Relevansen av kompetansen fra matematikk R2 i beregningsorientert biologi

En kvalitativ studie av biologistudenters bruk av problemløsningsstrategier

Sofie Rudberg

Master i realfagsutdanning

Institutt for biovitenskap
Det matematisk-naturvitenskapelige fakultet

30 studiepoeng

UNIVERSITETET I OSLO

1. juni 2020

«Det går faktisk an å gjøre det lett […]. Der har [Mendel] *sittet og paret og paret og paret alle de der erteplantene, og så skriver du det på fire linjer eller hva det var, og får samme svar. Det er fascinerende.»*

Relevansen av kompetansen fra matematikk R2 i beregningsorientert biologi

En kvalitativ studie av biologistudenters bruk av problemløsningsstrategier

2020

Relevansen av kompetansen fra matematikk R2 i beregningsorientert biologi

https://www.duo.uio.no/

Trykk: Reprosentralen, Universitetet i Oslo

Sammendrag

Den moderne biologien formes av statistikk og nye metoder for å analysere de store datamengdene fagfeltet består av. Dette gjør at samfunnet trenger biologer med tilstrekkelig eksperimentell og teoretisk bakgrunn i både biologi og matematikk. I en tidligere studie ble det funnet at biologistudenters strategiske kunnskaper i problemløsning var svake i møte med beregningsorientert biologi, og at dette kan henge sammen med studentenes manglende kompetanse i matematikk. I lys av at det ble innført krav om forkunnskaper i matematikk R2 på bachelorprogrammet i biovitenskap i 2019, er videre forskning på matematikkferdigheter og biologistudenters problemløsningsstrategier av interesse. Formålet med denne studien er derfor å undersøke hvordan kompetansen fra matematikk R2 gjør seg gjeldende i biologistudenters bruk av problemløsningsstrategier i beregningsorientert biologi.

Studien tar utgangspunkt i et rammeverk for problemløsningsstrategier kjent fra matematikk og *computational thinking*. Informantene i studien er 13 biologistudenter som tok BIOS1100 høsten 2019, et obligatorisk innføringsemne i beregningsorientert biologi for bachelorstudenter på biovitenskap. Dette er en kvalitativ studie, og datamaterialet er hentet inn ved observasjon av gruppearbeid og semistrukturert fokusgruppeintervju.

Studentene som deltok i studien er stort sett positive til at det har blitt R2-krav på utdanningen, og anser de strategiske kunnskapene fra matematikken som relevante. Funnene fra denne studien viser at studentene benytter seg av problemløsningsstrategier kjent fra matematikk og computational thinking når de løser oppgaver i beregningsorientert biologi. Studentene forteller derimot at det er vanskelig å løse mer sammensatte oppgaver, hvilket kan henge sammen med utfordringer knyttet til bruk av de ulike problemløsningsstrategiene. Dette gjenspeiles også når studentene forteller om hvordan de arbeider i et mer tradisjonelt biologiemne de tar parallelt med BIOS1100. Ut ifra hva studentene fortalte i intervjuene, benytter de seg ikke av problemløsningsstrategiene kjent fra matematikk og computational thinking når de arbeider med mer tradisjonell biologi.

I denne studien argumenterer jeg for at kompetansen fra matematikk R2 er relevant for studentene på biovitenskap, og at undervisning av problemløsningsstrategier bør inngå som en del av bachelorutdanningen.

Forord

Arbeidet med denne masteroppgaven har foregått i en uvanlig tid. Koronasituasjonen har vært både fortvilende og inspirerende, og har satt sitt preg på arbeidet. Det har vært et spennende og lærerikt semester, om noe annerledes enn det jeg forventet. Arbeidet med masteroppgaven har gitt meg ny motivasjon og vilje til å utdanne meg videre innen matematikk, og jeg gleder meg til å ta med meg lærdommen fra dette semesteret inn i læreryrket.

Først og fremst vil jeg rette en stor takk til mine tre veiledere, Lex Nederbragt, Maria Vetleseter Bøe og Tone Fredsvik Gregers. Deres fleksible veiledning over zoom og i google disk har vært helt uvurderlig i en tid der fysisk oppmøte ikke har vært en mulighet.

Jeg vil også takke min mor Berit Rudberg og hennes samboer Bård Eiker. Hadde det ikke vært for at dere stilte loftstuen til disposisjon som et provisorisk hjemmekontor, er det tvilsomt at oppgaven hadde blitt levert til normert tid. Dere har vært viktige støttespiller i en ellers annerledes hverdag!

Til slutt vil jeg takke mine tre studievenninner Guro Gustavsen, Kaja Herje Bergrem og Juliane Lauen Singstad. Mangfoldige lunsjer med dere over zoom har gjort at hjemmekontoret ikke ble fult så tungt, og deres innsikt fra matematikkdidaktikken har vært et stort bidrag til oppgaven.

--
Sofie Rudberg

Innholdsfortegnelse

1 Innledning

1.1 Bakgrunn og tidligere forskning

Den 12. mars 2020 satte regjeringen i gang kraftige tiltak mot det som for mange oppleves som den største biologiske krisen i moderne tid. Verden var rammet av en pandemi. I løpet av kort tid florerte nettaviser, sosiale medier og nyhetssendinger med simuleringer av hvordan Norge ville påvirkes av koronaviruset. Vi kunne høre statsminister Erna Solberg fortelle om inkubasjonstid, immunitet og smitterate, og hvordan Norge skulle stå sammen for å undertrykke viruset og minimere R-verdien. Vi ble vist grafer og matematiske modeller som skulle simulere smittespredningen og antall dødsfall, som videre ble utgangspunktet for de smittereduserende tiltakene regjeringen valgte å sette inn. Koronasituasjonen krevde mye av mange, og førte til stor dugnadsånd og tverrfaglige samarbeid i alle disipliner. Ett slikt tverrfaglig samarbeid kommer til syne i simuleringene av smittespredningen, der kunnskap fra biologi, matematikk og programmering forenes for å lage gode modeller.

I kronikken *Koronaepidemien – litt matematisk oppklaring midt i depresjonen* beskriver professor Steinar Engen (2020) koronasituasjonen ved hjelp av en modell for aldersstrukturerte populasjoner. Denne modellen er kjent for de fleste biologer, og er en av mange matematiske modeller som brukes i biologi. Matematikken har blitt en stadig større del av biologien og åpner for at nye metoder kan tas i bruk. Tidligere har biologien vært preget av det kvalitative, der man i stor grad har beskrevet, systematisert og kategorisert det man har observert. Slik er det ikke lenger. «En 400 år lang periode hvor de fleste biologer kunne forstå sitt fagområde uten nevneverdig formell skolering i matematikk nærmer seg slutten» (Omholdt, 2008, s. 18). De biologiske problemstillingene vi står ovenfor i dag vil umulig kunne løses uten matematiske metoder, og vi er avhengige av å utdanne en ny generasjon biologer med tilstrekkelig eksperimentell og teoretisk bakgrunn i både biologi og matematikk (Omholdt, 2008).

I yrkesbeskrivelsen på Utdanning.no beskrives biologen som en som jobber med dyre- og planteliv både i felt og på laboratoriet. Av personlige egenskaper bør man være interessert i mangfold i naturen, være nysgjerrig og kunne tilegne seg kunnskap basert på fysiske bevis (Utdanning.no, 2020). Formålet med programfaget biologi i videregående skole er å danne et grunnlag for å bruke biologisk kunnskap i ulike sammenhenger, der biologisk kunnskap inkluderer alt fra mikronivå i cellene til samspillet i jordens økosystemer. Naturen skal tas i

bruk som læringsarena, og man skal arbeide både praktisk og teoretisk (Utdanningsdirektoratet, 2006a). Slik biologi er beskrevet som fagområde og yrke legges det fremdeles stor vekt på det kvalitative og deskriptive i biologien. Dette gjenspeiles i læreplanmålene i programfaget, der elevene skal lære å beskrive, forklare og gjøre rede for prosesser og fenomener fra de 10 hovedområdene (Utdanningsdirektoratet, 2006a). Til tross for at både regning og bruk av digitale hjelpemidler er beskrevet som grunnleggende ferdigheter i biologi, benyttes det i liten grad denne typen analytiske metoder i faget. Dette kan bidra til holdningene om at matematikk ikke er en sentral del av biologien.

Høsten 2017 innførte Det matematisk-naturvitenskapelig fakultet ved Universitetet i Oslo nye studieprogrammer ved alle sine institutter. For bachelorprogrammet i biovitenskap innebar dette at alle studentene skulle ha et emne i programmering og beregningsmodeller det første semesteret i sitt studieløp. Deretter skal beregninger integreres som en naturlig del av de videre emnene i studiet. BIOS1100 skal gi en innføring i å lage og eksperimentere med enkle modeller av biologiske systemer gjennom programmeringsspråket Python (Institutt for Biovitenskap & Universitetet i Oslo, 2017). I bachelorprogrammet brukes matematiske beregninger og modellering til å forstå prosesser og fenomener i naturen, og ved å lære studentene å benytte seg av disse verktøyene, kan de allerede fra første semester eksponeres for reelle problemstillinger fra forskningsfronten (Dæhlen, 2016). Som Omholdt beskrev i 2008, er vi avhengige av å utdanne en ny generasjon biologer med bakgrunn i både biologi og matematikk. Matematikk er et sentralt verktøy for kunnskapsutvikling og forståelse i teknologi og realfag, herunder biologi. Dette uttrykkes også av dekan Morten Dæhlen ved Det matematisk-naturvitenskapelig fakultet: «Matematikk er også et fag som gir oss trening i å tenke logisk, noe som gjør at matematikken, både direkte og indirekte, har betydning for fag som i utgangspunktet ikke inneholder matematikk, som for eksempel biologi og geologi» (Dæhlen, 2016).

Universitetet i Oslo har i lang tid anbefalt studenter som søker seg til de realfaglige utdanningene å ha full fordypning i realfagsmatematikk fra videregående skole, altså bestått matematikk R1 + R2 (Dæhlen, 2016). I 2016 innvilget kunnskapsdepartementet en prøveordning der universitetene fikk sette R2 som opptakskrav til sine utdanninger innen realfag og teknologi der de selv mente det var relevant. Denne prøveordningen trådte i kraft i opptaksåret 2018, men for bachelorprogrammene i biovitenskap ble innføringen av R2-kravet utsatt til 2019 (Kunnskapsdepartementet, 2016). Det nye opptakskravet førte til at langt færre studenter søkte seg til bachelorutdanningene i biovitenskap på samtlige av de norske

universitetene i 2019 (Samordna opptak, 2019). Det lave søkertallet gjorde at Universitetet i Bergen valgte å trekke seg fra prøveordningen, og fra 2020 er det ikke lenger krav om R2 på deres utdanninger i biovitenskap. Visedekanen for utdanning ved det matematisk-naturvitenskapelige fakultet i Bergen uttalte i et intervju med *Khrono* at matematikkravet ikke er nødvendig, fordi studentene kan lære matematikken etter at de har begynt på bachelorprogrammet. «Grunnen til at vi ser at det er mulig å fjerne R2-kravet fra en del programmer er at vi ser på det som svært mulig å lære studentene denne matematikken etter at de har begynt hos oss» (Hystad & Tønnesen, 2019).

Universitetet i Oslo velger derimot å fortsette med prøveordningen, til tross for de lave søkertallene i 2019. Utdanningsdekan Knut Mørken ved Det matematisk-naturvitenskapelige fakultet understreker i sitt intervju med *Khrono* at det er faglige årsaker til at matematikkravet er ønsket av universitetet: «Den faglige utviklingen går mot at kvantitative metoder og modellering blir viktigere i alle fag og dermed i bachelorutdanningene, og dette er hovedargumentet for å øke matematikkravet» (Hystad & Tønnesen, 2019).

I tillegg påpeker Mørken at denne faglige utviklingen også skal møtes ved hjelp av de integrerte emnene i programmering og modellering som ble innført på alle bachelorprogrammene i realfag i 2017 (Hystad & Tønnesen, 2019). For studentene på biovitenskap betyr dette at både matematikk og programmering blir en sentral del av deres studiehverdag, både direkte og indirekte. Ferdighetene de nå skal lære i første semester åpner for at de kan jobbe med simuleringer fra reelle biologiske problemstillinger, som de vi har sett i forbindelse med koronasituasjonen. Videre åpner tverrfagligheten for nye tilnærminger til biologi som fagfelt, der de kvantitative metodene nå står i fokus.

Den moderne biologien formes av statistikk og nye analysemetoder av de store datamaterialene fagfeltet består av. En ny retning i biologien er i ferd med å forandre fagfeltet ved å gjøre ny kunnskap tilgjengelig gjennom beregninger og modellering. I *computational biology* tas datamaskiner og datavitenskap i bruk for å forstå og modellere strukturene og prosessene i levende organismer (Searls, 2018). Dette innebærer at metodene og tilnærmingen til fagfeltet er i endring. «All biology is computational biology», hevder Markowetz (2017, s. 1), og argumenter for at *computational thinking* er helt avgjørende i søken etter å forstå levende systemer. Computational thinking er en metode for problemløsning kjent fra datavitenskap, men som er svært overførbar til en rekke andre fagfelt, også biologi. Det er for eksempel store likheter mellom kompleksiteten til systemer med kunstig intelligens og

informasjonsprosesseringen i levende systemer (Priami, 2007). Computational thinking kan dermed være et svært hjelpsomt verktøy for å besvare de store spørsmålene i biologi.

I en studie gjort av Håland (2019) ble det, med utgangspunkt i et rammeverk for computational thinking, undersøkt hvordan studenter arbeider når de programmerer i introduksjonskurs i biologi. Gjennom oppgaveløsning og gruppeintervjuer ble det funnet at biovitenskapelige problemstillinger påvirket studentenes programmering positivt ved at studentene får en pekepinn på hva svaret i oppgaven skal bli. Videre ble det funnet at studentene opplever de samme typer utfordringer som man ser i andre introduksjonskurs i programmering. Disse utfordringene knytter seg særlig til det å skrive store programmer og å finne ut av hva programmeringsoppgavene spør om. Håland (2019) fant også tegn på at studentenes strategiske kunnskaper var svake, der det i liten grad ble identifisert strategier kjent fra computational thinking. I stedet veksler studentene mellom strategiene prøv- og feil og bruk av eksempelprogrammer når de programmerer. Ifølge Håland (2019) ser dette ut til å henge sammen med mangelen på konseptuelle kunnskaper, og manglende forkunnskaper i matematikk i form av problemløsningsstrategier. Da Håland gjennomførte sin studie høsten 2018, var det ikke R2-krav på bachelorutdanningen i biovitenskap. I lys av at R2-kravet ble innført for første gang i 2019, foreslår Håland (2019) at videre forskning på matematikkferdigheter og biologistudenters problemløsningsstrategier er av interesse. Det er denne tråden jeg ønsker å plukke opp i min masteroppgave.

