyttet de åpne oppgavene, og at oppstart av disse var utfordrende. Tanken var å la elevene få utforske programmering, tilhørende innebygde funksjoner, se at ulike løsningsmetoder fungerer og ikke minst skape en tilhørighet til det de produserer. Med et noe annerledes grunnlag, for eksempel at koblingen mellom blokk- og tekstbasert programmering var noe tydeligere, eller at overgangen allerede var startet på ungdomsskolen for flere kunne dette fungert bedre. Det jeg nå forventer er et undervisningsopplegg som treffer hele klassen på bedre måte, og at en som lærer kan være bevisst på de oppgavene med potensial for å ta neste steg og utfordre de som tar introduksjonen fort. Det er dog ikke sånn at ekstraoppgaver og de noe mer utfordrende delene av opplegget er kun for elever som tar andre deler veldig fort, men det er også muligheter for felles diskusjon og gjennomgang for å vise fram og legitimere bruksområder for programmering.

I det siste avsnittet tilknyttet didaktisk ingeniørvirksomhet skal jeg prate om tenkt realisering, med andre ord en tjuvstart på tredje fase. Basert på original gjennomføring vil en utvilsomt trenge tre til fire dobbeltimer for å kunne dekke hele undervisningsopplegget på en god måte.

Rekkefølgen vil bli nokså lik som originalt, og følge inndelingen celle for celle – se vedlegg 2. Det som er nytt er oppstarten, med et eksempel som forsøker og knytte blokkprogrammering til oppstarten av tekstbasert programmering. I tillegg vil det her være naturlig å kunne ta i bruk for eksempel et flytskjema for å illustrere struktur og oppbygning av blokkene de nå må lage selv. Videre er fokuset på å dekke flere løsningsmetoder gjennom ekstraoppgaver og diskusjon nytt, sammenlignet med åpenheten som var originalt. Plassen denne mer utforskende delen av opplegget får, vil jeg anta at en som lærer må tilpasse ut fra klasse og generell mottagelse av opplegget. Hvorvidt det blir diskusjon mellom medelever, felles i klassen, håndtert som en vanlig oppgave eller gjennomgang av lærer mener jeg må falle ned på læreren sin vurdering av hva som trengs. Det overordnede målet med å kunne beholde opplegget som et slags oppslagsverk til senere gjelder fortsatt, slik at når det kommer programmeringsoppgaver senere finner elevene igjen undervisningsopplegget når de åpner «Jupyter Notebook», med tilhørende eksempelkode, gjennomførte oppgaver, kommentarer og begrepsliste.

Kapittel 7

Diskusjon

Nest siste kapittel i denne studien er diskusjon, og vil foregå både ut fra analyse av datamateriale, undervisningsopplegg og det etablerte teoretiske rammeverket, men også med fokus på tidligere forskning, samt refleksjoner forankret i profesjonen jeg nå skal tre inn i. Kapittelet er bygd opp slik at jeg først ønsker å ta tak i et av de store målene i denne studien, nemlig hvordan relatere programmering til matematikk? Videre følger et delkapittel om bruk av undervisningsopplegget, og hvordan tidligere forskning og mine erfaringer fra realisering motiverer bruken av et slikt opplegg. I 7.3 skal jeg se diskutere begrensninger tilknyttet både undervisningsopplegget og studien som helhet, da en som forsker og ikke minst profesjonsutøver bør være bevisst på at en studie tilknyttet realisering i en klasse ikke nødvendigvis er representativt for mangfoldet en møter i skolen. Avslutningsvis ønsker jeg å ta tak i mulig videre forskning, som en naturlig overgang mot konklusjonen i denne studien.

7.1 Hvordan relatere programmering til matematikk?

Jeg har gjennom denne studien sett en generell aksept for oppgaver hvor programmering har ulik grad av følt nytteverdi og ikke minst plasseringen av programmering i matematikkfaget fra deltakerne. Det har også kommet fram at de ser nytten av programmering i matematisk sammenheng, så det store spørsmålet blir hvordan opprettholde og synliggjøre relasjonen mellom programmering og matematikk? Det å programmere matematiske operasjoner, og sette disse inn eksempelvis en løkke kan anses som en relasjon mellom de to. Stegene en tar for å komme dit kan ofte være forankret i oppgaveformulering, enten ved at en eksplisitt ber elevene løse oppgaven ved bruk av programmering eller at en bygger på konsepter som store tallmengder for å dytte elevene i retning av programmering. Felles for de to er at programmering blir et verktøy innen matematikk, og at relasjonen mellom de to kan for eksempel være at programmering gjør utførelsen av matematikk mindre tidkrevende eller lignende. Den relasjonen jeg prater om her er med andre ord mer lik den elevene har med graftegning og geogebra, de skal vite at programmering er et hjelpemiddel, men at det må beherskes og mestres for å få best mulige utbytte – noe som kan sammenlignes med slik deltakerne fra intervjuene omtalte økt forståelse av programmering.

Benton et al. (2017) peker på de fem «E'ene» som et rammeverk for utvikling av programmeringsopplegg med forankring i matematikk. Fra dette dukker det opp et nytt

spørsmål, nemlig hvordan disse passer med måten jeg har utviklet mitt opplegg på. Studien deres har som utgangspunkt elever i aldersgruppen 9-11 år, så et annet interessant spørsmål blir hvorvidt dette rammeverket er overførbart for en eldre aldersgruppe.

Tabell 7.1: De fem «E'ene»

Engelsk	**Norsk**	**Forklaring**
Explore	Utforske	Muligheter til å teste ut og prøve ideer selv, samt feilsøking.
Envisage	Visualisere/Forutse	Muligheten til å se for seg hva som kommer til å skje, og reflektere over hvorfor.
Explain	Forklare	Muligheten for diskusjon med medelever og hele klassen.
Exchange	Dele	Muligheten for å dele og bygge på andres ideer (både medelever og lærer).
bridgE	Koble sammen	Koblingen opp mot matematikken burde være en sterk ide, som utrykkes eksplisitt.

Tabell 7.1 inneholder oversatte tolkninger av de fem «E'ene» (Benton et al., 2017, s. 122-123).

Fra tabell 7.1 kan vi se at mye av rammeverket til Benton et al. (2017) er gjenkjennbare prinsipper også for utviklingen jeg har bedrevet. Starter en på toppen, hadde jeg originalt lagt enda mer i utforskning enn slik den forklares i tabellen, og til dels også i videreutviklingen ettersom fokuset fortsatt er å introdusere flere teknikker for samme oppgavetype, men på et noe mer lukket og ledende måte – i form av hjelp med oppstart, og ekstraoppgaver som dekker andre teknikker. Det å ha muligheten til å teste ut ideer og ikke minst feilsøk er dog felles, og en måte å jobbe på jeg mener passer like godt for yngre og eldre aldersgrupper.