Per nå er det gjort lite forskning på sammenhengen mellom forkunnskaper i matematikk og studenters arbeid med biologi og programmering. Sadler og Tai beskriver i artikkelen *The Two High-School Pillars Supporting College Science* at studenters prestasjon i introduksjonsemner i realfag avhenger av tidligere erfaringer med fagfeltet, og «more advanced study of mathematicks in high school» (Sadler & Tai, 2007, s. 458). Samtidig viser en litterær metaanalyse av studenters læring i introduksjonskurs i programmering at evnen til å anvende problemløsningsstrategier er helt avgjørende for å lære å programmere, og at studenters bakgrunnskunnskap i matematikk er av betydning (Medeiros, Ramalho & Falcão, 2018). I tillegg viser resultatene fra Norsk Matematikkråds forkunnskapstest at mange begynnerstudenter strever med grunnleggende ferdigheter knyttet til blant annet algebra og problemløsning (Nortvedt & Bulien, 2018). Forkunnskapstestene gjennomføres primært av begynnerstudenter med minimum 60 studiepoeng matematikk i sin utdanningsplan, men ut ifra det Håland (2019) fant i sin studie, er det god grunn til å tro at biologistudenter også har svake forkunnskaper i matematikk. I og med at forrige forkunnskapstest, og studien til

Håland, ble gjennomført før det var krav om R2 på realfagsutdanningene i Norge, er det lite forskning på hvilken betydning kompetansen fra R2 har for studentene. Gjennom denne masteroppgaven ønsker jeg derfor å sette fokus på hvordan kompetansen fra realfagsmatematikk kan være relevant for studenter på bachelorprogrammet i biovitenskap, ved å se på deres bruk av problemløsningsstrategier i beregningsorientert biologi.

1.2 Emnet BIOS1100

BIOS1100 – Innføring i beregningsmodeller for biovitenskap, er et emne på 10 studiepoeng som skal gi en innføring i å lage og eksperimentere med enkle modeller av biologiske systemer. I emnet får studentene utforske problemstillinger hentet fra genetikk, evolusjon, økologi og bioinformatikk i et virtuelt biologisk laboratorium. Læringsmålene i BIOS1100 er hentet fra emnesiden på nett (Institutt for Biovitenskap & Universitetet i Oslo, 2017), og læringsutbyttebeskrivelsen er som følger:

> Etter å ha fullført emnet
>
> - har du fortrolighet med Python-programmering og kan bruke datastrukturer, funksjoner og moduler, samt løkker og betingelsestester
> - kan du organisere biologiske data, lese og skrive slike data til/fra fil og lage grafiske framstillinger
> - kan du modellere biologiske systemer med hjelp av vektor- og matrisearitmetikk og grunnleggende sannsynlighetsregning
> - kan du dokumentere, presentere og formidle enkle modeller av biologiske systemer

Gjennom emnet skal studentene få en innføring i programmeringsspråket Python. Dette er et objektorientert språk, som typisk brukes til automatisering, dataintegrering og dataanalyse (Dvergsdal, 2019). Python brukes i mange begynneremnene i programmering fordi det er godt egnet til å sette sammen raske løsninger med avansert funksjonalitet. Programmet kan utvides med en rekke kodebiblioteker med ulike programstrukturer, hvilket gir god tilgang på ferdigproduserte funksjoner.

BIOS1100 er et emne som går hvert høstsemester, og strekker seg over 14 undervisningsuker. Undervisningen på emnet besto i 2019 av totalt åtte undervisningstimer per uke, hvorav fire av disse er frivillige. Det frivillige tilbudet besto av to timer forelesning per uke, der nytt stoff ble gjennomgått, og to timer med samkoding. I samkodingen viser en gruppelærer hvordan

man skriver programmer og programelementer, og studentene skriver det samme på sin maskin. Dermed får de testet det nye programmet direkte. Det ble også gjennomført noe samkoding i de fire obligatoriske gruppetimene studentene hadde ukentlig. I gruppetimene ble noen av eksempeloppgavene vist ved hjelp av samkoding for å illustrere hvordan man kan løse oppgavene studentene jobbet med. Formålet med gruppetimene var å la studentene jobbe med ukens pensum i form av oppgaver, der studentene hadde tilgang på gruppelærer og mulighet til å jobbe sammen med medstudenter. Gruppetimene var lagt opp slik at de to første timene var organisert undervisning, mens de to siste var satt av til selvstendig arbeid. I tillegg hadde studentene tilgang til et detaljert kompendium der relevante programelementer forklares og vises med eksempler. Utover undervisningen i Python, ble tre av kursukene satt av til matematikkundervisning. Studentene jobbet med temaene *Funksjoner, Rekursive systemer, Grenser* og *Dynamiske systemer* ut ifra et kompendium i matematikk som er utarbeidet for emnet.

Parallelt med BIOS1100 tar alle førsteårsstudentene på biovitenskap et emne i celle- og molekylærbiologi, BIOS1110, som også er på 10 studiepoeng. Emnet tar for seg de viktigste biologiske molekylene og prosessene disse er involvert i, cellens oppbygning og funksjon og genetikk. Oppgavene i BIOS1100 har, der det har vært relevant, tatt utgangspunkt i samme tematikk som BIOS1110 for å anvende fagstoffet i flere kontekster. Førsteårsstudentene på biovitenskap har i tillegg et emne på 10 studiepoeng i kjemi sitt første semester på studiet.

1.3 Matematikk R2

Programfaget matematikk R2, matematikk for realfag, gir fordypning i matematikk for videre studier og arbeid innen blant annet naturvitenskap, medisin og teknologi (Utdanningsdirektoratet, 2006b). Programfaget bygger på matematikk R1, og er det mest avanserte matematikkfaget i norsk videregående skole. Den realfaglige matematikken er per nå det eneste matematikkfaget i videregående skole der problemløsningsstrategier vektlegges: «Arbeid med programfaget skal gi en innføring i logisk og analytisk tankegang med vekt på matematisk argumentasjon og framstillingsform, samtidig som elevene gjennom anvendelse får trening i sentrale metoder» (Utdanningsdirektoratet, 2006b, s. 1).

I 2021 og 2022 innføres det nye læreplaner for programfaget i matematikk for realfag, R1 og R2. Disse er ikke ferdig utarbeidet, men kjerneelementene for faget er vedtatt. Også i de nye læreplanene vil problemløsningsstrategier stå i fokus, der elevene blant annet skal lære modellering, abstraksjon og generalisering (Utdanningsdirektoratet, 2019). Algoritmisk

tenkning og programmering skal også inngå som en del av undervisningen. Beskrivelsen av programfaget i matematikk for realfag er foreløpig som følger:

> Matematikk R er et sentralt fag for å kunne forstå moderne anvendelser av matematikk i realfaglige og samfunnsmessige sammenhenger. Faget gir elevene mulighet til å utvikle et presist språk for kritisk tenkning, evne til problemløsing og matematisk forståelse. Matematikk R vektlegger verktøy i og anvendelser av matematikk i problemstillinger knyttet til realfag. Faget gir elevene mulighet til å utvikle problemløsingsstrategier som forbereder dem til videre arbeid og utdanning som stiller krav til matematisk forståelse gjennom teoretiske anvendelser av matematikk. (Utdanningsdirektoratet, 2019, s. 7)

1.4 Hensikt og forskningsspørsmål

Formålet med denne studien er å få et innblikk i hvilken betydning full fordypning i realfagsmatematikk fra videregående skole har for studentene som tar BIOS1100. R2-kravet kan ha stor betydning for studentene i form av den logiske tankegangen matematikken trener elevene opp i. Mitt ønske er derfor å undersøke hvilken betydning kompetansen fra R2 kan ha for biologistudenters problemløsningsstrategier i beregningsorientert biovitenskap.

Problemstillingen i denne oppgaven er derfor:

Hvordan gjør kompetansen fra R2 seg gjeldende i problemløsningsstrategier hos studenter i arbeid med beregningsorientert biologi?

For å besvare problemstillingen har jeg reist følgende forskningsspørsmål:

1. Hvilke strategier kjent fra matematikk og computational thinking bruker studentene når de programmerer?
2. Hvilke utfordringer knyttet til bruk av problemløsningsstrategier møter studentene når de programmerer?
3. Hvordan vurderer studentene relevansen av kompetansen de fikk fra R2 inn i BIOS1100?
4. Hvilke tanker har studentene om bruk av strategier i arbeid med fagstoff i BIOS1100 og BIOS1110?

Ettersom hovedargumentet for å øke matematikkravet er den faglige utviklingen med kvantitative metoder og modellering, ønsker jeg å se på betydningen av kompetansen fra R2 i

en kontekst der disse metodene brukes. Derfor har jeg valgt å undersøke hvilke problemløsningsstrategier studentene benytter seg av når de skal arbeide med beregningsorientert biologi i BIOS1100. Datamaterialet i oppgaven vil bestå av observasjon av fire grupper som løser en programmeringsoppgave gitt i emnet, og et påfølgende intervju.

1.5 Avgrensning av oppgaven

Grunnet studiens omfang og valg av metode er det aspekter ved problemstillingen jeg ikke har anledning til å undersøke. Jeg kommer ikke til å se på hvordan R2-kravet påvirker læringsutbyttet til studentene, eller gå i dybden på det programmeringstekniske. Formålet er ikke å undersøke hvor godt studentene programmerer eller hvilke programstrukturer de velger å bruke, men hvilke problemløsningsstrategier de benytter seg av underveis i programmeringen. Jeg må også understreke at studien vil ta for seg hvordan kompetansen fra R2 gjør seg gjeldende i studentenes bruk av problemløsningsstrategier. Jeg vil dermed ikke ta for meg hvordan man skal arbeide med disse strategiene, eller hvordan undervisningen bør legges opp for å fasilitere bruken av problemløsningsstrategier.

BIOS1100 tar for seg beregningsmodeller i biovitenskap, men det er først og fremst programmeringsdelen av emnet jeg fokuserer på. Dette skyldes at problemløsningsstrategiene jeg ønsker å undersøke hovedsakelig er kjent fra computational thinking, og dermed er det mest hensiktsmessig å ta utgangspunkt i en oppgave der studentene arbeider med programmering.

Min studie blir en forlengelse av forskningen Håland gjorde i 2019. Hensikten med hans studie var å undersøke hvordan studentene på BIOS1100 arbeidet når de programmerte med biologiske problemstillinger, der han blant annet så på hvilke strategier studentene brukte i programmeringen og hvilke utfordringer de møtte på. Selv om to av mine forskningsspørsmål er svært like de som ble stilt av Håland, så vil ikke denne oppgaven ha som hensikt å sammenlikne studentenes strategier før og etter R2-kravet ble innført. Min studie skal bygge videre på Håland sine funn, og formålet er å beskrive hvordan kompetansen fra R2 gjør seg gjeldende i biologistudentenes problemløsningsstrategier.

2 Teori

I dette kapittelet vil jeg legge frem det teoretiske rammeverket som blir utgangspunktet for studien. Først vil jeg redegjøre for begrepet «computational thinking» og de ulike komponentene som inngår i konseptet. Deretter vil jeg vise hvilke komponenter fra computational thinking som overlapper med matematisk tenkning, og gi en grundigere redegjøring av disse. Videre vil jeg redegjøre for de to strategiene Håland (2019) identifiserte som de vanligste strategiene hos studentene som deltok i hans studie. Rammeverket som brukes i denne studien presenteres i delkapittel 2.5.

2.1 Computational thinking

Konseptet «computational thinking» ble først introdusert av Seymour Papert i 1980, men har siden den gang blitt gitt flere ulike definisjoner. Jeg vil i denne oppgaven ta utgangspunkt i definisjonen fremstilt av Shute, Sun og Asbell-Clarke (2017, s. 5), som definerer CT som

> *the conceptual foundation required to solve problems effectively and efficiently (i.e., algorythmically, with or without the assistance of computers) with solutions that are reusable in different contexts.*

Computational thinking, heretter omtalt som CT, handler om problemløsning ved hjelp av konsepter og strategier som er relatert til datavitenskap (Grover & Pea, 2018). Det betyr *ikke* at man skal tenke som en datamaskin, men at man skal løse problemer ved hjelp av kognitive prosesser knyttet til problemløsning og problemformulering. For å beskrive hvordan problemløsning ved hjelp av CT ser ut, har Shute et al. (2017) fremstilt et rammeverk basert på komponentene som inngår i majoriteten av forskningen gjort på CT. Som vist i tabell 1, kan CT deles opp i *problemnedbryting, abstraksjon, algoritmer, feilsøking, iterasjon og generalisering.* Det er beskrevet hvilke kognitive prosesser som ligger til grunn for den enkelte komponenten.

Tabell 1 – Beskrivelse av komponentene i CT og deres kognitive prosesser fra Shute et al. (2017), oversatt av Håland (2019).

CT komponenter	Underliggende kognitive prosesser
Problemnedbrytning	Dele opp et komplekst problem/system til håndterlige deler. De ulike delene er ikke tilfeldige, men funksjonelle elementer som kollektivt utgjør hele problemet/systemet.
Abstraksjon	Ekstrahere essensen av et (komplekst) system. Abstraksjon har tre underkategorier: 1. *Datainnsamling og analyse:* Samle inn den mest relevante og viktige informasjonen fra flere kilder og forså forholdet mellom de ulike datasettene. 2. *Gjenkjenne mønstre:* Identifisere mønstre/regler som ligger i dataene/informasjonen. 3. *Modellering:* Bygge modeller eller simuleringer som representerer hvordan et system opererer, og/eller hvordan et system vil fungere i fremtiden.
Algoritmer	Designe logiske og strukturerte instruksjoner for å gi en løsning til et problem. Instruksjonene kan utføres av en datamaskin eller et menneske. Det er fire underkategorier: 1. *Algoritmisk design:* Lage et sett med strukturerte steg for å løse et problem. 2. *Parallellisme:* Utføre flere steg på samme tidspunkt. 3. *Effektivitet:* Designe færrest mulige steg for å løse et problem ved å fjerne overflødige og unødvendige steg. 4. *Automatikk:* Automatisere utførelsen av prosedyren når det kreves for å løse liknende problemer.
Feilsøking	Detektere og identifisere feil og fikse feilen når løsningen ikke virker som den skal.
Iterasjon	Gjenta designprosessen for å raffinere løsningen til et ideelt resultat er oppnådd.
Generalisering	Overføre CT ferdigheter ti l et bredt spekter av situasjoner/domener for å løse problemer effektivt.

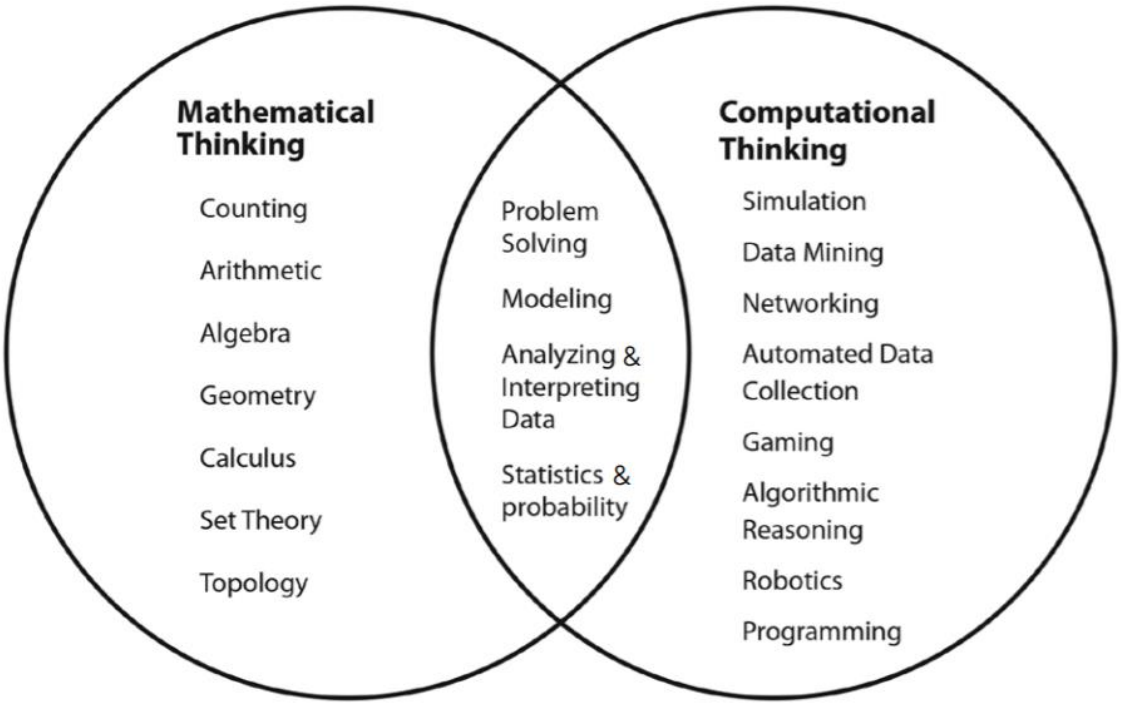
Rammeverket viser viktigheten av systematisk problemløsning. *Problemnedbryting* innebærer å bryte ned et komplekst problem til mindre biter, for så å bruke systematiske prosesser til å angripe hver enkelt del av problemet. *Abstraksjon* handler om å finne mønstre i problemene og i løsningene på disse, mens *algoritmer* gir et verktøy for å løse problemet. *Iterasjon* og systematisk *feilsøking* vil bidra til at den enkelte biten av problemet løses effektivt. Ut fra abstraksjonen av dataene kan man videre *generalisere* løsninger som kan brukes i liknende situasjoner.

Jeg har valgt å ta utgangspunkt i rammeverket til Shute et al. (2017) av tre årsaker. For det første er rammeverket basert på en litterær metaanalyse av forskningsartikler som er relevant for CT, der både definisjonen og rammeverket er basert på funn fra analysen. Det var også dette rammeverket som ble brukt av Håland (2019), hvilket gjør det relevant for min oppgave. Ved å ta utgangspunkt i samme teoretiske rammeverk blir det enklere å sette mine funn inn i den faglige konteksten som forskningen inngår i. I tillegg viser Shute et al. (2017) noen av komponentene som overlapper mellom CT og strategier for problemløsning som er kjent fra matematikk. Disse vil gjennomgås i det neste delkapittelet.

2.2 Computational thinking og matematikk

Matematisk tenkning og CT er to metoder for analytisk tenkning, der begge involverer strategier for hvordan vi løser et problem (Wing, 2008). Selv om matematisk tenkning og CT presenteres som to separate metoder, så henger de svært tett sammen. «The Using Mathematics and Computational Thinking practice is all about finding precise ways to describe the patterns and processes that make up scientific and engineered systems» (Wilkerson & Fenwick, 2017, s. 184).

Matematisk tenkning handler om evnen til å tilnærme seg nye situasjoner og problemer ved hjelp av matematiske ferdigheter. På samme måte handler CT om forståelsen av hvordan en datamaskin kan bidra til å visualisere systemer og løse problemer (Sneider, Stephenson, Schafer & Flick, 2014). Modellering brukes for å redegjøre for systemer fra den virkelige verden, og disse kan spesifiseres matematisk eller ved hjelp av CT, hvilket gjør de to metodene svært sammenflettet (Wilkerson & Fenwick, 2017). I artikkelen *Computational Thinking in High School Science Classrooms* presenterer Sneider et al. (2014) en modell som illustrerer noen av forskjellene og likhetene mellom matematisk tenkning og CT. Denne modellen inkluderer andre konsepter enn rammeverket fremstilt av Shute et al. (2017), men bygger på den samme forståelsen av CT.



Figur 1 – Forskjeller og likheter mellom matematisk tenkning og computational thinking, hentet fra Sneider et al. (2014).

Det vil i tillegg til problemløsning, modellering, analyse og tolkning av data og statistikk og sannsynlighet, være andre elementer i CT som er kjent fra matematikken. Konseptet mønstergjenkjenning er for de fleste kjent fra matematikken gjennom læren om geometriske former og matematiske rekker. I CT kan mønstergjenkjenning føre til en generaliserbar løsning, hvilket igjen overlapper med matematikk (Grover & Pea, 2018). Mønstergjenkjenning er en sentral del av abstraksjon, hvilket også er en strategi kjent fra matematikken, gjennom algebra og tallteori (Grover & Pea, 2018). Abstraksjon kan også relateres til andre komponenter i CT, der både algoritmer og modellering er abstraksjoner (Wing, 2008).

Jeg vil i de neste avsnittene redegjøre for ferdighetene i CT som er direkte relatert til matematikk: problemnedbryting, abstraksjon og algoritmisk tenkning. I tillegg vil jeg inkludere feilsøking og bruk av eksempelprogrammer, da dette var de strategiene Håland (2019) identifiserte som de vanligste strategiene hos studentene på BIOS1100. Jeg har valgt å ekskludere generalisering og iterasjon, da disse komponentene av CT ikke har en direkte overlapp med matematisk tenkning.

2.3 Problemløsningsstrategier kjent fra computational thinking og matematikk

2.3.1 Problemnedbryting og problemløsning

Allerede i 1637 ble problemnedbryting beskrevet av Rene Descartes som en rettesnor for riktig tenkning, der man skal dele problemet opp i så mange biter som mulig, og nødvendig, for å finne løsningen. Problemnedbryting blir sett på som hovedtilnærmingen i CT, og strategien er også kjent fra matematikken. Å bryte et problem ned i mindre deler gjør problemet mer medgjørlig og prosessen mer overkommelig (Grover & Pea, 2018). I kontekst av programmering fører problemnedbryting ofte til at programstrukturene blir skrevet separat, for så å bli satt sammen til et program, altså en løsning på problemet.

Problemløsning er satt sammen av analyse og syntese. Analysen består av problemnedbryting og forståelse av hver enkelt del som problemet brytes ned til. I syntesen konstrueres en løsning basert på løsningen av hvert enkelt delproblem (Ambrósio, Costa, Almeida, Franco & Macedo, 2011). Problemnedbryting er en grunnleggende ferdighet for å forstå og lære programmering. Studenter som ikke har trening i denne delen av problemløsning har vanskelig for å bestemme hvilke deler de skal bryte ned problemet til, og dermed løse problemet (Keen & Mammen, 2015).

2.3.2 Abstraksjon

Abstraksjon er en metode for forenkling og håndtering av kompleksiteten i et problem. Det brukes til å definere mønstre og til å generalisere sammenhenger ved hjelp av parametere. Abstraksjon lar et objekt representere flere andre objekter, og brukes til å fange essensielle egenskaper som er felles for disse, samtidig som irrelevante forskjeller mellom dem skjules (Weintrop et al., 2016; Wing, 2014). For eksempel vil likningen $A = \pi r^2$ representere arealet til alle sirkler, uavhengig av omkrets og andre egenskaper.

I rammeverket til Shute et al. (2017) er abstraksjon delt inn i tre underkategorier. Datainnsamling og analyse omfatter den delen av abstraksjonen der man henter ut relevant informasjon og ser på sammenhengen mellom ulike nivåer. Modellering brukes til å representere hvordan et system fungerer, og hvordan systemet vil fungere i fremtiden. I modelleringen blir problemet forenklet, strukturert og presisert. Videre er gjenkjenning av mønstre en viktig del av abstraksjonen, ved at man gjenkjenner deler av et problem som likner

på noe man allerede har løst (eller programmert) tidligere (Grover & Pea, 2018). I denne konteksten betyr «mønster» informasjon som repeterer seg selv. Når man har gjenkjent et mønster kan man fremstille en løsning, som videre kan anvendes på andre liknende problemstillinger (Anderson, 2016).

Når man jobber med abstraksjoner er det kritisk at den «riktige» abstraksjonen defineres. Prosessen der man bestemmer hvilke detaljer som skal fremheves, og hvilke som skal ignoreres, er hovedgrunnlaget i CT. Abstraksjonsprosessen er bygd opp av nivåer. Hvert nivå må defineres, og forholdene mellom disse må redegjøres for. Dette involverer abstraksjon i hvert nivå, abstraksjon i helheten av nivåene og sammenhengene mellom nivåene. Disse forholdene kan være definert av en funksjon, en simulering, en transformasjon eller en mer generell kartlegging (Wing, 2008). Det er for eksempel lagt inn forhåndsdefinerte programstrukturer i mange programmeringsspråk. Her opererer man på et høyere abstraksjonsnivå, da man kun trenger å ta hensyn til hva kommandoen gjør, og ikke hvordan den fungerer. *Choice* er et eksempel på en slik programstruktur, og hentes fra et programmeringsbibliotek i Python. Når funksjonen brukes korrekt i Python vil den returnere et tilfeldig element fra en liste, uten at det fremkommer hvordan elementet velges.

2.3.3 Algoritmer

En algoritme er en serie av eksplisitte instruksjoner man følger for å løse et problem. Algoritmen har alltid et definert start- og sluttpunkt, og det er opp til problemløseren å bestemme hvilke steg som skal følges for å løse problemet (Anderson, 2016). Grover og Pea (2018) redegjør for tre grunnleggende deler som algoritmer i programmering består av: *sekvens, seleksjon* og *repetisjon*. Stegene fra start til slutt følger en sekvens, men det er ikke uvanlig at algoritmen velger (selekterer) mellom ulike handlinger, som repeteres. Slike valg kan for eksempel forekomme i en *if-else*-test. Noen ganger er det behov for at handlinger repeteres, for eksempel ved at programmet kjører en løkke frem til ønsket resultat er oppnådd.

En algoritme må ikke nødvendigvis løses av en datamaskin (Anderson, 2016; Grover & Pea, 2018). Enhver serie av handlinger som er rettet mot et mål er en algoritme. Bruken av algoritmer er dermed en del av CT som ikke trenger å innebære programmering. Algoritmisk tenkning kan deles opp i underkategoriene *algoritmisk design, parallellisme, effektivitet og automatikk* (Shute et al., 2017). Når man designer algoritmer planlegger man et sett med steg som skal gjennomføres slik at man finner en løsning på problemet. Parallellisme tillater problemløseren å foreta flere steg på samme tid, mens effektivitet fører til at flere steg slås

sammen. Slik kan problemløseren se bort ifra overflødige steg og lage en optimal løsning. Automatikk handler om å gjøre systemet automatisk, for eksempel ved å lage et program som gjennomfører samme operasjon flere ganger. Denne typen automatikk er blant annet vanlig i de ulike løkkene som brukes i Python og andre programmeringsspråk.

2.4 Feilsøking og bruk av eksempelprogrammer

Feilsøking er en prosess der man identifiserer feil, og løser disse, når programmet ikke fungerer som det skal, og er også en komponent i CT (Shute et al., 2017). Feil kan enten være syntaktiske eller konseptuelle. Syntaktiske feil knytter seg til feil bruk og plassering av symboler, som for eksempel mangelen på semikolon, ugyldig indeks eller feil bruk av parenteser (Qian & Lehman, 2017). Problemer med syntaksfeil er svært vanlige, men er stort sett enkle å rette opp (Altadmri & Brown, 2015). Konseptuelle feil er som regel vanskeligere å korrigere enn syntaksfeil, og leder ofte til større utfordringer (Bayman & Mayer, 1983). Den konseptuelle kunnskapen handler om forståelsen for hvordan systemet fungerer, og konseptuelle feil oppstår når programmereren har misoppfatninger knyttet til hvordan strukturene og prinsippene i et program virker (Bayman & Mayer, 1988). I motsetning til syntaksfeil, kan det i tilfeller med konseptuelle feil produseres et program som gir en gyldig output, men et annet resultat enn forventet. Slike feil kalles logiske feil, og kan deles opp i kategoriene *algoritmiske feil, feiltolkninger og misoppfatninger* (Ettles, Luxton-Reilly & Denny, 2018). Algoritmiske feil handler om at valget av fremgangsmåte for å løse problemet i seg selv er feil, og feilen oppstår før programmeringen starter. Feiltolkninger av oppgaven fører ofte til at det produseres en gyldig output, men som gir galt svar. Algoritmiske feil kan skyldes at programmereren sliter med å overføre problemet til en matematisk form (Gomes, Carmo, Bigotte & Mendes, 2006), men hverken algoritmiske feil eller feiltolkning av oppgaven reflekterer den konseptuelle forståelsen av programmet (Ettles et al., 2018). Det gjør derimot misoppfatninger, som fører til feil som skyldes programmererens manglende forståelse. Det er disse feilene som tar mest tid å rette opp.