Det å visualisere eller forutse hva som kommer til å skje, samt reflektere over hva som faktisk skjedde når du kjørte koden, er et annet fellestrekk gjennom spesielt diskusjonsoppgavene jeg inkluderte i undervisningsopplegget. Visualisere og forutse sammenfaller i større grad med å forklare i min utvikling enn den nevnt av Benton et al. (2017). Spesielt fokuset på helklasse

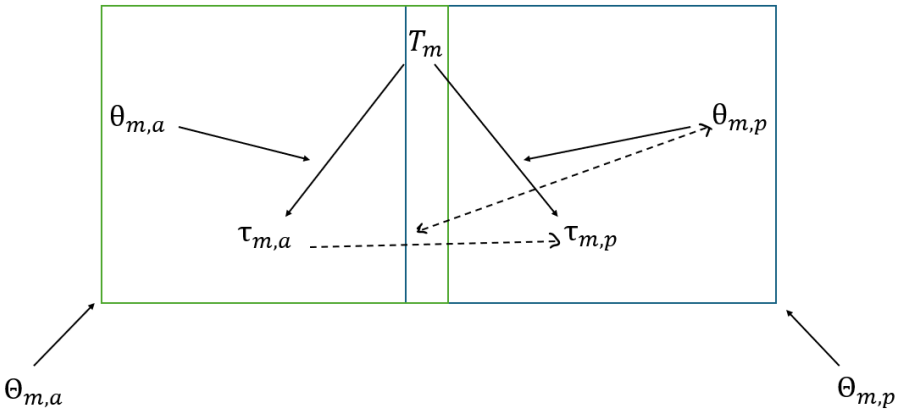
diskusjoner og forklaringer er noe jeg overlater å vurdere om trengs til læreren som gjennomfører opplegget, og her kommer kanskje det første tydelige skille mellom barneskole og videregående – altså forskjellene i alder mellom målgruppene i studiene. Et poeng her kan være at forskjellene på hvor selvstendig og raskt elever plukker opp teknikker innenfor programmering, og ikke minst måten de kommuniserer med medelever og lærer er forskjellig for de to aldersgruppene.

De to siste «E'ene» handler om deling og det å koble sammen matematikk og programmering. Deling slik det presenteres av Benton et al. (2017) ligner nokså mye på det jeg har omtalt som gjenbruk av kode, samt samarbeid mellom elever. Samarbeid og deling av ideer for løsningsmetoder faller inn under samme vurderingssak som dette med helklasse diskusjoner, men her blir det naturlig at valget tas av elevene istedenfor læreren. Igjen mener jeg handler det om forskjellene tilknyttet selvstendighet og kommunikasjon, og lærer sin rolle mener jeg blir i videregående skole og heller følge med på hvorvidt deling og refleksjoner skjer hos alle involverte parter – for å unngå at det er en elev som hele veien gjør alt av jobb. Når det gjelder å etablere koblingen mellom programmering og pensum innenfor matematikk som en sterk ide hos elevene, er dette noe jeg har forsøkt gjennom eksempler og matematisk innhold i oppgaver, for eksempel andregradsformelen. Det som skiller mitt syn på denne koblingen og Benton et al. (2017) er dette med å begrense seg med pensum. Det kan være naturlig for 9-11 åringer, men jeg mener at i tråd med den mer eksplisitte utforskende delen av kompetansemål og læreplan (Utdanningsdirektoratet, 2020) og det å stille spørsmål til verden (Chevallard, 2015), er også oppgaver med potensiale for å bygge videre på kjent matematikk fra 1T for å legitimere og se nytten av programmering i enda større grad viktig i videregående skole.

Fra studien til Stigberg og Stigberg (2020) kommer det fram spesielt to resultater fra intervjuprosessen i case-studien jeg ønsker å ta tak i her. Den første er tilknyttet intervju av lærere, som peker på problemløsningsstrategier og kommunikasjon som to overførbare teknikker som til en viss grad er mer framtvunget innenfor programmering enn matematikk, men som er vel så nyttig å anvende innen matematiske problemer. Den andre er elevene i studien sin oppfattelse av relasjonen mellom matematikk og programmering, og at matematikk er en forutsatt forkunnskap for å kunne programmere (Stigberg & Stigberg, 2020, s. 492). Førstnevnte er utvilsomt noe å være bevisst på som lærer, og ikke minst at disse teknikkene er noe en kan jobbe med også før oppstarten av programmering, for å forberede elevene på hvordan en kan angripe ukjente typer oppgaver. Hva angår elever sin oppfattelse

av relasjonen mellom programmering og matematikk, kan denne minne om koblingene mellom de ulike delene av en matematisk og programmeringsrelatert prakseologi.

Figur 7.1:



Figur 7.1 kjenner vi igjen fra kapittel 5, og her ser vi koblingene som springer ut av en matematiske type oppgave (T_m) til matematiske teknikker innenfor algebra ($\tau_{m,a}$) og programmering ($\tau_{m,p}$). Det interessante sammenlignet med elevene sin beskrivelse av relasjonen mellom programmering og matematikk er den stiplede linjen mellom de to teknikkene. Med andre ord mener elevene, slik jeg tolker det, at kunnskap om de nødvendige matematiske teknikkene er en forutsetning for programmering innenfor matematikk. Hvorvidt dette er tilfelle for de to prakseologiene T_m springer ut i, er jeg mer usikker på. Ettersom utgangspunktet er en matematisk oppgave, kan en utvilsomt si at noe matematisk kunnskap er en forutsetning for å klare og løse problemet. Når det er sagt vil for eksempel 1T elever være i stand til å regne på rekker og følger med bruk av programmering, uten å trenge den teoretiske bakgrunnen for summen av en rekke eller lignende. Så i dette eksemplet kan en faktisk snu på det, og si at programmering kan være en forutsetning eller innfallsvinkel i arbeidet med matematiske teknikker. Det er dog viktig å nevne at resultatene fra Stigberg og Stigberg (2020) er hentet fra deres interaksjoner med lærere og elever på 9. trinn, noe som kan forklare hvorfor matematikken omtales som en forutsetning, fordi du ikke enda har nådd litt mer komplekse ideer som det gir mening å visualisere gjennom programmering – eksempelvis rekker, følger, grenseverdi, derivasjon, med mer.

7.2 Hva motiverer bruk av undervisningsopplegget?

Tanken bak undervisningsopplegget, hvis en isolerer bruken til en klasse hvor jeg som lærer tar det i bruk, er rett og slett å ha et oversiktlig, gjennomtenkt og utfordrende opplegg som

skal kunne benyttes av hele klassen. Som nevnt tidligere er noe av begrunnelsen for strukturen, forklaringene, begrepsliste og oversikt over operatorer at elevene skal kunne ta i bruk undervisningsopplegget ved en senere anledning, for å repetere, øve mot prøve eller bare utforske programmering videre. I tillegg har jeg selv opplevd at elever på videregående skole ikke nødvendigvis har den beste strukturen på hvor de lagrer potensielt nyttige filer, en utfordring for lærere som ønsker å hjelpe, som i stor grad elimineres av hvordan «Jupyter Notebook» lagrer og sorter filer. Beveger en seg inn i en større setting, og vekk fra den ene klassen, vil et velfungerende undervisningsopplegg kunne være en ressurs i skolen – og kanskje spesielt for et tema som programmering, som både er nokså nytt innenfor læreplan, men også som en del av utdanningen til lærere. Stigberg og Stigberg (2020, s. 488-489) peker på utfordringer og intensive forkurs for lærere etter at programmering ble inkludert i fellesfagene, og selv om det nå har gått et par år, kan en tenke seg at ikke alle lærere er like komfortable med undervisning av programmering, slik at en ekstra ressurs inn i skolen ville blitt godt mottatt.