Eksempelprogrammer er ferdig skrevne programmer som brukes som et utgangspunkt for egen programmering (Müller, Silveira & de Souza, 2018). Håland (2019) fant i sin studie at mange av studentene ser på eksempelprogrammer for å forstå og huske hvordan de skal bruke ulike programstrukturer, og for å forstå programmeringsspråket. Eksempelprogrammer brukes enten som referanse, ved at studentene ser på dem for å lage sine egne programmer, eller som templat i studentenes eget program (Müller et al., 2018). Når studentene bruker

eksempelprogrammene som templat kopierer de enten hele eller deler av programmet for å løse liknende oppgaver. Dette ble av Håland (2019) identifisert som den vanligste bruken av eksempelprogrammer hos studentene på BIOS1100.

2.5 Studiens teoretiske rammeverk

Til denne studien har jeg laget et rammeverk som tar utgangspunkt i teorien presentert av Håland (2019), Sneider et al. (2014) og Shute et al. (2017). Rammeverket består av fem problemløsningsstrategier, der strategiene problemnedbryting, abstraksjon og algoritmer er strategier kjent fra både matematikk og CT. I tillegg inkluderer rammeverket strategiene feilsøking og bruk av eksempelprogrammer. Jeg har valgt å inkludere disse strategiene i rammeverket fordi Håland (2019) i sin studie fant at fjorårets studenter på BIOS1100 brukte disse strategiene, fremfor de problemløsningsstrategiene som er kjent fra matematikk og CT. Det er derfor relevant å undersøke om disse strategiene også brukes av studentene i min studie. Jeg må understreke at strategien feilsøking også er en komponent i CT (Shute et al., 2017), men ettersom den ikke har en direkte overlapp med matematikk, blir den i mitt rammeverk plassert under *øvrige problemløsningsstrategier.* Rammeverket presenteres i tabell 2.

Tabell 2 – Oversikt over problemløsningsstrategier kjent fra matematikk og CT, samt strategiene feilsøking og bruk av eksempelprogrammer, med tilhørende beskrivelse av strategiene. Rammeverket tar utgangspunkt i definisjonene fremstilt av Shute et al. (2017) og Müller et al. (2018).

Problemløsningsstrategier kjent fra matematikk og CT	**Beskrivelse av problemløsningsstrategien**
Problemnedbryting	Dele opp et komplekst problem/system til håndterlige deler. De ulike delene er ikke tilfeldige, men funksjonelle elementer som kollektivt utgjør hele problemet/systemet.
Abstraksjon	Ekstrahere essensen av et (komplekst) system. Abstraksjon har tre underkategorier: 1. *Datainnsamling og analyse:* Samle inn den mest relevante og viktige informasjonen fra flere kilder og forså forholdet mellom de ulike datasettene. 2. *Gjenkjenne mønstre:* Identifisere mønstre/regler som ligger i dataene/informasjonen. 3. *Modellering:* Bygge modeller eller simuleringer som representerer hvordan et system opererer, og/eller hvordan et system vil fungere i fremtiden.
Algoritmer	Designe logiske og strukturerte instruksjoner for å gi en løsning til et problem. Instruksjonene kan utføres av en datamaskin eller et menneske. Det er fire underkategorier: 1. *Algoritmisk design:* Lage et sett med strukturerte steg for å løse et problem. 2. *Parallellisme:* Utføre flere steg på samme tidspunkt. 3. *Effektivitet:* Designe færrest mulige steg for å løse et problem ved å fjerne overflødige og unødvendige steg. 4. *Automatikk:* Automatisere utførelsen av prosedyren når det kreves for å løse liknende problemer.
Øvrige problemløsningsstrategier	
Feilsøking	Detektere og identifisere feil og fikse feilen når løsningen ikke virker som den skal.
Bruk av eksempelprogrammer	Bruke et ferdig skrevet program som utgangspunkt for egen programmering, enten som templat eller som referanse.

3 Metode og analyse

Formålet med denne studien er å undersøke hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier. Informantene i studien var studenter som tok emnet BIOS1100 høsten 2019. Datamaterialet i studien består av lydopptak av fire grupper som løste et sett med programmeringsoppgaver, og fokusgruppeintervjuer med de samme studentene.

I dette kapittelet vil jeg først begrunne valg av forskningsdesign og metode. Deretter vil jeg redegjøre for valg av oppgaver og utforming av intervjuguiden, og gi en beskrivelse av utvalgskriteriene som ble brukt i denne studien. Videre vil jeg gi en grundig redegjørelse av den tematiske analysen, før jeg avslutter kapittelet med en drøfting rundt studiens pålitelighet og troverdighet, og de etiske hensynene som er tatt.

3.1 Valg av forskningsdesign og metode

I denne studien har jeg brukt et kvalitativt forskningsdesign. Hensikten med forskningsdesignet er å sikre en god sammenheng mellom forskningsspørsmålet, datamaterialet og dataanalysen. Ved bruk av kvalitativ metode er målet «å utvikle forståelse av fenomener som er knyttet til personer og situasjoner i deres sosiale virkelighet» (Dalen, 2011, s. 15). Hensikten med denne studien er å undersøke hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier. Ettersom jeg har en åpen tilnærming til forskningsspørsmålet, og primært ønsker å beskrive studentenes bruk av problemløsningsstrategier som fenomen, vil et kvalitativt forskningsdesign være godt egnet (Larsen, 2017).

For å undersøke hvordan studentene på BIOS1100 benytter seg av problemløsningsstrategier og håndterer utfordringer knyttet til programmeringsoppgaver i biologi, har jeg valgt å observere mindre grupper mens de løser en oppgave utformet for emnet. Observasjon er en god metode for å få innsikt i hvordan studentene går frem når de skal løse en oppgave. Under datainnsamlingen har jeg gjennomført en ustrukturert observasjon, som kjennetegnes ved at man først og fremst registrerer detaljer som kan være interessante for å besvare problemstillingen (Kleven, 2014). I min studie innebar dette å notere ned observasjoner som jeg ville diskutere med studentene i det påfølgende intervjuet.

Min problemstilling tar for seg aspekter knyttet til både studentenes handlinger og refleksjoner, og bruk av observasjon i kombinasjon med intervju vil derfor gi et godt utgangspunkt for å belyse tematikken i studien (Kleven, 2014). Et intervju er godt egnet når hensikten er å utvikle en forståelse av informantenes tanker, følelser og erfaringer (Dalen, 2011). Et kvalitativt intervju kan ha ulik grad av struktur, avhengig av om spørsmålene er forhåndsbestemt eller tilpasset den enkelte intervjusituasjonen (Johannessen, Christoffersen & Tufte, 2016). Et strukturert intervju vil stille færre krav til intervjueren ved at spørsmålene er bestemt på forhånd. Mitt forskningsdesign bærer preg av struktur ved at jeg utarbeidet en intervjuguide i forkant av intervjuene. Samtidig har jeg valgt å stille oppfølgingsspørsmål underveis i intervjuet, og bedt om avklaring rundt observasjonene jeg gjorde da studentene løste oppgaven. Dette er kjennetegn på et ustrukturert intervju (Dalen, 2011; Kleven, 2014). Ettersom min intervjuform kombinerer strukturen fra en intervjuguide med fleksibiliteten fra oppfølgingsspørsmålene, faller den innenfor kategorien *semistrukturert intervju* (Johannessen et al., 2016).

I min studie har jeg valgt å observere og intervjue små grupper av studenter fremfor enkeltstudenter. Dette gjorde at jeg kunne inkludere flere informanter i studien, samtidig som situasjonen jeg gjennomførte undersøkelsen i ble mer autentisk for deltakerne, ettersom de vanligvis jobber i grupper. Jeg har valgt å ta utgangspunkt i fokusgruppeintervjuer fremfor vanlige gruppeintervjuer, fordi det ved bruk av fokusgrupper i større grad tilrettelegges for at deltakerne snakker med hverandre (Lerdal & Karlsson, 2008). Ved bruk av fokusgrupper får man tilgang til data som ikke ville oppstått uten den dynamiske interaksjonen i gruppen, og metoden er særlig egnet i studier der man er opptatt av holdninger og hvordan kunnskap anvendes i en bestemt kontekst (Lerdal & Karlsson, 2008). Dette gjør metoden godt egnet til å belyse problemstillingen i denne studien.

For å dokumentere hvordan studentene arbeidet med oppgaven har jeg tatt opptak av skjermen til PC-en de arbeidet på. Dette gjorde at jeg kunne gå tilbake å se hvordan gruppene valgte å løse oppgaven steg for steg. I tillegg har jeg tatt lydopptak av både oppgaveløsningen og intervjuet. Ved å bruke opptak kan man systematisk gå igjennom datamaterialet flere ganger uten at detaljer faller bort, hvilket gjør at man kan oppdage aspekter ved fenomenet som er umulige å se *in situ* (Blikstad-Balas, 2017). Ved bruk av lydopptak i forskning må man ta etiske hensyn ovenfor informantene. Dette vil jeg diskutere nærmere i avsnitt 3.4.3.

3.2 Oppgave og intervju

3.2.1 Valg av oppgave

Ettersom hensikten med studien er å undersøke hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier i beregningsorientert biologi, ble det valgt en oppgave fra pensum i BIOS1100 som i stor grad integrerer biologi og programmering. Jeg har valgt å ta utgangspunkt i samme oppgave som Håland (2019), da den legger til rette for bruk av ulike problemløsningsstrategier i de forskjellige deloppgavene, og dermed egner seg til å belyse min problemstilling. Å ta utgangspunkt i samme oppgave som tidligere forskning, gir også spennende muligheter for sammenligning av resultater, selv om dette ikke er hensikten i denne studien. Oppgaven har blitt utvidet med ytterlige tre deloppgaver, for å tilrettelegge for bruk av flere problemløsningsstrategier og programstrukturer. Den originale oppgaven hadde en feil i oppgave b), som er rettet opp for bruk i denne studien.

Oppgaven (figur 2 og vedlegg C) har et oppsett som studentene allerede er kjent med, da programmeringsoppgaven inngår som en del av oppgavesettet som er utviklet for BIOS1100. Dette bidrar til at situasjonen der undersøkelsen gjennomføres er mest mulig lik studentenes vanlige gruppetime. Oppgaven består av seks spørsmål (oppgave a-f), som etterspør kompetanse i enten biologi eller biologi og programmering. Temaet i oppgaven er genetikk og arv, og studentene må blant annet ha kunnskaper om hvordan recessive sykdommer nedarves. Oppgaven ber studentene om å lage et program der de skal finne genotypene og fenotypene for først fem, og senere tusen barn, og deretter legge til en teller som viser fordelingen mellom syke og friske barn. For å kunne gjøre dette må de i tillegg ha kunnskaper om sannsynlighetsregning og ulike løkker som brukes i Python.

Exercise 8: Autosomal Recessive Inheritance

Autosomal recessive disorder appears in individuals with two abnormal copies of a gene. The genes are located on an autosome, which is a chromosome that is not a sex chromosome. Humans usually have 22 pairs of autosomes and one pair of sex chromosomes. The inherited trait of an autosomal recessive disorders usually skip a few generations, while dominant traits are usually present in every generation. Therefore, the individual that inherits an autosomal recessive disorder must inherit the abnormal alleles from parents that are carriers or from parents where one of them is a carrier and the other has the disorder. Examples of autosomal recessive disorders are cystic fibrosis, sickle cell disease, and phenylketonuria.

This exercise is about a three generation family: Grandparents, parents and grandchildren. The grandparents are the parents of the parent mother.

a) The parents and grandparents show no signs of the autosomal recessive condition, however one of the grandchildren is affected by an autosomal recessive disorder. Given this information, can you identify the genotypes of the parents and the grandparents?

b) Given that the parents are carriers and we do not know anything about the grandchildren, what is then the probability of having three affected grandchildren? Calculate the probability using Python and print the answer to screen.

c) Now that you are familiar with the probabilities of inheriting the disease allele, you are going to check the genotypes and phenotypes of 5 grandchildren. You only need to consider the parental alleles and their offspring.

Write a program that iterates over the 5 children and randomly generates the child's genotype from the parental alleles. Remember, if the child has the dominant allele present it will possess a normal phenotype. Print the genotypes and corresponding phenotypes for each child to screen.

d) Run the program you wrote a couple of times. Do you get the same output each time? Is this expected?

e) Add a counting step to your program where you count the number of affected and normal offspring (you are free to reuse your previous code). Print the probability of having affected children and the probability of having normal children to screen. Do they correlate with the true probabilities?

Try this for 1000 children and calculate the same probabilities, were they different for 5 and 1000 children?

f) Again, run the program you wrote a couple of times. Do you get the exact same output each time? Is this expected?

Figur 2 – Oppgaven studentene løste under observasjonen

3.2.2 Intervjuguiden

I alle intervjustudier er det behov for å utarbeide en intervjuguide som omfatter temaer og spørsmål som studien skal belyse (Dalen, 2011). Ettersom mitt forskningsdesign har mange likheter som forskningsdesignet fremstilt i Håland (2019), valgte jeg å ta utgangspunkt i intervjuguiden som ble brukt i hans studie. Denne ble i 2018 utformet til den samme målgruppen, studenter på BIOS1100. Intervjuguiden besto av fire hovedspørsmål, som alle hadde flere utdypende underspørsmål. I min studie er denne intervjuguiden revidert, men jeg har valgt å beholde tre av hovedspørsmålene fremstilt av Håland. Disse dreier seg om oppgaven studentene arbeidet med, hvordan studentene gikk frem for å løse oppgaven og hvilke utfordringer studentene hadde i møte med oppgaven. I tillegg besto intervjuguiden av spørsmål knyttet til arbeidsvaner i BIOS1100 og erfaringer med programmering. I min intervjuguide (vedlegg A) har jeg også valgt å legge til noen generelle spørsmål i oppstarten av intervjuet. Dette for å etablere relasjon til informantene, og for å starte intervjuet med spørsmål som har lav terskel for å bli besvart. Disse spørsmålene handlet blant annet om interesse for biovitenskap, tidligere erfaringer og tanker om emnet BIOS1100. De generelle åpningsspørsmålene brukes også for å bryte isen, slik at det skal bli enklere å ta del i gruppesamtalen senere på eget initiativ (Johannessen et al., 2016).

Under det første intervjuet valgte jeg å spørre informantene om hvilke tanker de hadde om det nylig innførte R2-kravet. Ettersom dette spørsmålet skapte en diskusjon som er interessant for å belyse problemstillingen i denne studien, valgte jeg å inkludere spørsmålet i de øvrige intervjuene, selv om spørsmålet opprinnelig ikke inngikk i intervjuguiden. Dette vil være markert i vedlegg A. Spørsmålet ble stilt til alle gruppene, og utdrag av svarene vil derfor inngå som en del av mitt datamateriale.

3.2.3 Utvalg og rekruttering av informanter

I kvalitativ forskning er målet å finne informanter som er relevante og interessante for å belyse problemstillingen (Johannessen et al., 2016). Det er ikke et krav om at utvalget skal være representativt, som i kvantitative studier, men det må være hensiktsmessig for studien (Johannessen et al., 2016). For å svare på problemstillingen i min studie ønsket jeg et utvalg bestående av studenter både med og uten tidligere programmeringserfaring. Jeg ønsket også å rekruttere informanter som hadde studert tidligere, og informanter som hadde bachelorprogrammet i biovitenskap som sitt første møte med studielivet. Jeg vurderer disse to kriteriene som hensiktsmessige for min studie fordi tidligere erfaringer vil påvirke studentenes

evner til å anvende problemløsningsstrategier i programmering (Medeiros et al., 2018). I tillegg ønsket jeg å rekruttere informanter fra hver av de fire undervisningsgruppene i BIOS1100, for å ta hensyn til eventuelle forskjeller i undervisningen, og for å rekruttere informanter som vanligvis jobber sammen. Tanken var at dette skulle bidra til økt autentisitet når studentene skulle løse oppgaven.

Ettersom at dette er en kvalitativ studie om hvordan kompetansen fra R2 gjør seg gjeldende i studenters problemløsningsstrategier i beregningsorientert biologi, var det biologistudenter meldt opp til BIOS1100 som ble valgt ut som informanter til studien. Studentene ble først informert om studien i emnets oppstartforelesning. Der holdt jeg et kort fremlegg om hensikten med studien og når den ville finne sted, og hva en eventuell deltakelse ville innebære. Studentene fikk deretter muligheten til å oppgi sin kontaktinformasjon til meg dersom de ønsket å delta, slik at jeg kunne kontakte dem i forkant av hovedrekrutteringen til studien. Det var totalt tre studenter som oppga sin kontaktinformasjon på dette tidspunktet.

Hovedrekrutteringen til studien ble gjort én undervisningsuke før undersøkelsen skulle foregå. For å rekruttere informanter gikk jeg inn i hver av de fire gruppetimene som studentene på BIOS1100 var delt opp i, og holdt et kort fremlegg om undersøkelsen. Her fikk studentene detaljert informasjon om hvordan datainnsamlingen ville foregå, mulighet til å stille spørsmål om studien, og informasjon om at de ville få pizza som en takk for deltakelsen. Studentene som hadde meldt sin interesse i starten av semesteret ble kontaktet på e-post i forkant av dette, slik at de visste når jeg kom innom deres respektive undervisningsgruppe. I etterkant av mitt fremlegg snakket jeg med studentene i deres arbeidsgrupper, og forhørte meg om hvem som kunne tenke seg å delta i studien. Jeg tok her utgangspunkt i arbeidsgruppene der jeg allerede hadde en kontaktperson. Det var kun to av mine kontaktpersoner som hadde mulighet til å delta, og de hørte til i hver sin undervisningsgruppe. De andre studentene som ble rekruttert ble spurt direkte om deltakelse i sin gruppetime.

Da jeg skulle rekruttere informanter til studien fikk jeg ikke anledning til å gjennomføre en strategisk utvelgingsprosess. Av de studentene som hadde et ønske om å delta i studien var det svært få som hadde en timeplan som la til rette for deltakelse. Et av argumentene for å delta i studien var at studentene *ikke* skulle gå glipp av undervisning, samt at de skulle få det samme læringsutbyttet som studentene som ikke deltok i undersøkelsen. Under rekrutteringen anså jeg det som viktigere at intervjuet ble gjennomført umiddelbart etter informantene hadde løst oppgaven, fremfor å rekruttere etter strenge kriterier. Dette førte blant annet til at jeg i svært liten grad klarte å rekruttere studenter som vanligvis jobber sammen, selv om jeg hadde

dette som hensikt. For å bøte på dette ba jeg studentene som skulle delta i undersøkelsen om å programmere sammen fra starten av gruppetimen den aktuelle dagen.

Timeplanen til studentene på BIOS1100 førte i tillegg til at de ble en skjevfordeling mellom kjønnene som deltok i studien. For å rekruttere nok informanter ble jeg nødt til å inkludere alle informantene som både hadde et ønske om å delta, og en timeplan som tillot deltakelse. Jeg lyktes likevel i å rekruttere informanter med ulik utdanningsbakgrunn, og informanter med og uten programmeringserfaring. Det ble rekruttert totalt fire grupper, en fra hver undervisningsgruppe, der gruppene besto av tre eller fire informanter. Informasjon om informantene som deltok i studien er gitt i tabell 3:

Tabell 3 – Informasjon om informantene som deltok i studien

Gruppe	**Antall informanter**	**Antall informanter med tidligere programmeringserfaring**	**Antall informanter som har studert tidligere**	**Kjønnsfordeling**
1	4	1	1	2 kvinner 2 menn
2	3	2	2	1 kvinne 2 menn
3	3	1	1	1 kvinne 2 menn
4	3	2	1	3 menn
Totalt	13	6	5	4 kvinner 9 menn

3.2.4 Pilotering av intervjuguiden

I kvalitative intervjustudier bør det gjennomføres ett eller flere prøveintervjuer i forkant av datainnsamlingen (Dalen, 2011). Hensikten med prøveintervjuene er å teste ut intervjuguiden og det tekniske utstyret, samt trene seg selv i rollen som intervjuer. Ved å ha god kjennskap til intervjuguiden, og være trygg i rollen som intervjuer, vil det være enklere å være tilstede i samtalen under de faktiske intervjuene og stille gode oppfølgingsspørsmål (Tjora, 2017). I forkant av datainnsamlingen gjennomførte jeg tre prøveintervjuer. De to første

prøveintervjuene gjennomførte jeg med to medstudenter, der jeg primært øvde meg på rollen som intervjuer. Det tredje prøveintervjuet gjennomførte jeg med en medstudent som tok BIOS1100 sammen med informantene mine. Han fikk tid til å sette seg inn i oppgaven, og vi gjennomførte hele prøveintervjuet som om det skulle vært en autentisk intervjusituasjon. På denne måten fikk jeg undersøkt om spørsmålene i intervjuguiden fungerte, og testet det tekniske utstyret som skulle brukes i undersøkelsen.

3.2.5 Gjennomføring av gruppeoppgave og fokusgruppeintervju

Alle intervjuene ble gjennomført i løpet av kursuke 10 i emnet. Tidspunktet ble valgt for at studentene skulle ha tilstrekkelige forutsetninger for å løse oppgaven med hensyn på både biologi og programmering, samtidig som det ikke ble for tett opp mot studentenes eksamensperiode. Gruppeoppgaven ble lagt til den siste timen av studentenes obligatoriske gruppeundervisning i emnet, mens intervjuet ble gjennomført umiddelbart etter dette. Datainnsamlingen strakk seg dermed omtrent en time utover oppsatt undervisning for studentene. Gjennomføringen av både gruppeoppgaven og intervjuet foregikk på et annet rom enn der studentene vanligvis har undervisning, for at lyden av andre studenter ikke skulle fanges opp på opptaket. Rommene som ble brukt til undersøkelsen var alle utstyrt med storskjerm som PC-en var koblet til. Dette gjorde at en av studentene kunne sitte med PC-en, samtidig som de andre studentene fikk god oversikt over hva som ble skrevet ned. Det var hensiktsmessig at det kun ble programmert på én PC, da det ble tatt screencast av skjermen under gruppeoppgaven. Oppgaven ble lagt ut i Jupyterhub, et verktøy som studentene bruker i undervisningen når de skal jobbe med oppgaver i Python. Det var også i dette verktøyet studentene løste oppgaven.

Ettersom mitt masterprosjekt foregår parallelt med et annet masterprosjekt knyttet til BIOS1100, ble det opprettet et felles samtykkeskjema som alle studentene signerte i oppstartsforelesningen til emnet. Før studentene begynte med oppgaveløsningen repeterte jeg i korte trekk samtykkeskjemaet de hadde signert på tidligere, for å tydeliggjøre at de når som helst kunne trekke seg, at dataene ville anonymiseres og oppbevares etter gjeldende retningslinjer. Studentene samtykket så muntlig til dette. Videre ble studentene informert om at de når som helst kunne benytte seg av gruppelærer dersom det var behov for det og at de kunne tilkalle gruppelærer ved å sende en kort SMS. Studentene ble oppfordret til å samarbeide og diskutere med hverandre, og jobbe slik som de vanligvis gjør i gruppetimene.

Et generelt trekk ved alle de fire gruppene som deltok var at de løste oppgaven greit, og de hadde relativt god dialog mens de gjorde dette. Alle gruppene ble ferdige med hele oppgaven innenfor en time. I intervjudelen forøkte jeg å stille gode oppfølgingsspørsmål til det som ble sagt, samt flette inn noen spørsmål knyttet til det jeg hadde observert da studentene jobbet med oppgaven. Det var god flyt i intervjuene, der studentene i stor grad diskuterte spørsmålene sammen.

Gruppe 1

Det var studenten med tidligere programmeringserfaring som var ansvarlig for PC-en med oppgaven, som førte til at programmet tidvis ble skrevet ned litt raskere enn hva gruppa rakk å få med seg. HDMI-kabelen som PC-en var koblet til var noe kort, hvilket gjorde at studenten som noterte satt ytterst på rekken. Studentene snakket mye sammen da de løste oppgaven, men ofte snakket jentene sammen og guttene sammen. Dette skyltes sannsynligvis at jentene satt ved siden av hverandre, og guttene ved siden av hverandre på rekken. I tillegg slet den ene jenta med å forstå valget av programstruktur i den ene deloppgaven, hvilke resulterte i at jentene snakket om dette mens guttene gikk videre eller så over programmet. Dette gjorde at det tidvis er vanskelig å høre hva som blir sagt på opptaket, da samtalene foregikk parallelt. Gruppen brukte omtrent 35 minutter på å løse oppgaven, og intervjuet varte i en time.

Gruppe 2

Før studentene begynte å løse oppgaven spurte jeg om noen av dem hadde programmeringserfaring fra tidligere. Jeg erfarte under observasjonen av gruppe 1 at det ville være hensiktsmessig at en uten programmeringserfaring styrte PC-en, slik at hele gruppen til enhver tid hang med på hva som ble skrevet. Det var en student som ikke hadde tidligere programmeringserfaring, og som dermed styrte PC-en. På denne gruppen hadde jeg glemt å informere om at gruppelæreren var tilgjengelig dersom de sto fast. Jeg avbrøt derfor studentene da de begynte på oppgave c) for å informere om dette. Gruppen brukte omtrent en time på å løse oppgaven og en time på intervjuet.

Gruppe 3

En av studentene uten programmeringserfaring styrte PC-en. På denne gruppen brukte den ene studenten notater fra egen PC, hvilket jeg ikke har fått fanget opp på min screencast. Det fremkommer i lydopptaket hva notatene inneholder. Gruppen brukte omtrent 40 minutter på å løse oppgaven, og intervjuet varte i 50 minutter.

Gruppe 4

PC-en ble styrt av studenten uten tidligere programmeringserfaring. Da studentene skulle begynne å løse oppgaven ga en av dem uttrykk for at han til vanlig ville løst oppgave a) med penn og papir. Jeg brøt inn og sa at han gjerne måtte hente hjelpemidler i sekken sin som lå et stykke unna pulten, for at oppgaveløsningen skulle foregå så autentisk som mulig. En av studentene meldte seg litt ut av intervjuet, og deltok i liten grad i gruppesamtalen. Gruppene brukte omtrent 40 minutter på å løse oppgaven, og intervjuet varte i overkant av en time.

3.3 Tematisk analyse

Tematisk analyse er en metode for identifisering, analysering og rapportering av temaer i datamaterialet (Braun & Clarke, 2006). Et tema fanger opp noe i datamaterialet som er viktig for å belyse forskningsspørsmålet og representerer mønstre innad i datamaterialet (Braun & Clarke, 2006). Ideelt sett skal temaene representeres på tvers av dataene, men dette er ikke alltid mulig. Som jeg vil vise i resultatdelen, vil for eksempel temaet «R2-kravet» kun finnes i intervjudataene i min undersøkelse. Et tema vil videre favne over én eller flere koder. Kodene identifiserer trekk ved datamaterialet som er interessante for forskningsspørsmålet, og brukes for å organisere dataene på en hensiktsmessig måte (Braun & Clarke, 2006). Mine temaer består av ulike antall koder, og jeg må her understreke at antallet koder ikke indikerer viktigheten av temaet eller antall ganger temaet ble identifisert i datamaterialet. En oversikt over hvilke temaer og koder jeg har brukt i min analyse vil bli gitt i delkapittel 4.1.