Overgangen fra blokk- til tekstbasert programmering, er også blitt en større del av undervisningsopplegget – spesielt gjennom videreutviklingen. Weintrop og Wilensky (2015) konkluderer med at elever syntes blokkprogrammering er enklere å anvende, noe som kanskje ikke er en stor overraskelse, men peker også på hvordan fargesammensetninger kan brukes til å bedre forståelsen av syntaks og generell oppbygning av programmer. I tillegg understreker de at på videregående nivå vil elever merke at blokkprogrammering har begrensninger tilknyttet beregningshastighet og når en skal lage større og mer komplekse programmer med flere operasjoner eller handlinger på en gang. Dette passet til en viss grad med mine observasjoner, da det ikke kom spørsmål om hvorfor vi ikke bruker blokkprogrammering, men heller at kompleksiteten av tekstbasert programmering gjorde det vanskelig å se hvor forkunnskapene skulle anvendes. Det er med andre ord et potensiale til stede for å utnytte elevene sine forkunnskaper tilknyttet programmering, men det kan virke som at det forutsetter at en som lærer bruker noe tid på å løfte fram disse sammenhengene, og hvilke egenskaper som faktisk er overførbare. Hvorvidt det holder med eksemplene jeg har inkludert, se bilde 6.4, eller om man må til med flytskjemaer, en kjapp repetisjon i «Scratch» eller lignende hadde vært interessant å teste ut, men for denne gang kan en som lærer heller ta det med seg, og være bevisst på hvilke muligheter som er til stede.

7.3 Begrensninger

Diskusjon av begrensninger har jeg valgt å dele opp, med en del tilknyttet selve undervisningsopplegget og en som omhandler forskningsmetode og validitet. Begrensninger kan kanskje høres ut som et negativt ladd ord i forbindelse med en studie, men fokuset i dette delkapittelet handler like mye om bevisstgjøring og realisme, og ikke nødvendigvis negative sider med opplegg eller studie. Jeg ønsker derfor å se på utfordringer som kan oppstå med opplegget, samt diskutere spesielt datainnsamling og utvalg opp mot hvor anvendbar studien er i en helt annen klasse enn hvor jeg realiserte undervisningsopplegget og baserte videreutviklingen på.

7.3.1 Undervisningsopplegg

Ønsket om å kunne bruke undervisningsopplegget som en ressurs for både elever og mulig kolleger, kommer med visse krav til ferdigheter. Spesielt er det ferdigheter tilknyttet «Jupyter Notebook» som må til for å kunne ta i bruk hele opplegget, og det til tross for at jeg brukte en annen editor under realiseringen. Videreutviklingen har gjort det vesentlig mer essensielt å benytte «Jupyter Notebook», ettersom det nå er inkludert både bilder og formel-grafikk i form av «LaTex». Når det er sagt vil jeg tro at dette ikke blir en stor hindring, og at hvis en ønsker å bruke et slik opplegg er ikke nedlastning av «Anaconda Navigator» og lignende noe stort problem.

Ser en mot de potensielt utforskende og til dels utfordrende delene av undervisningsopplegget, som for eksempel å skulle utforske flere løsningsmetoder for samme problem, eller det å kunne bygge videre på oppgaver og kode-eksempler for å stimulere ulikt engasjement og utforskertrang hos elever, er dette noe som ikke er utprøvd eller inkludert i undervisningsopplegget. Slik jeg ser det handler det da om erfaring, både som lærer og innenfor matematisk programmering. Det kan da oppstå ulike situasjoner, som at en nyutdannet lærer med en god del erfaring innen matematisk programmering, for eksempel ikke helt klarer å finne det neste steget for en elev, og at det enten blir for vanskelig og for likt som der den var. Tenker en seg et ytterpunkt på den andre siden, kan en meget erfaren lærer som ikke har programmert like mye, skjønne at denne eleven kan jo starte på halvveringsmetoden eller Newtons metode, men møte på problemer med å hjelpe og implementere disse.

Essensen av undervisningsopplegget sine begrensninger kan kanskje spores til usikkerhet, og jeg kjenner på et ønske om å teste ut videreutviklingen. Blir det relevant at andre skal bruke

undervisningsopplegget vil det sannsynligvis være naturlig å inkludere et slags løsningsforslag, samt tips til videre arbeid og potensielle utfordringer.

7.3.2 Forskningsmetode og validitet

Et av de store spørsmålene når det gjelder forskningsmetode og validitet er hvorvidt undervisningsopplegget passer inn i en hvilken som helst klasse. Sannsynligvis er det veldig få, om noen, undervisningsopplegg som passer perfekt i alle klasser – så spørsmålet profesjonsutøveren lærer kanskje heller må stille seg kan for eksempel være hva kan jeg bruke eller hvordan kan jeg bruke det? Realisering og intervjuer er kun tilknyttet en klasse, og til tross for at min oppfatning og til dels faglærer i klassen, var jeg at fikk samlet datamateriale fra elever med ulikt utgangspunkt er fortsatt mange ukjente. Det er spesielt tre punkter jeg ønsker å ta tak i, for å diskutere hvordan undervisningsopplegget kan bli oppfattet og mottatt i andre klasser, nemlig den ukjente læreren, graden av aksept og tilhørighet.

Den ukjente læreren, meg i dette tilfelle, kan utvilsomt gi utslag i både positiv og negativ retning. Jeg var såpass heldig at jeg hadde med faglærer i timen, men relasjoner og sammenligningsgrunnlag i ulike didaktiske situasjoner mellom lærer og elever blir naturlig nok svekket når læreren ikke kjenner klassen, og klassen ikke kjenner læreren. Et viktig poeng med denne typen undervisningsopplegg, både som ukjent og kjent lærer hos klassen, er at det er nokså omfattende og elevene får relativt mye servert når opplegget deles. Det er kanskje ikke alltid lett å se eller legge merke til, men spesielt gjennom intervjuene merket jeg at elevene verdsetter den innsatsen som er lagt ned.

Når det gjelder grad av aksept vil en nok utvilsomt møte klasser og enkeltelever som ikke har den samme aksepten for at det læreren eller oppgaven sier at skal gjøres, det gjør jeg. Det kan være snakk om å ikke se nytteverdien av programmering som verktøy, enten i akkurat denne oppgaven eller innenfor faget matematikk. Det er langt fra garantert at dette opplegget blir møtt med interesse og aksept, og for noen elever kan det nok være enklere å utfordre den kjente faglæreren på akkurat dette. Derfor mener jeg det er viktig å kunne vise fram nytteverdien av programmering, og legge opp til oppgaver som i det minste har potensial til å legitimeres gjennom endring av oppgaven sine parametere. I dette legger jeg for eksempel at hvis du blir utfordret på hvorfor en skal lære å lage funksjoner med bruk av programmering, når de kjenner til geogebra, så er det en fordel å kunne anvende lister eller sammensatte eksempler som illustrerer hvordan verktøyet programmering kan brukes på en annerledes måte en geogebra.

Tilhørighet til opplegget gjennom at elevene faktisk gjør oppgaver samme sted som «pensum», eksempler og generell teori presenteres, ble omtalt positivt i intervjuene. I analysen til Szabo et al. (2019) peker de på flere studier som tyder på at motivasjon og engasjement tilknyttet programmering påvirkes i stor grad av omgivelsene eller miljøet elevene arbeider i. Omgivelser eller miljø har her ikke den fysiske og hverdagslige betydningen, men omhandler strukturen, orden og utseende av kodefilen eller programmet elevene jobber innenfor. Disse omgivelsene, presisering av at elevene kan få bruk for opplegget senere og ikke minst tilhørigheten gjennom eget arbeid gjør at jeg tror undervisningsopplegget kan bli godt mottatt, gitt at en som lærer har en klar plan på hvordan håndtere mulige problemer, samt presentere og understreke både koblingen til matematikk og nytten en senere kan ha av å gjøre en god jobb nå.