Jeg har valgt å bruke denne analysemetoden fordi den handler om å analysere temaer på tvers av datasett. Dette er hensiktsmessig fordi dataene mine består av både intervjuer og observasjoner av flere grupper. I tillegg er det en godt egnet metode for studier med store begrensninger i tid og omfang, slik som et masterprosjekt på 30 studiepoeng, da den er mer tilgjengelig enn andre metoder, som for eksempel grounded theory (Braun & Clarke, 2006). Tematisk analyse vil også egne seg godt i deskriptive studier slik som denne, der jeg gjennom analysen ønsker å *beskrive* hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier. Metoden er svært fleksibel (Braun & Clarke, 2006), hvilket har vært positivt i mitt analysearbeid, men det gjør også at det kreves en god avklaring på hvordan analysen ble gjennomført. Denne avklaringen vil bli presentert i de neste avsnittene.

I tematisk analyse vil forskeren spille en aktiv rolle når temaene i datamaterialet skal identifiseres og velges ut (Braun & Clarke, 2006). Temaer kan identifiseres på to måter. De kan identifiseres induktivt, ved at man går inn i datamaterialet for å finne temaer som er

interessante og nyttige for å belyse problemstillingen, eller de kan identifiseres deduktivt. Ved deduktiv koding tar man utgangspunkt i eksisterende teori, og bruker denne aktivt i analysen. Den deduktive tematiske analysen har som regel mindre rike beskrivelser av datamaterialet enn hva den induktive analysen har (Braun & Clarke, 2006). I min studie har jeg valgt å gjøre en kombinasjon av de to, der jeg først har kodet datamaterialet induktivt to ganger, og deretter kodet deduktivt en gang. «Researchers must first look at their data in order to see what they should look for in their data» (Sandelowski, 1995, s. 371). Mine induktive koder og temaer ga meg utgangspunktet for hvilken teori studien skulle forankre seg i. Denne teorien ble videre utgangspunktet for de deduktive kodene og temaene. På denne måten har jeg forhåpentligvis fanget opp alle temaene som er interessante og relevante for å belyse problemstillingen i min oppgave.

I artikkelen *Using thematic analysis in psychology* redegjør Braun og Clarke (2006) for seks faser i tematisk analyse. Jeg har tatt utgangspunkt i disse seks fasene i mitt analysearbeid og vil her gi en beskrivelse av hver fase og min gjennomføring av disse.

1. Bli kjent med datamaterialet

   Jeg har selv samlet inn alt datamaterialet til min studie, og begynte derfor å bli kjent med dataene mine allerede i innsamlingsfasen. Det er viktig å bli kjent med datamaterialet for å få et godt utgangspunkt for analysen og for å få en oversikt over mulige mønstre. I mitt arbeid har jeg transkribert dataene selv, der jeg samtidig noterte ned ideer og tanker jeg fikk da jeg gikk gjennom datamaterialet. Hvordan dataene ble transkribert gjennomgås i avsnitt 3.3.1.

2. Generere koder

   I denne fasen genereres de første kodene som brukes på datamaterialet. Under analysearbeidet har jeg benyttet meg av ATLAS.ti, et analyseprogram for kvalitative studier. Jeg startet kodingen med å lage så mange koder som mulig, og kodet induktivt alle segmenter av datamaterialet som jeg anså som interessant og relevant for tematikken i undersøkelsen. Ettersom mange koder oppsto underveis i prosessen, gikk jeg igjennom datamaterialet to ganger for å forsikre meg om at alle segmentene som passet i kodene ble inkludert.

3. Lete etter temaer

   Her skal kodene kategoriseres i potensielle temaer som kan brukes i analysen. Denne fasen dannet utgangspunktet for mine deduktive koder. Jeg kategoriserte mine induktive koder inn i foreløpige temaer, og disse temaene dannet utgangspunktet for teorien i denne oppgaven. Temaene som oppsto fra de induktive kodene dannet også utgangspunktet for to av forskningsspørsmålene i denne studien.

4. Revidere temaene

   Etter at jeg ferdigstilte teorikapittelet lagde jeg meg et rammeverk med temaene jeg ville inkludere fra de induktive kodene, og temaer og koder med utgangspunkt i strategiene beskrevet i teoridelen. Ettersom jeg allerede hadde kodet igjennom datamaterialet to ganger, og hadde fått en god oversikt, kunne jeg velge koder og foreløpige temaer samtidig. Jeg kodet deretter datamaterialet med de deduktive kodene.

5. Definere temaene

   Her skal hver kode og hvert tema defineres og revideres på nytt, slik at dataene som presenteres i analysen representerer essensen av datamaterialet. I mitt arbeid ble dette gjort ved at jeg gjennomgikk min kategorisering av kodene og definerte hver enkelt kode i rammeverket. Rammeverket presenteres i starten av kapittel 4.

6. Skrive rapport

   Den siste fasen av tematisk analyse handler om fremstilling av funnene fra analysearbeidet. I mitt arbeid er denne fasen fremstilt som metode- og resultatkapittel.

Tematisk analyse er ikke en lineær prosess, og de ulike fasene vil derfor gjøres om hverandre (Braun & Clarke, 2006). Jeg har i min analyse særlig beveget meg mellom fase 2 og 3 i prosessen der jeg skulle finne kodene mine, både induktivt og deduktivt, og tilpasse disse til temaene i analysen.

3.3.1 Transkribering

I forkant av analysen ble datamaterialet transkribert fra lyd til tekst. Transkripsjonen blir sett på som en av nøkkelfasene i analysen, da det er her man blir kjent med datamaterialet sitt (Braun & Clarke, 2006). Det er anbefalt at transkriberingen forgår umiddelbart etter intervjuet

(Dalen, 2011), men da jeg i etterkant av perioden for datainnsamlingen hadde egne eksamener og arbeidskrav, ble denne prosessen utsatt med 3 måneder. Ettersom jeg har lydopptak og screencast fra datainnsamlingen kunne jeg enkelt sette meg inn i situasjonen der undersøkelsen foregikk, og følge studentene stegvis i oppgaveløsningen. Jeg anser det derfor ikke som en ulempe at transkriberingen ble utsatt.

Ettersom det i tematisk analyse ikke finnes faste retningslinjer for transkribering, har jeg valgt å inkludere alle utsagn, pauser og non-verbale utsagn i transkripsjonene mine for å best mulig sette sitatene inn i riktig kontekst, og for å unngå å utelate mulige interessante deler av situasjonen. Jeg har i størst mulig grad forsøkt å gjengi nøyaktig hva som ble sagt, men det er enkelte deler av observasjonsopptakene der lyden av samtalen faller bort grunnet parallelle samtaler i gruppa. Dette ble markert i transkripsjonene med korte kommentarer.

3.4 Kvalitet i forskningen

I dette delkapittelet vil jeg drøfte hvilke tiltak jeg har gjort for å styrke kvaliteten ved mitt forskningsprosjekt, og beskrive hvilke etiske hensyn som er tatt i forbindelse med denne studien. Når kvaliteten i en studie skal vurderes, gjøres dette ut ifra forskningens reliabilitet og validitet. I kvalitativ forskning betyr dette at man vurderer påliteligheten til datamaterialet, og at man vurderer datamaterialets relevans for å besvare problemstillingen (Everett & Furseth, 2012). I all forskning bør man ta stilling til spørsmål knyttet til reliabilitet og validitet, men disse begrepene er ikke like meningsfulle i kvalitative studier (Dalen, 2011). Jeg har derfor valgt å bruke begrepene pålitelighet og troverdighet når jeg skal drøfte kvaliteten i denne studien.

3.4.1 Pålitelighet

Pålitelighet i kvalitativ forskning handler om hvor nøyaktig og omfattende forskeren har rapportert valgene som ble gjort (Cohen, Morrison & Manion, 2007). Det stilles ikke samme krav til etterprøvbarhet som i kvantitativ forskning, ettersom resultatene er et produkt av den unike situasjonen der undersøkelsen finner sted (Cohen et al., 2007; Johnson & Christensen, 2017). Det er likevel viktig å redegjøre for metodevalg og forholdene rundt forskningen, slik at andre kan sette seg inn i situasjonen der studien ble gjennomført (Dalen, 2011). I min studie har jeg forsøkt å styrke påliteligheten ved å gi rike og detaljerte beskrivelser av gjennomføringen av datainnsamlingen, rekrutteringen av informanter og analysen av datamaterialet. Videre har jeg styrket påliteligheten ved å bruke lydopptak under

datainnsamlingen, hvilket har gjort at jeg kan gjengi utsagn fra informantene mine korrekt. I resultatdelen har jeg vist til flere utsagn og sitater fra informantene, og deretter beskrevet hvordan jeg har tolket disse. Dette har styrket påliteligheten ved at jeg tydelig viser hva mine tolkninger baserer seg på.

3.4.2 Troverdighet

Troverdighet i kvalitative studier handler om hvorvidt tolkningene som gjøres fra resultatene er korrekte og sannferdige (Johnson & Christensen, 2017), og om «how accurately the account represents participants´ realities of the social phenomena and is credible to them» (Creswell & Miller, 2000, s. 124). Dalen (2011) beskriver fire aspekter ved kvalitativ forskning der troverdigheten må redegjøres for. Troverdigheten må drøftes med hensyn på forskerrollen, forskningsprosjektet, datamaterialet og bearbeidingen av dette, og tolkningen og den analytiske tilnærmingen. Jeg vil videre ta utgangspunkt i disse fire aspektene i drøftingen av troverdigheten i min studie.

Ettersom det i stor grad er forskerens tolkninger av datamaterialet som utgjør resultatene i kvalitative studier, må forskeren redegjøre for sin tilknytning til det som skal undersøkes (Dalen, 2011). En mulig trussel mot troverdigheten i denne studien er *forskerbias,* som ofte oppstår fordi forskerens tolkninger blir påvirket av forskerens egne synspunkter og perspektiver (Johnson & Christensen, 2017). Forskerbias kan forebygges gjennom *refleksivitet*, der forskeren avslører sine egne antagelser og sin førforståelse for tematikken i studien (Creswell & Miller, 2000). Jeg vil argumentere for at en av styrkene ved meg i forskerrollen i denne studien er at jeg ikke har tidligere erfaringer med matematikk R2, da jeg valgte S1 + S2 på videregående. Jeg har også lite erfaring med programmering, da jeg kun har vært borti programmering i forbindelse med ett emne på 10 studiepoeng det første semesteret av utdanningen. Dermed gikk jeg inn i studien med færre synspunkter på bruk av problemløsningsstrategier kjent fra matematikk og CT enn hva jeg ville gjort dersom jeg kjente til arbeidsmetodene i R2 og programmering. Samtidig var jeg bevisst på at jeg hadde noen forventninger om at R2 kan være positivt for studentene på biovitenskap, fordi jeg i løpet av min utdannelse har sett behovet for en dypere forståelse av matematikk og matematiske metoder, enn hva jeg selv fikk fra S1 + S2. Jeg har derfor, etter beste evne, forsøkt å ikke la disse forventingene påvirke meg i mitt analysearbeid.

For å drøfte troverdigheten til forskningsopplegget tas det utgangspunkt i datainnsamlingsmetodene, og om disse er tilpasset den aktuelle undersøkelsens mål,

problemstillinger og teoretiske forankring (Dalen, 2011). Formålet med denne studien er å undersøke hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters bruk av problemløsningsstrategier. Jeg har valgt å bruke både observasjon og intervju for å belyse problemstillingen i denne studien. Dette har gjort at jeg kan undersøke *hvordan* studentene bruker problemløsningsstrategier når de løser en oppgave, og *hva de sier* at de gjør når de løser en oppgave (Kleven, 2014). Ved å belyse problemstillingen min gjennom to ulike metoder bruker jeg *metodetriangulering*, som styrker troverdigheten i forskningen (Creswell & Miller, 2000; Johnson & Christensen, 2017).

Troverdigheten i datamaterialet og bearbeidingen av dette vil styrkes ved at forskeren stiller gode spørsmål og legger til rette for at informantene har anledning til å komme med innholdsrike og fyldige uttalelser (Dalen, 2011). Et grep jeg har gjort i min studie for å sikre god oppfølging av informantene er bruken av prøveintervjuer. Ved å være forberedt på min rolle som intervjuer har jeg i større grad kunnet stille gode oppfølgingsspørsmål, slik at jeg best mulig kan beskrive informantenes bruk av problemløsningsstrategier. Dalen (2011) beskriver også bruk av dataprogrammer i kodingsprosessen som et grep for å styrke troverdigheten i datamaterialet. I min studie har jeg valgt å bruke ATLAS.ti som analyseverktøy, som blant annet har hjulpet meg med å sortere de ulike kodene inn i temaer.

I min studie har troverdigheten til tolkningene og den analytiske tilnærmingen blitt styrket gjennom valget av analysemetode. Tematisk analyse er en godt egnet analysemetode for uerfarne forskere, fordi den ikke krever den samme teoretiske tilnærmingen til datamaterialet som andre analysemetoder (Braun & Clarke, 2006). I tillegg vil analysemetoden, gjennom mine beskrivelser i delkapittel 3.3, gjøre min analyse transparent. En mulig svakhet ved tematisk analyse er at det er forskeren som velger temaene i analysen. Som jeg problematiserte tidligere, vil forskerens førforståelse påvirke hva forskeren ser etter i datamaterialet. I min studie har jeg vært dette bevisst, og har etter beste evne hentet ut de temaene i datamaterialet som er interessante, og relevante, for å belyse min problemstilling fra et nøytralt ståsted. Jeg må likevel understreke at tolkningene vil være subjektive, da man som forsker aldri kan fristille seg helt fra sin egen førforståelse.

3.4.3 Etiske hensyn

All forskning skal forankres i etiske verdier, der forskeren har et etisk ansvar ovenfor deltakerne i studien (Everett & Furseth, 2012). Det er særlig tre forskningsetiske hensyn som skal tas i kvalitative studier. Disse handler om fritt og informert samtykke, konfidensiell og

anonym deltakelse, og aktsomhet for deltakerrisiko (Befring, 2015). Jeg vil her beskrive hvordan jeg har ivaretatt informantene som deltok i min studie.

Et informert samtykke innebærer at det gis ærlig informasjon om formålet med, og omfanget av, undersøkelsen (Everett & Furseth, 2012). Informantene har krav på å vite at deltakelsen er frivillig, at de har rett til å trekke seg til enhver tid og at undersøkelsen er anonym (Befring, 2015; Everett & Furseth, 2012). Informantene som deltok i undersøkelsen mottok informasjon om studien ved to ulike anledninger; i emnets første forelesning, og i gruppetimen der jeg rekrutterte informantene. Som jeg beskrev i avsnitt 3.2.3, gikk jeg gjennom undersøkelsens formål og omfang i detalj, samt hva en eventuell deltakelse ville innebære. Jeg åpnet også for at studentene kunne stille spørsmål dersom det var noe de lurte på. Studentene som tok BIOS1100 høsten 2019 signerte på et samtykkeskjema i første forelesningen av emnet i forbindelse med et masterprosjekt som gikk parallelt med mitt prosjekt. Dette samtykkeskjemaet var felles for våre prosjekter, men studentene måtte eksplisitt samtykke til lydopptak i undervisningen og deltakelse i intervju for å kunne delta i min studie, se vedlegg B. I og med at det gikk lang tid mellom utdelingen av samtykkeskjema og gjennomføringen av undersøkelsen, repeterte jeg hovedpunktene i samtykkeskjemaet til informantene før jeg begynte undersøkelsen, og informerte om at de når som helst kunne trekke seg uten grunn.

Alle personlige opplysninger skal behandles anonymt og konfidensielt, hvilket handler om vern av informantenes integritet og privatliv (Befring, 2015). I og med at jeg har benyttet meg av lydopptak i denne studien, og at dette innebærer registrering og lagring av personopplysninger, ble prosjektet meldt inn til Norsk Senter for forskningsdata (NSD). For å møte kravet om anonymitet har informantene fått tildelt hver sin faste bokstav, og blitt referert til på en kjønnsnøytral måte gjennom studien. Lydopptakene har blitt oppbevart på en kryptert minnepinne, som kun jeg har hatt tilgang til, og vil slettes så snart oppgaven er ferdig sensurert.

Å delta i forskning skal være risikofritt, og aktsomhet for deltakerrisiko innebærer derfor forebyggende tiltak for å sikre minst mulig belastning på informantene (Befring, 2015). For å belaste informantene minst mulig valgte jeg en oppgave de allerede hadde på undervisningsplanen, og la undersøkelsen til siste undervisningstime, slik at de fikk samme læringsutbytte som studentene som ikke deltok i undersøkelsen. Som en takk for at studentene ble igjen en time etter oppsatt undervisning for å gjennomføre intervjuet, ble de tilbudt pizza.

4 Resultater

I denne studien undersøker jeg hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier i beregningsorientert biologi. R2-kravet er satt fordi den faglige utviklingen går mot mer kvantitative metoder (Mørken, intervjuet i Hystad & Tønnesen, 2019). For at studentene skal bli kjent med de kvantitative metodene har de et emne i beregningsorientert biologi der de blant annet skal lære programmering i Python. For å svare på problemstillingen ønsker jeg å se på hvilke strategier kjent fra matematikk og computational thinking studentene benytter seg av når de programmerer. Jeg ønsker å finne ut av hvilke utfordringer de møter på, hvordan de vurderer relevansen av kompetansen fra R2 og hvilke tanker de har om arbeidet med programmering i forhold til celle- og molekylærbiologiemnet BIOS1110 som de tar samme semester.

Når jeg presenterer resultatene fra analysen, vil jeg i de to første delkapitlene presentere funn fra både observasjonene og intervjuene sammen. Dette velger jeg å gjøre fordi jeg vil gi en rik beskrivelse av hvilke strategier studentene bruker når de programmerer og hvilke utfordringer de møter på. I de to siste delkapitlene vil jeg utelukkende presentere funn fra intervjuet. Dette fordi observasjonsdataene ikke fanger opp tematikken som skal presenteres. For å gi en oversikt over hvor resultatene er hentet fra, blir sitater hentet fra observasjonen av gruppene markert med *OGX*, der X representerer gruppenummeret fra 1-4. På samme måte blir sitater hentet fra intervjuene markert med *IGX*. Ettersom studentene som deltok i undersøkelsen skal forbli anonyme, blir studentene referert til med hver sin faste bokstav. Studentene betegnes som A-D på gruppe 1, E-G på gruppe 2, H-J på gruppe 3 og K-M på gruppe 4. I rollen som intervjuer har jeg fått betegnelsen S.

I dette kapittelet vil jeg først presentere hvilke temaer og koder jeg har brukt i denne analysen. Jeg har valgt å bruke betegnelsen «kode» for kategoriseringen av datamaterialet, studentenes kommandoer betegnes som «programstruktur» og den ferdige programmeringskoden til studentene betegnes som «program». Etter at jeg har presentert temaene og kodene, presenterer jeg funnene mine. Disse er delt inn i fire delkapitler, der hvert delkapittel tar for seg ett forskningsspørsmål. Jeg har valgt å presentere funn på tvers av temaer i det første delkapittelet, da dette er mest hensiktsmessig for å gi en oversikt over studentenes bruk av strategier og utfordringene de møter på. I de to siste delkapitlene presenterer jeg funn knyttet til de to siste temaene i analysen.

4.1 Temaer og koder i analysen

Temaene og kodene i min analyse har oppstått både induktivt og deduktivt. De induktive kodene som ble brukt er vist i de to siste temaene i tabell 4. De induktive kodene ga meg utgangspunktet for teorien som er presentert i denne studien, og de deduktive kodene er videre laget ut ifra denne. Jeg har valgt å dele kodene inn i temaene som knytter seg til strategier kjent fra matematikk og CT. I tillegg har jeg inkludert temaene *feilsøking* og *Eksempelprogrammer*, da dette er strategier som Håland (2019) i sin studie viste at studentene på BIOS1100 benyttet seg mye av. Temaene og kodene som faller inn under dem er vist i tabell 4.

Tabell 4 – Oversikt over temaer og koder brukt i analysen

Temaer	Koder	Beskrivelse av kode
Problemnedbryting	Problemnedbryting	Problemet brytes ned til mindre og mer håndterlige biter
Abstraksjon	Henter ut informasjon	Henter ut relevant informasjon, ser på sammenhengen mellom ulike nivåer
	Bruker modellering	Problemet forenkles, struktureres og presiseres ved hjelp av modeller eller simuleringer
	Gjenkjenner mønstre	Gjenkjenner deler av problemet som likner på noe man har løst tidligere, eller som gjentar seg i problemet
Algoritmer	Designer algoritmer	Planlegger et sett med steg for å løse problemet
	Automatikk	Automatiserer en operasjon ved hjelp av løkker
	Slår sammen steg	Programmerer mer effektivt ved å slå sammen steg
Feilsøking	Syntaks- og konseptuelle feil	Feil som skyldes skrivefeil, manglende semikolon etc. Feil som skyldes feil plassering eller feil bruk av programstrukturer som if-test, løkker etc.
	Logiske feil	Programmet virker, men gir galt resultat. Kan skyldes feiltolkninger, algoritmiske feil eller misoppfatninger
	Feilsøkingsrutiner	Feilsøking av programmet, reparasjon av feil. Tester om programmet virker
Eksempelprogrammer	Bruker eksempelprogram som templat	Bruker hele eller deler av eksempelprogram som templat for egen koding
	Bruker eksempelprogram som referanse	Refererer til undervisning eller tidligere oppgaver, men kopierer ikke deler av koden
Utfordringer	Utfordringer	Vanskeligheter med å forstå hva oppgaven spør om eller forstå hva som skal settes inn i programmet
R2-kravet	Negativ til R2	Studentene uttrykker negative holdninger til R2 kravet
	Positiv til R2	Studentene uttrykker positive holdninger til R2 kravet
Overføring av strategier til biologi	Strategier i biologi	Studentene bruker, eller bruker ikke, strategier kjent fra matematikk og CT når de jobber med biologi

Resultatene som presenteres vil sannsynligvis ikke gjenspeile alle strategiene som studentene benytter seg av når de jobber med programmering, da disse funnene kun baserer seg på studentenes arbeid med én oppgave. Det kan tenkes at andre strategier er mer relevante i andre oppgaver. For eksempel handler *Algoritmer* om design, parallellisme, effektivitet og automatikk (Shute et al., 2017). I min tematiske analyse har jeg ikke funnet resultater knyttet til parallellisme, og jeg har derfor valgt å ikke inkludere denne strategien videre i min analyse. Det samme gjelder den delen av problemløsning der det konstrueres en løsning basert på løsningen av hvert enkelt delproblem. I og med at oppgaven besto av flere deloppgaver som hjalp studentene med denne delen, gir ikke analysen tilstrekkelig med funn til at denne strategien kan presenteres som en del av resultatene.

Videre vil funnene som presenteres i delkapittelet *studentenes tanker om arbeid med programmering og biologi* kun basere seg på intervjudata fra min undersøkelse. Disse funnene vil dermed kun indikere hvilke tanker disse studentene har om arbeid i to av de tre emnene de møter i første semester, og gir ikke et fullstendig bilde av studentenes arbeid med fagstoffet. Jeg har valgt å inkludere disse funnene som en del av mine resultater fordi de gir en vinkling på spørsmålet om hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier, som er interessant for diskusjonsdelen, og som favner bredere enn problemløsningsstrategier knyttet til programmering.

4.2 Studentenes bruk av problemløsningsstrategier

Dette delkapittelet skal svare på forskningsspørsmålet

Hvilke strategier kjent fra matematikk og CT bruker studentene når de programmerer?

Delkapittelet består av to avsnitt. Det første avsnittet, *Før programmeringen starter*, tar for seg hvilke strategier studentene bruker når de skal planlegge hvordan oppgaven skal løses. Det andre avsnittet, *Underveis i programmeringen*, tar for seg hvilke strategier studentene bruker når de er i gang med programmeringen og skal ferdigstille en løsning.

4.2.1 Før programmeringen starter

Når studentene skal begynne på en oppgave starter de ganske likt. De stater med å skaffe seg en oversikt over hva oppgaven spør om, og hvilken informasjon de har tilgjengelig. Dette er et tydelig tegn på at de benytter seg av strategien «datainnsamling og analyse», som er en del av abstraksjonen. Nedenfor vises et utdrag som er kodet med «henter ut informasjon».

OG2:

«F: Okei, så hva er det vi har informasjon om her

G: Vi vet at hvert fall, altså hvert fall en av, nei.

E: Skal jeg bare skrive ned det vi vet sånn punktvis eller?

F: Ja.»

Studentene henter særlig ut informasjon om hvilke variabler de har fått oppgitt, og noterer som oftest disse ned. Hvordan de velger å notere ned informasjonen varierer. De fleste noterer dette direkte inn i programmet, mens andre velger å notere variablene på papir, som student «C» i sitatet under.

IG1:

«C: jeg må ta ut, sånn, når det er sånn lange tekster, så må jeg ofte skrive ned relatert, eller viktig informasjon ned på papir, før jeg gjør oppgaven.»

Etter at studentene har hentet ut relevant informasjon og skaffet seg en oversikt, begynner de på oppgaven. Et av de første stegene er å bryte ned problemet og forstå hva oppgaven spør om. I gruppe 3 benyttet de seg av strategien *problemnedbryting* i deloppgave e) ved å diskutere hva «counting step» betyr, som vist i dette utdraget.

OG3:

«H: Okei, counting step. Hva vil det si?

I: At du bare legger til en tellevariabel for affected og unaffected.

J: Vi skal liksom skrive hvor mange ganger fikk en affected.»

Som beskrevet i avsnitt 3.2.1, hadde denne oppgaven en tydelig oppdeling og struktur, der problemet i stor grad var delt opp i mindre biter grunnet oppgavens formulering. Derfor finnes det få sitater der det er helt tydelig at studentene aktivt bryter ned problemet. Dette gis det også utrykk for av student «D».

IG1:

«D: […] du fikk jo mye litt gratis da. Du fikk jo spørsmålet uten at [pause]. *Du slapp å tenke ut hva du måtte finne ut selv.»*

I utdragene som er presentert til nå vises problemnedbryting og abstraksjon som separate strategier når studentene begynner på en oppgave. Når de løser programmeringsoppgaver, benytter de seg som oftest av flere strategier parallelt, som vist i sitatet under. Her bruker student «D» abstraksjon, problemnedbryting og algoritmisk tenkning.

IG1:

«D: Jeg bruker å sette, hvis jeg skal lage en kode [programstruktur] *da, eller et program, så tenker jeg litt på, først sånn okei, er det en while-løkke, eller hva er det egentlig de vil at jeg skal bruke her? Og så setter jeg opp det jeg vet at jeg skal ha med. Jeg vet liksom hvor mange år for eksempel jeg skal ha, og da vet jeg liksom hvordan jeg skal plassere det og sånn. Også er det kanskje en, noen tomme plasser der jeg lurer litt på hva jeg skal, hva jeg skal putte inn her. Men når jeg har satt opp resten, så er det mye lettere å putte inn det jeg mangler.»*

Algoritmisk tenkning er tydelig hos flere av studentene ved de ser for seg mulige løsninger på problemet allerede når de leser oppgaveteksten. Dette illustreres godt i de to følgende sitatene.

IG3:

«I: […] tar først på en måte å setter opp variablene som man får fra teksten, og så er det, også gjør jeg ganske fast at jeg på en måte ser, hva er det jeg vil ha til slutt? Også på en måte, hva er sluttproduktet, og så på en måte bygge meg oppover derfra da. At jeg jobber, løser oppgaven baklengs hvis det gir mening? At jeg ser, for å få dette resultatet, så trenger jeg dette. Og for dette så trenger jeg det. Også går jeg tilbake på en måte derfra til variablene. […] Jeg skriver siste greien som kode [program]*, og derfra tar jeg det bakover liksom.»*

IG4:

«K: Første jeg gjør vil jeg si er å skaffe oversikt over hva slags variabler det er jeg har å jobbe med. Og hva slags, hva slags resultater jeg skal få på slutten. Hvis jeg får en oppgave som begynner å bare kaste tall i ansiktet mitt, så fører jeg det inn først at en variabel skal være dette tallet som jeg har fått. Så etter jeg har fått alt av antall sekunder og konsentrasjoner og alt mulig rart oversiktlig, og jeg vet hva det er jeg ser etter, så er det jo greit å bare finne den logiske retningen hvor alt sammen går. Om jeg leter etter et produkt av to forskjellige funksjoner for eksempel, så kan jeg bare dra tilbake og se: ja, de to funksjonene de har jeg. Så da kan jeg jobbe meg videre derfra. Så fortsetter jeg videre sånn nedover til jeg oppnår det resultatet jeg ser etter.»