7.4 Videre forskning

Avslutningsvis i kapittel 7 skal jeg ta tak i videre forskning, og diskutere hvilke muligheter som er for å ta videreutvikling og generell validitet, i form av å passe inn hos så mange som mulige, et steg videre. Noe jeg allerede har gitt uttrykk for er et ønske om å teste ut og realisere videreutviklingen, og dette er sannsynligvis det mest naturlige neste steget i arbeidet med et undervisningsopplegg som skal introdusere tekstbasert programmering forankret i matematikk. Ved test og vurdering av undervisningsopplegg kan det også være interessant og ta en enda mer kvantitativ tilnærming, gjennom for eksempel er pre- og posttest av elevene som har hatt og skal ha undervisningsopplegget. Eksempelvis kunne det vært interessant å teste et slik samlet undervisningsopplegg hvor elevene har tilgang til alle temaene samtidig, opp mot de eksterne programmeringsressursene som finnes fra lærebøker og som kurs på nettet.

Ser en mer isolert på et enkeltelement fra studien, er utvilsomt koblingen mellom blokk- og tekstbasert programmering noe som hadde vært interessant å studere enda nøyere. Etablering av en modul for denne overgangen, gjennom for eksempel fokus på hva de ulike fargene på blokkene betyr (Weintrop & Wilensky, 2015) eller arbeid med å oversette blokk til tekst, og omvendt. Det blir uansett veldig spennende å ta med seg denne erfaringen inn i profesjonshverdagen, og se hva jeg kan få ut av sammenhenger mellom de to, samt hvordan videreutviklet undervisningsopplegg blir mottatt.

Kapittel 8

Konklusjon

Som avsluttende kapittel i denne studien skal jeg eksplisitt se tilbake på innholdet, og hvordan dette svarer på forskerspørsmål og problemstilling.

8.1 Forskerspørsmål

Innledningsvisningsvis i kapittel 8 skal jeg ta tak i forskerspørsmål. Fra kapittel 1 kjenner vi igjen forskerspørsmålene presentert under.

> *(1) Hvordan har jeg utviklet, testet og samlet inn data tilknyttet undervisningsopplegget om programmering innenfor matematikk, og hvordan opplevde elevene dette undervisningsopplegget?*
> *(2) Hvordan kan en videreutvikle undervisningsopplegget basert på analyse av realisering og datainnsamling?*

Naturlig nok starter vi på toppen med første forskerspørsmål, der jeg gjennom didaktisk ingeniørvirksomhet (Strømskag, 2020a), realisering i en 1T klasse og fokusgruppe intervjuer i samme klasse har tatt tak i de tre delene av forskerspørsmålet. Det har vært veldig interessant å se hvordan Benton et al. (2017) sitt rammeverk gjennom de fem «E'ene» ble så tydelig gjenkjennbare også for utvikling og analyse gjennom didaktisk ingeniørvirksomhet. I delkapittel 7.1 har vi sett på likheter og ulikheter i utvikling og videreutvikling av et undervisningsopplegg innenfor matematisk programmering for de to rammeverkene, og naturlig nok ble det noen forskjeller på grunn av aldersgruppene i fokus – til tross for at mange fellestrekk.

ATD (Chevallard, 2019) kom enda tydeligere inn når jeg startet med forskerspørsmål nummer to, nemlig videreutvikling basert på analyse av realisering og datainnsamling. Det å kategorisere datamateriale gjennom etablering av i bunn og grunn to prakseologier, en elevsentrert og en med læreren i fokus. Her så vi blant annet hvordan bevisstgjøring rundt elevsentrerte og didaktiske teknikker (Artaud, 2019) kan være et nyttig verktøy for elever og lærere i en didaktisk situasjon, og samtidig hvordan en prakseologisk tilnærming kan hjelpe til med profesjonsrelatert utvikling. Profesjonen jeg nå skal inn i, og ikke minst hvor jeg skal benytte meg av studien jeg nå har gjennomført, legger også til grunn observasjoner og refleksjoner som har vært en viktig del av videreutviklingen av undervisningsopplegget. Elevene kan gi uttrykk for hvilke komponenter i undervisningsopplegget de satt pris på eller

savnet, men det ble min jobb å tolke og reflektere rundt hvordan de savnede skulle løftes fram, samtidig som en beholdt det verdsatte.

8.2 Problemstilling

Som med forskerspørsmål kjenner vi også igjen problemstillingen fra kapittel 1. Det er naturlig nok flere måter å introdusere tekstbasert programmering på, men forankringen i matematikk er for meg det essensielle med problemstillingen.

Hvordan introdusere tekstbasert programmering i 1T forankret i matematikk?

Jeg har flere ganger underveis i denne studien spurt meg selv, hvordan kan overgangen mellom blokk- og tekstbasert programmering forankres i matematikk? Konklusjonen på dette spørsmålet er for meg at målet er å kunne ta i bruk forkunnskapen, for å enklere og mer effektivt ta steget inn i matematisk programmering. Videre har vi sett at noen grunnleggende ferdigheter tilknyttet for eksempel betingelser og vilkår, kan fint bygges opp rundt matematikk, men hvorvidt det er en matematikk elevene er relevant for faget matematikk 1T er et annet spørsmål. Igjen handler det, slik jeg ser det, om å legge et grunnlag for hva som kommer. Når det er sagt kommer anvendelse av samtlige temaer med matematisk innhold, og avveiningen om å la blokk til tekst få en større rolle, samt beholde det som ble omtalt som «enkel matematikk» handler om et ønske om å få med så mange elever som mulig på logikken, oppbygning og ikke minst hvilke forkunnskaper de sitter på.

Mulighetene for å introdusere programmeringskonsepter gjennom matematikk er utvilsomt til stede, og all den tid matematikk har fått mye av ansvaret for den generelle introduksjonen før potensielle valgfag i 2. og 3. klasse på videregående – mener den fremtidige lektoren innenfor matematikk at som et verktøy innenfor mattefaget har en mulighet til å arbeide med pensum i faget, samt utforskning av monumentene pensum er transponert fra.

Referanser

Artaud, M. (2019). Praxeologies to be taught and praxeologies for teaching: 33A delicate frontier. I *Working with the Anthropological Theory of the Didactic in Mathematics Education* (s. 33-40). Routledge.

Benton, L., Hoyles, C., Kalas, I. & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, *3*, 115-138.

Bosch Casabo, M. (2018). Study and Resesrch Paths: A Model For Inquiry. I (s. 4033-4054). Proceedings to the International Congress of Mathematicians: Rio de Janiero. https://doi.org/10.1142/9789813272880_0210

Bosch, M. & Gascón, J. (2014). Introduction to the Anthropological Theory of the Didactic (ATD) IA. Bikner-Ahsbahs & S. Prediger (Red.), (s. 67-83). Springer. https://doi.org/10.1007/978-3-319-05389-9_5

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A. & Barone, D. (2017). Development of computational thinking skills through unplugged activities in primary school. Proceedings of the 12th workshop on primary and secondary computing education,

Chevallard, Y. (2006). *Steps towards a new epistemology*. IV Congress of the European Society for Research in Mathematics Education, http://yves.chevallard.free.fr/spip/spip/IMG/pdf/Steps_towards_a_New_Epistemology.pdf

Chevallard, Y. (2015). Teaching Mathematics in Tomorrow's Society: A Case for an Oncoming Counter Paradigm. I S. J. Cho (Red.), *The Proceedings of the 12th International Congress on Mathematical Education* (s. 173-187). Springer. https://doi.org/10.1007/978-3-319-12688-3_13

Chevallard, Y. (2019). Introducing the Anthropological Theory of the Didactic: An Attempt at a Principled Approach. *HIROSHIMA JOURNAL OF MATHEMATICS EDUCATION 12*, 71-114. https://www.jasme.jp/hjme/download/05_Yves%20Chevallard.pdf

Chevallard, Y. & Bosch, M. (2020a). Anthropological Theory of the Didactic (ATD). I S. Lerman (Red.), *Encyclopedia of Mathematics Education* (s. 53-61). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_100034

Chevallard, Y. & Bosch, M. (2020b). Didactic Transposition in Mathematics Education. I S. Lerman (Red.), *Encyclopedia of Mathematics Education* (s. 214-218). Springer International Publishing. https://doi.org/10.1007/978-3-030-15789-0_48

Danielsen, M. F. (2023). *Introduksjon av tekstbasert programmering* [Upublisert studie i forbindelse med RFEL3100]. NTNU.

FN. (2023). *The Sustainable Development Goals Report*. https://unstats.un.org/sdgs/report/2023/The-Sustainable-Development-Goals-Report-2023.pdf

Gueudet, G., Doukhan, C. & Quéré, P.-V. (2022). *Teachinig mathematics to non-specialists: A praxeological approach*. INDRUM2022: Fourth conference of the International Network for Didactic Research in University Mathematics, Leibniz University Hannover and INDRUM.

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

Kunnskapsdepartementet. (2017). *Overordnet del - verdier og prinsipper for grunnopplæring*. Kunnskapsdepartementet. https://www.regjeringen.no/contentassets/53d21ea2bc3a4202b86b83cfe82da93e/overordnet-del---verdier-og-prinsipper-for-grunnopplaringen.pdf

Mangiante-Orsola, C., Perrin-Glorian, M.-J. & Strømskag, H. (2018). Theory of didactical situations as a tool to understand and develop mathematics teaching practices. *Annales de Didactique et de Sciences Cognitives. Revue internationale de didactique des mathématiques*, (Special issue), 145-174.

NESH. (2021). *Forskningsetiske retningslinjer for samfunnsvitenskap og humaniora* (Bd. 5). https://www.forskningsetikk.no/retningslinjer/hum-sam/forskningsetiske-retningslinjer-for-samfunnsvitenskap-og-humaniora/

Robinson, N. (1999). The use of focus group methodology—with selected examples from sexual health research. *Journal of advanced nursing*, *29*(4), 905-913.

Robson, C. & McCartan, K. (2016). *Real World Research* (Bd. 4). John Wiley & sons Ltd.

Sevik, K. (2016). *Programmering i skolen*. Senter for IKT i utdanning. https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Stigberg, H. & Stigberg, S. (2020). Teaching programming and mathematics in practice: A case study from a Swedish primary school. *Policy futures in education*, *18*(4), 483-496.

Strømskag, H. (2020a). Didaktisk ingeniørvirksomhet i matematikk. *Samtaleorientert matematikk-et samspill mellom didakatiske og adidaktiske situasjoner*, 81-118. https://www.researchgate.net/profile/Heidi-Stromskag/publication/346488997_Didaktisk_ingeniorvirksomhet_i_matematikk_Multiplikasjon_som_modell_for_situasjoner_pa_3_trinn/links/62f6b0c8b8dc8b4403da82ac/Didaktisk-ingeniorvirksomhet-i-matematikk-Multiplikasjon-som-modell-for-situasjoner-pa-3-trinn.pdf

Strømskag, H. (2020b). Teorien for didaktiske situasjoner i matematikk. *Samtaleorientert matematikk-et samspill mellom didakatiske og adidaktiske situasjoner*, 25-80.

Szabo, C., Sheard, J., Simon, A. L.-R., Becker, B. A. & Ott, L. (2019). *Fifteen Years of Introductory Programming in Schools: A Global Overview of K-12 Initiatives*. Proceedings of the 19th Koli Calling International Conference on Computing Education Research, Koli, Finland. https://doi.org/10.1145/3364510.3364513

Utdanningsdirektoratet. (2020). *Kompetansemål og vurdering (MAT09-01)*. Utdanningsdirektoratet. https://www.udir.no/lk20/mat09-01/kompetansemaal-og-vurdering/kv42?lang=nob

Utdanningsdirektoratet. (2024a). *Eksamensveiledning matematikk 10. trinn*. https://sokeresultat.udir.no/eksamensoppgaver.html?start=1&query=matematikk%2010.trinn&ExCatalogTypeName=Eksamensveiledninger

Utdanningsdirektoratet. (2024b). *Eksamensveiledning matematikk fellesfag (MAT1019, MAT1021, MAT1023, MAT1151)*. https://sokeresultat.udir.no/eksamensoppgaver.html?start=1&query=matematikk%201t&ExCatalogTypeName=Eksamensveiledninger

Utdanningsdirektoratet. (2024c). *Eksamensveiledning matematikk programfag (REA3056, REA3058, REA3060, REA3062)*. https://sokeresultat.udir.no/eksamensoppgaver.html?start=1&query=matematikk%20r1&ExCatalogTypeName=Eksamensveiledninger

Weintrop, D. & Wilensky, U. (2015). To block or not to block, that is the question: students' perceptions of blocks-based programming. Proceedings of the 14th international conference on interaction design and children,

Wæge, K. & Nosrati, M. (2021). Oppgaver som fremmer resonnering og problemløsning. I *Motivasjon i matematikk* (Bd. 3, s. 79-90). Universitetsforlaget.

Vedlegg 1 – Intervjuguide

Formål med intervjuet:

Kartlegge deltagerne sin erfaring med introduksjon av tekstbasert programmering i matematikkfaget etter gjennomføring av undervisningsopplegg. Intervjuformen vil være semi-strukturert. Intervju og tilhørende spørsmål forutsetter deltagere som har tatt del i undervisningsopplegg som forsøker å introdusere tekstbasert programmering, gjennom eksempler og anvendelser knyttet til matematikk.

Problemstilling:

«Hvordan introdusere tekstbasert programmering i 1T forankret i matematikk?»

Introduksjon:

I denne fasen av intervjuet vil deltagerne få informasjon om formålet, samt en gjennomgang av det mest essensielle fra samtykkeskjemaet. Det vil også bli mulig for deltagerne å stille spørsmål om intervjuet, databehandling, osv.

Spørsmål:

- Har dere hatt programmering før 1T?
- Hvordan syntes dere undervisningsopplegget fungerte?
- Følte dere at det var relatert til matematikkfaget?
- Synes dere man burde ha matematikk i fokus når man driver med programmering i 1T?
- Er det noe med tekstbasert programmering dere tror passer spesielt bra/dårlig sammen med matematikk?
- Er det noe dere skulle ønske ble gjort annerledes?
- Hva er viktigst for dere når dere skal lære noe nytt?
- Hvordan opplever dere oppgaver om matematisk programmering som kunne vært gjort i Geogebra eller for hånd?

Vedlegg 2 – Undervisningsopplegg som Jupyter Notebook

I læreplanen til matematikk 1T står det:

"Formulere og løse problemer ved hjelp av algoritmisk tenkning, ulike problemløsingsstrategier, digitale verktøy og programmering"

Her skal dere få en gjennomgang av de grunnleggende komponentene i Python-programmering. Denne filen er tenkt som et oppslagsverk som du forhåpentligvis får bruk for senere, så lagre den et lurt sted :-)

Et tips når du tester ut kode er å bruke # foran f.eks. print() for å gjøre det om til tekst, dersom du ikke vil printe det akkurat nå. Det samme kan du gjøre for deler av koden du ikke har lyst til å kjøre akkurat nå.

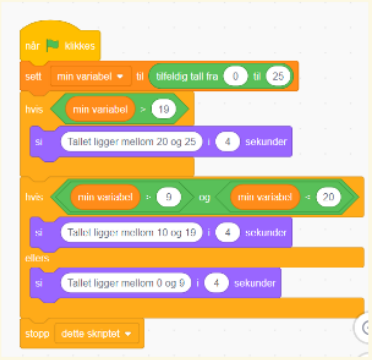
Hvis du skulle møte problemer i oppstarten av en oppgave, prøv og skrive ned eller tenke gjennom hva du vil at koden skal gjøre, og hva som kan hjelpe deg med å løse problemet. F.eks. hvis du skal sjekke om noe er oppfylt --> If-setning. Skal du gjenta noe mange ganger --> For- eller while-løkke. Skal du gjenta noe flere ganger med ulike tall --> Funksjon, osv...

Algoritmisk tenkning som det står om i kompetansemålet har mange forklaringer, men en av de mest brukte er å bryte ned et stort problem i flere små og "enklere" problemer - denne måten å arbeide på er absolutt anbefalt når du programmerer

Fra blokkprogrammering til python

Overgangen fra blokkprogrammering som "scratch" eller "microbit" kan føles stor. Det du må huske på er at selv om det ser annerledes ut, kan du fortsatt bygge opp koden på samme måte - du må bare lage blokkene selv. Kanskje enklere sagt enn gjort, men her kommer et par eksempler.

If-setningen:



Løkker:

De to eksemplene over kan skrives i python på følgende måte:

```python
#If-setningen:
import random as rnm

min_variabel=rnm.randint(0,25)

if min_variabel>19:
    print("Tallet ligger mellom 20 og 25")
elif min_variabel>9 and min_variabel<20:
    print("Tallet ligger mellom 10 og 19")
else:
    print("Tallet ligger mellom 0 og 9")

#Løkker:

a=0
i=0
while i<10:
    a=a+1
    i=i+1
    print(i)

print('a=',a)
```

Her ser vi at strukturen er nokså lik, men at du i python må bygge blokkene slik at de gjør det du ønsker. I tillegg kommer python med en god del bibliotek eller pakker, som f.eks "random", disse må importeres for å kunne bruke innholdet i dem. Vi skal bli bedre kjent med flere pakker senere, som blant annet numpy. For løkke-eksemplene, eller gjenta som det heter i scratch, bruker vi her while-løkke og istedenfor å si gjenta 10 ganger - introduserer vi en "telle-variabel" som vi kaller "i", og så lenge denne er mindre enn 10 gjentas handlingen inne i løkken.

Noen nyttige operatorer:

```
+ - multiplikasjon
- - substraksjon
* - gange
/ - dele
** - potens
!= - ulik
< - mindre enn
> - større enn
<= - mindre enn eller lik
```

Ekstra:

```
% - modulo (rest ved deling)
// - heltallsdivisjon
```

```python
#BETINGELSER

a=4 #definerer variabel a
b=2 #definerer variabel b

#bool() er en innebygd funksjon som returnerer true/false ettersom betingelsen stemmer eller ikke
bool(a>b)
```

$bool(a < b)$ ville returnert false.

$bool(a == b)$ ville returnert false, legg merke til at vi må bruke == istedenfor =. Dette er for å skille mellom betingelser og definering/endring av variabler.

$a! = b$ er betingelsen ulik, mens $a <= b$ betyr at a skal være mindre enn eller lik b.

```python
#OPPGAVE
#Lag et program som sjekker fire selvvalgte regnestykker ved å returnere true/false, bruk plus, minus, gange og dele
```

Vilkår

Vilkår kunne her like gjerne vært "if-setningen", og tar ut bruk en eller flere betingelser for å fortelle deg om en betingelse er oppfylt eller ikke - f.eks. om et tall (a) er større enn null, a>0.

Denne "if-setningen" kan bygges opp på flere måter, f.eks.

```
if a>6:
    ..gjør dette
if a<6 and a>3:
    ..gjør dette
elif a<3 and a!=0:
    ..gjør dette
else:
    ..gjør dette
```

Her ser vi at "if" kan brukes flere ganger, og hvis vi vil ha flere betingelser bruker vi "and". "else" brukes når du vil at alt som ikke havner innenfor de andre betingelsene skal gjøre noe spesielt.

Diskuter hvilket tall som sendes til "else" i eksempelet over.

```
#VILKÅR
import random as rnm #Her importeres pakken random, og gis forkortelsen rnm ved bruk av as.
#Grunnen til at vi ofte gir pakker forkortelser er for å effektivisere bruken

a=22
b=rnm.randint(0, 50) #variabel b defineres som et tilfeldig tall mellom 0 og 50, dette tallet vil endre seg hver
                     #..gang du kjører koden

if a<b: #hvis a er mindre enn b, gjør dette
    print('a er mindre enn b') #print er en innebygd funksjon som brukes til å returnere tekst, tall, etc.
elif a>b: #eller hvis a er større enn b, gjør dette
    print('a er større enn b')
else: #ellers, altså hvis ingen av de andre vilkårene stemmer, gjør dette. Hvilke tilfeller er dette?
    print('a er lik b')
```

```
import random as rnd
#OPPGAVE VILKÅR
#Lag en kode som ved bruk av if-setningen bestemmer om et tall er et partall eller oddetall.

tall=rnd.randint(0,100)
if tall%2==0:
"Fortsett på denne koden..."
```

EKSTRA: Får du til å gjøre oppgaven over med helltallsdivisjon (//)? Isåfall, hvordan skal du sjekke om heltallet du får er en avrundning eller ikke?

```
#Ekstraoppgaven kan gjøres her:
```

Lister

Lister er en måte å kunne sortere og/eller arbeide med flere tall samtidig i python. Disse kan lages manuelt med klammeparenteser [...], eller ved bruk av innebygde funksjoner som numpy sin arange eller linspace.

```
#LISTER
import numpy as np

liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] #Liste er navnet på listen
#print(liste)
liste_4 = liste[4] #liste[4] er det elementet som står på den 5 plassen i lista, fordi første elementet er liste[0]
#print(liste_4)

#Det finnes mange innebygde funksjoner som kan lage lister for deg
liste2=np.linspace(0,20,11) #Linspace er en innebygd funksjon fra pakken numpy, som tar inn (start, slutt, antall elementer)
liste2_1=np.linspace(0,20,10)
print(liste2)
print(liste2_1)

liste3=np.arange(0,10,1)
print(liste3)
```

DISKUSJONSSPØRSMÅL:

Diskuter med personen ved siden av deg:

Hvorfor velger vi 11 elementer og ikke 10 f.eks?

Hva er forskjellen på linspace og arange?

```
#OPPGAVE LISTE
#Lag en liste mellom 0 og 200 som øker med 2 for hvert steg, kan du gjøre det med linspace og arange?
```

Funksjoner

Funksjoner innenfor programmering kan bety litt forskjellig. Vi kan både ha de klassiske funksjonene du kjenner fra matematikken, men også funksjoner som utfører programerte handlinger med ønsket input.

Nedenfor skal vi se på både en klassisk "matte-funksjon", i tillegg til en som gjør noe litt ekstra.

Den første funksjonen vi skal se på er:

$$f(x) = x^2 - 2x + 1$$

I tillegg skal vi ta i bruk andregradsformelen:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
#FUNKSJONER
import numpy as np

def f(x): #funksjoner i python starter alltid med def
    return x**2-2*x+1 #return sier hva funksjonen skal returnere

#print(f(2))

#Under ser vi et annet eksempel på hvordan en funksjon kan se ut ved bruk av vilkår og betingelser
def andregrad(a,b,c):
    rot = b**2 - 4*a*c #Rotutrykket i ABC-formelen regnes ut
    if rot<0: #Hvis rotutrykket er mindre enn null er det ingen nullpunkter
        return print('Funksjonen har ingen nullpunkter')
    elif rot==0: #Hvis rotutrykket er lik null er det ett nullpunkt
        x_1 = (-b)/2*a
        return print('Funksjonen har ett nullpunkt, x=', x_1)
    else: #Ellers, altså hvis rotutrykket er større enn null, har vi to nullpunkter
        x_1 = (-b+np.sqrt(rot))/2*a
        x_2 = (-b-np.sqrt(rot))/2*a
        return print('Funksjonen har to nullpunkter, x=', x_1, '& x=', x_2)

#andregrad(1, -4, 2)
```

Diskusjonsoppgave

Hvorfor regner vi "rot" uten å ta kvadratroten? Kan dette forklares matematisk?

Dette kan gjøres for hånd og i geogebra også, men når kan det være nyttig å ha slik funksjon?

Fungerer funksjonen for lineære funksjoner?

```
#OPPGAVE FUNKSJONER
#Skriv om andregrad slik at den i tillegg til å returnere nullpunkter også returnerer en funksjon du kan bruke senere.
#TIPS: Det går an å ha en funksjon inne i en funksjon

def andregrad(a,b,c):
    rot = b**2 - 4*a*c #Rotutrykket i ABC-formelen regnes ut
    if rot<0: #Hvis rotutrykket er mindre enn null er det ingen nullpunkter
        return print('Funksjonen har ingen nullpunkter')
    elif rot==0: #Hvis rotutrykket er lik null er det ett nullpunkter
        x_1 = (-b+np.sqrt(rot))/2*a
        return print('Funksjonen har ett nullpunkt, x=', x_1)
    else: #Ellers, altså hvis rotutrykket er større enn null, har vi to nullpunkter
        x_1 = (-b+np.sqrt(rot))/2*a
        x_2 = (-b-np.sqrt(rot))/2*a
        return print('Funksjonen har to nullpunkter, x=', x_1, '& x=', x_2)
```

Løkker

Løkker brukes i programmering for å lage en gjentagende prosess. Der vilkår fungerer sånn at hvis dette gjør det, hvis noe annet gjør det og ellers gjør sånn, fungerer løkker slik at så lenge $a < 100$ gjenta dette eller så lenge $a \epsilon [0, 100]$ gjenta dette.

Det er i hovedsak to typer løkker vi bruker, **while** og **for**, hovedforskjellen her at den første tar i bruk en betingelse - dvs. så lenge $a < 100$, mens den andre bruker et intervall.

```python
#LØKKER
import numpy as np

a=7
i=0
b=53

#a+=1 betyr det samme som a=a+1, og ny a vil oppdateres for hver runde i løkka
while a<b:#while er en løkke som gjentas helt til en betingelse er møtt
    a+=1 #for hver runde i løkka legges det til en i a
    #print(a)
    i+=1 #"i" kan i dette tilfellet f.eks være en måte å telle antall runder løkka kjører


A=7
B=53
k=0
for j in range(A,B): #For er en annen type løkke hvor variabel j i et intervall (in range(start, stop, steg)) fra A til B
    k+=j #k+=j vil si at alle tall mellom A og B summeres sammen
    #print(k)
```

Hvorfor må jeg ha egne store A og store B? Hva hadde skjedd hvis jeg kun hadde skrevet $k = 0$ over for-løkken, og brukt $range(a, b)$? Diskuter med eleven ved siden av før dere tester ut.

```python
#OPPGAVE: Lag en kode som legger sammen de 15 første partallene og oddetallene i hver sin variabel.
#TIPS: Finn den øvre grensen, altså hvilket tall du må opp til for å ha akkurat 15 odde- og partall - velg så løkke-type.
```

Hvordan kunne oppgaven over vært løst med den andre løkke-typen (while eller for)?

```python
#SAMMENSATT EKSEMPEL
#Under skal vi se på et eksempel som bruker funksjon, "for" og "if" for å prøve og finne nullpunktene til en funksjon
#..av grad tre.
import numpy as np

def nullpunkt(a, b, c, d, start, stop):
    def f(a, b, c, d, x):
        return a*x**3 + b*x**2 + c*x + d
    for i in range(start, stop):
        if f(a, b, c, d, i)==0:
            print('Nullpunkt i x=', i)
    return

#nullpunkt(1, -2, 4, 7, -5, 5)
```

SPØRSMÅL: Koden over kan kun brukes til å finne heltallallige nullpunkter, kan du se hvorfor?

OPPGAVER

Det binære tallsystemet består av 0 og 1. Hvis vi vil skrive om f.eks. 10010 til vanlige tall må vi vite at fra høyre mot venstre dobles verdien for tall 1, tall 2, tall 3, osv, altså 1, 2, 4, 8, 16... Et 1-tall betyr at tilhørende verdi skal være med.

Eksmepel $10010 = 2^4 + 0 + 0 + 2^1 + 0 = 16 + 0 + 0 + 2 + 0 = 18$ eller $11111 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 16 + 8 + 4 + 2 + 1 = 31$

Du kan finne hint for oppgave 1 til 3 nederst (under begrepsliste).

```
import numpy as np
#OPPGAVER
# (1) Lag en funksjon som regner ut verdien til en binær sekvens med seks plasser.
def binary(x0, x1, x2, x3, x4, x5):
    a=0
    b=0
    c=0
    d=0
    e=0
    f=0

    bin_val=a**5+b**4+c**3+d**2+e+f
    return bin_val
```

```
#(2) Forklar hva koden under gjør:

def bin(liste):
    tall=0
    antall=len(liste)
    potens=antall-1
    for i in range(0, antall):
        tall+=liste[i]*2**(potens-i)
    return tall
```

SKRIV SVAR HER:

input er en innebygd funksjon som lar deg stille et spørsmål, og skrive inn et svar når du kjører koden. I oppgaven under bruker jeg input for å få den som kjører programmet til å velge tallet som skal tas fakultet av. int() delen før input forteller input at den skal forvente å få inn ett heltall (integer). Tallet som skrives inn lagres med navnet "fakultet".

```
# (3) Lag en kode som bruker while-løkke til å regne ut fakulteten til et tall. Eks: 5! = 5*4*3*2*1

fakultet=int(input('Hvilket tall skal tas fakultet av?'))
```

Plotting

Å lage grafer er også fullt mulig i python. Under skal vi se på hvordan du kan bruke matplotlib.pyplot biblioteket til å lage slike. Vi skal se på den samme funksjonen som vi brukte tidligere i opplegget, men denne kan dere selvfølgelig endre på. I tillegg tar vi i bruk lister, for å gi funksjonen vår flere verdier og samtidig lagre disse som en liste.

Eksempelet under inneholder ganske mye, men det eneste du egentlig trenger er:

```
plt.plot(x_verdier, y_verdier)
plt.show()
```

Alt annet er ekstra, og hvis du ønsker to plott eller fler hver for seg, kan du bruke plt.show() mellom hver plt.plot() for å skille dem. NB: Det er viktig å huske på at du må ha like mange tall i x_verdier og y_verdier, dvs. at hvis du prøver deg med 20 x_verdier og bare 15 y_verdier vil ikke python like det.

```
#PLOTTING
import matplotlib.pyplot as plt #matplotlib.pyplot er en pakke med grafiske verktøy
import numpy as np

def f(x):
    return x**2-2*x+1

def g(x):
    return -6*x+80

#Under ser vi på hvordan vi kan plotte en funksjon
x_verdier = np.linspace(0,10,11) #x-verdier for plottet vårt som en liste
f_verdier = f(x_verdier) #f-verdier for plottet finnes ved å sette listen med x-verdier inn i funksjonen
#f_verider blir nå en ny liste hvor alle verdiene fra x_verdier har blitt satt inn i f(x) - f(0), f(1),...,f(10)

g_verdier= g(x_verdier) #vi gjør det samme for g

plt.plot(x_verdier, f_verdier, 'r', marker='o') #plt. sier i fra hvilken pakke vi bruker, plot er en innebygd funksjon
plt.plot(x_verdier, g_verdier, marker=',')
plt.grid()
plt.title("Plott")
plt.xlabel("x-verdier")
plt.ylabel("y-verdier")
plt.show() #gjør at plottet vises, spesielt viktig hvis vi skal plotte flere funksjoner enten sammen eller hver for seg
```

BEGREPSLISTE

Pakke: Inneholder ekstra egenskaper, uttrykk, innebygde funksjoner, o.l. som kan benyttes i programmet ditt, f.eks. math og numpy.

Importere: Når vi henter inn en pakke, et datasett, e.l. i starten av et program.

Returnere: Hva programmet eller funksjonen sender ut.

Variabel: Et definert navn med en tilhørende verdi, f.eks. a=0

Lokal variabel: En variabel tilknyttet en del av koden, f.eks innenfor en funksjon

Global variabel: En frittstående variabel tilknyttet hele koden

String: Tekst

Integer: Heltall

Float: Desimaltall

Betingelse: I programmering brukes betingelse om en form for grense, veldig ofte et stoppested. F.eks. så lenge a<0 gjenta operasjonen over.

Vilkår: Hva skjer videre hvis en betingelse i koden er oppfylt, f.eks. hvis a>0 gjør det første, hvis a<0 gjør det andre og hvis a=0 gjør det tredje. Eks. If-setning.

Lister: En måte å utrykke flere tall, bokstaver, ord, osv. på en kompakt måte, hvor du kan sortere og hente ut enkeltelementer.

Funksjoner: Et større begrep enn den tradisjonelle oppfatningen i matematikk. Kan være alt fra f(x)=x+3 til mer komplekse funksjoner som tar inn en liste og sorter den fra størst til minst e.l.

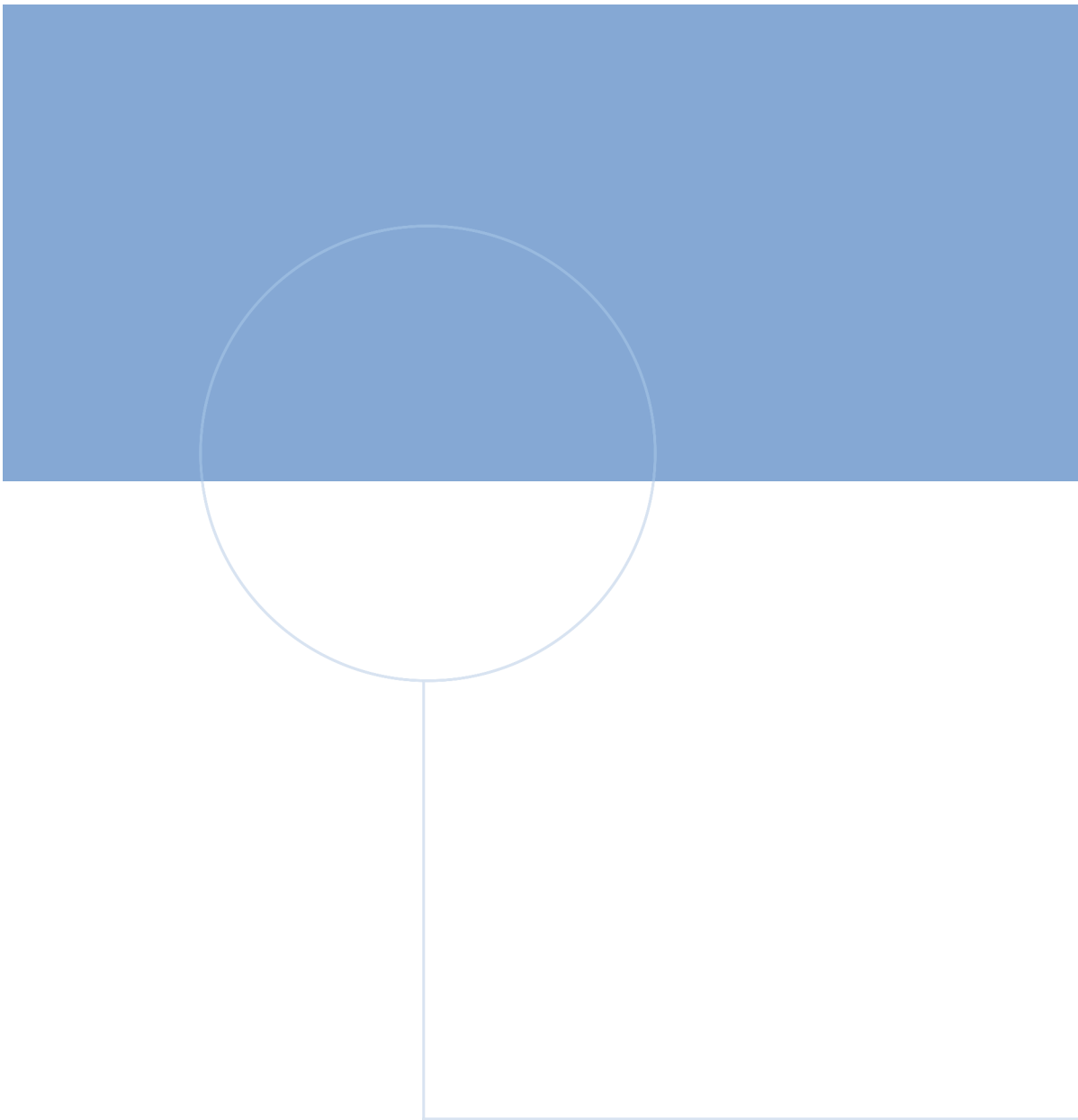
Løkker: En eller flere operasjoner gjentas. Operasjonen(e) vil gjentas et visst antall ganger og/eller til en betingelse er møtt.

Hint, oppgave 1-3

Oppgave 1: Bruk if til å sjekke om x0, x1,... er 0 eller 1, og hvis den er 1 oppdater tilhørende a,b,c,... til 2.

Oppgave 2: len(liste) betyr lengde av listen, altså len([0,1,1])=3. For-løkken ser på hver plass i lista, og gjør noe med tallet som står der.

Oppgave 3: Start koden med tall=1 og while fakultet>0, før du lager en løkke som for hver runde trekker fra 1 på fakultet og ganger med tall.



NTNU

Kunnskap for en bedre verden