Ut ifra mine funn ser det ut til at studentene er bevisste på hvordan de skal gå frem for å løse en oppgave. De har her vist til tre konkrete strategier de benytter seg av *før* de starter med programmeringen. De bryter ned problemet, henter ut informasjon og ser for seg en mulig løsning på problemet. Noen av studentene begynner i tillegg å se for seg hvordan de skal designe algoritmer i denne fasen. Hvordan studentene går videre frem for å løse oppgaver i programmering tar jeg for meg i det neste avsnittet.

4.2.2 Underveis i oppgaven

Da studentene skulle begynne å løse oppgaven, planla de først hvilke steg de måtte gjøre for å få resultatet de hadde sett for seg. Utdraget under viser hvordan gruppe 1 gikk frem for å finne hvilke programstrukturer de skulle bruke i oppgave c), og er et eksempel på et utdrag kodet med «designer algoritmer».

OG1:

«B: Okei, så hvis vi skal skrive fem og det skal være random, så må vi skrive en while-løkke da?

A: Ja.

D: Vi får til noen om choice eller noe?

C: Ja.

B: Da kan vi jo sette opp en choice-liste med enten homozygot for dominant, heterozygot eller homozygot for recessiv

D: Også få den til å hente random derifra.»

Underveis effektiviserer gruppene programmeringen sin ved å gjøre flere operasjoner i samme steg. To eksempler fra gruppe 2 viser hvordan de effektiviserer ved at de 1) planlegger å printe fenotypen samtidig med genotypen, og 2) lager en løkke som kjører programmet for dem når se skal sammenlikne outputet programmet gir i oppgave d) og f). Det at studentene lager en løkke som kjører programmet for dem når de skal gjøre samme operasjon flere ganger er også et eksempel på automatikk. Løkker ble i stor grad brukt av alle gruppene, og de andre gruppene hadde liknende løsninger som gruppe 2 i sine programmer.

OG2:

«E: Okei, så hvis vi appender de to greiene til barn nå.

G: Hvis du appender de, så kan du printe barn etterpå, kan du ikke det?

F: Ja.

s5: Ja, det går an. Okei, så barn […]. Også skal vi gjøre det samme med gen to etterpå, ikke sant?

F: Ja, eller [tenkepause].

E: Kan vi gjøre det i samme slengen?

F: Eller, det vi kan gjøre er bare å plusse dem sammen, så får vi en [blir avbrutt av student «G»].

G: Men da får du jo en streng, har du lys til [tenkepause].

F: Da får vi, ja men da får vi en streng for ett barn.»

OG2:

«E: Okei, vi kan godt kjøre den en gang til vi.

F: Ja, eller så kan vi ta en for [tenkepause]. *Vi kan pakke det som et program, også kjøre det som en for-løkke, hvis du vil det?*

G: Ja, vi kan det.»

Selv om gruppene viser evner til å slå sammen steg gjøres ikke dette i alle stegene der det er mulig. Dette kan skyldes at de har ulike erfaringer med å tenke algoritmisk, ikke leser oppgaven nøye nok eller at de ikke er trygge på programstrukturene de skal bruke. Dette illustreres godt i utdraget under, der student «K» føler det blir mer komplisert å slå sammen flere steg.

OG4:

«K: […] må vel lage ny while-løkke for det.

L: Nei, vi kan ha det inni samme, kan vi ikke det?

K: Inni den while-løkken vi allerede har der?

M: Men det vil jo påvirke [tenkepause].

K: Ja, vil ikke ha den.

M: Nei.

K: Eller, vi kan. Men det, jeg føler at det blir komplisert.»

Sammenslåing av steg og automatikk krever at studentene er på et høyt abstraksjonsnivå. Ettersom de fleste av studentene kun hadde programmert i to måneder på tidspunktet der undersøkelsen ble gjennomført, er det naturlig at de ikke mestrer dette fullt ut helt enda. En av studentene med noe tidligere programmeringserfaring viser til gjengjeld et svært høyt nivå av abstraksjon og gode evner til å designe algoritmer, som vises i samtalen mellom oss i dette utdraget.

IG3:

«I: […] Ofte så jobber jeg for å gjøre koden [programmet] *min så lett forklarlig som mulig. For jeg vet at som regel så kommer de på gruppen til å spørre meg om hjelp på en måte. Så hvis jeg da gjør det veldig avansert, eller gjør det på det som jeg føler der den letteste måten, men på en måte er ting som vi ikke har lært i kurset enda eller om. Ja, som bruker ting som ikke er lært, eller ikke har blitt, eller som er vanskelig å forklare. Selv om jeg syns det er*

logisk eller lettere å bruke det, så prøver jeg på en måte å unngå å bruke de tingene for jeg vet at sannsynligvis så må jeg forklare hva jeg har gjort senere. Så da egentlig ofte også, at jeg lager på en måte to oppgaver. En først, på en måte for meg selv som bare gjør oppgaven slik at jeg kan det, også senere en litt mer enkelt forklarlig oppgave.

[…]

S: Men hvorfor vil du ikke bare gå rett på den lett forklarlige versjonen?

I: Fordi at det er ikke alltid det jeg syntes virker mest logisk. Noen ganger så er det lettere å, lettere å gå rett på. Istedenfor å lage masse forskjellige variabler som forklarer ting da. Jeg vet ikke. Bare samle masse linjer opp i en linje er, gir mer mening syns jeg.»

En sentral komponent i abstraksjon er evnen til å forenkle problemet ved hjelp av strategien *modellering*, som er en komponent i abstraksjon. Ut ifra sitatet under, ser også modellering ut til å være en strategi student «I» benytter seg av.

IG3:

«I: […] men jeg bruker likevel matte noen ganger. […] Når stabiliserer grafen seg, eller nå, hva er bæregrensen eller. Så har kanskje brukt bare mye matte for å regne det ut. Både modellerings, det vi har lært av modellering i kurset her på forelesningene, men også bare. Ja, det jeg tenker som logisk matte da på en måte.»

Modellering blir også brukt av de andre studentene. I denne oppgaven brukte alle gruppene krysningsskjema, en vanlig modell kjent fra genetikken. Denne ble riktignok laget på papir, men de kognitive prosessene bak er de samme. Dette tyder på at modellering er kjent for studentene fra andre fagområder enn matematikk og programmering. At alle gruppene valgte å bruke krysningsskjema tyder også på at deler av oppgaven liknet på noe de har løst tidligere. Bruk av strategien *gjenkjenning av mønstre* ble også gjort i andre deler av oppgaven, som vist i utdraget under. Her refererer de til en oppgave de hadde løst i gruppetimen tidligere samme dag.

OG3:

«I: Men her kan vi faktisk regne ut sannsynligheten da. For å få to små [to recessive alleler].

H: Noe ala det vi gjorde i sta? Med liksom, det skulle i funksjonen, tok og delte på 1000 gange med 100. Tror du vi kunne gjort, gjort det her?

I: Ja, det kan vi gjøre der.»

En annen form for mønstergjenkjenning er å gjenkjenne hvilke situasjoner det egner seg å bruke hver enkelt programstruktur. Sitatet under viser hvordan student «F» skiller mellom

situasjoner der det er hensiktsmessig å bruke *for*-løkker, og situasjoner der det er hensiktsmessig å bruke *while*-løkker.

IG2:

«F: Fra mitt synspunkt, så er det det at det på sånn, rent programmerbart eller effektivitetsmessig, så er for-løkker mer effektive enn while-løkker. Men måten jeg vanligvis skiller mellom det er at det hvis jeg vet at den, når jeg går inn i løkka, eksakt antall hvor mange ganger jeg skal gjøre noe, så er for-løkker det beste. Hvis det er sånn at jeg endrer noe i en løkke, som da skal avgjøre når den skal stoppe, da er åpenbart en while-løkke det beste. Det kommer an på.»

Når studentene gjenkjenner et mønster i oppgaven ser det ut til at de gjør en av to ting. Enten så vet de hvilke programstrukturer de skal bruke, og designer algoritmer for videre programmering, eller så kjenner de igjen deler av oppgaven fra et eksempelprogram. I sitatet under forteller student «K» hvordan han tenker tilbake på tidligere oppgaver dersom han møter på et problem i programmeringen sin, og er et eksempel på et sitat kodet med «bruker eksempelprogram som referanse».

IG4:

«K: Med mindre jeg støter på et problem jeg ikke klarer å løse, så bare gønner jeg på, skriver i vei. Alle variabler er nede, og da er det egentlig bare å fylle inn. Så hvis jeg støter på et problem, så må jeg så klart fikse det på en eller annen måte. Da er det enten å tenke tilbake på om jeg har gjort en liknende oppgave et annet sted, en annen gang. Tenke, når var det vi hadde om dette her i undervisning. Sjekke tilbake på notatene som jeg gjorde i timen, disse lange fire timers øktene våre. Hva det er jeg har skrevet da. Se på koden [programmet] *jeg skrev på den tiden og tenke er det noe av dette her jeg kan bruke? Kanskje ikke? Men jeg kan kanskje lage noe som likner, som kunne fungert? Også løser jeg det derfra.»*

Sitatet viser at studentene benytter seg av eksempelprogrammer dersom de støter på et problem de ikke klarer å løse. Som beskrevet i delkapittel 2.4, er det to måter å bruke eksempelprogrammer; som referanse eller som templat. Da studentene skulle løse oppgaven var det kun gruppe 3 som brukte eksempelprogrammer.

IG3:

«H: Jeg tok litt utgangspunkt i, fra kompendiet da. Igjen. Men ja, som jeg sa i sta, vi måtte jo modifisere litt.

I: ja. Eller, jeg så for meg liksom det her, også gjorde det i hodet. Så når vi gikk igjennom kompendiet, så så jeg etter noe som liknet det her, som kunne være lettere å kopiere over da. Fordi det blir litt vanskelig hvis man, eller, jeg følte at jeg hadde det i hodet også, men det var vanskelig å diktere det ut. Så når man så noe som liknet veldig, så er det lettere å kopiere det over bare.

[…]

H: Det er egentlig fra kompendiet. Så jeg bladde jo opp og ned [i kompendiet]*, også så han* [student «I»] *noe som han mente at vi burde bruke da. Også la vi til if-else test i det. Så den var ganske fritt lagd holdt jeg på å si.»*

For å løse oppgaven fant studentene et program i kompendiet som liknet på oppgaven, og kopierte over denne. Dermed brukte de eksempelprogrammet som templat. I tillegg la de til en *if-else* test. Gruppe 3 bærer preg av at studentene har ulike erfaringer med programmering, der student «I» ser ut til å ha en god forståelse, og programmerer stort sett selvstendig. Student «H» er derimot svært avhengig av eksempelprogrammer, hvilket kommer tydelig frem i sitatet under.

OG3:

«H: Hvordan klarte du [student «I»] *å komme helt til bunn med noe så enkelt, uten å bla sånn* [i kompendiet]*?»*

Studentene på de andre gruppene benytter seg også tidvis av eksempelprogrammer, men jeg får inntrykk av at de i stor grad benytter disse som referanse fremfor templat. Student «A» forklarer hvordan kompendiet er til stor hjelp.

IG1:

«A: Så det gjør det da at du trenger ikke å komme opp med det på egenhånd, på en måte det kan føles litt sånn innimellom når man sitter og skal løse en oppgave. Fordi du har lært hvordan man lager en while-løkke, men ikke helt hvordan du skal sette opp for en spesifikk biologioppgave da. Men det gjør på en måte kompendiet for deg. Den viser deg hvorfor og hvordan du gjør det.»

I tillegg til strategiene som er presentert til nå, benytter studentene seg av feilsøkingsstrategier for å finne og rette opp i feil. Studentene sier selv at de kjører programmet ofte for å raskt finne ut av om det virker som det skal, eller om det inneholder feil. Utdraget under er et godt eksempel på studentenes feilsøkingsrutiner, og illustrer hvordan de kjører programmet for hver nye programmeringsbit for å unngå en opphopning av feil.

IG4:

«L: Jeg vet ikke, jeg liker å bare kjøre koden [programmet] *min hele tiden, hver gang jeg gjør noe nytt. Så sjekke liksom, okay det funker fortsatt. Og så fortsette.*

S: Ja, hvis det blir feil da?

L: Ja da må jeg jo prøve å. Åja i, i jupyterhub [grensesnittet studentene jobber i] *her, så står det jo til og med hvor, hvor feilen er da. Og for det meste er det jo veldig greit å rette opp igjen.»*

Som beskrevet i delkapittel 2.4, er det i hovedsak to typer feil studentene møter når de programmerer; syntaksfeil og logiske feil. Det at studentene programmerer med biologiske problemstillinger ser ut til å gi dem en god pekepinn på hva svaret skal bli, hvilket hjelper dem når de skal designe algoritmer, og som i tillegg bidrar til å rette opp logiske feil. De to utdragene under viser hvordan studentene utnytter denne fordelen, og er eksempler på studentenes bruk av feilsøkingsstrategier.

IG1:

«D: Ja, du vet jo hvilket, fordelen her er at du hvilket svar du skal få. Også må du bare prøve å få det svaret du vil ha.»

IG2:

«E: Nei, bare at vi må sjekke biologien helt til slutt. Det er kjempeviktig. Får du at svaret er minus en pingvin eller noe sånt, så er det feil. Får du, ja.

S: Ja, jeg skjønner hva du mener.

E: Så selv om svaret rent matematisk er fint å se på, så er det ikke alltid at det er riktig biologisk.»

Da studentene løste oppgaven opplevde de få feil i programmene sine, og feilene de opplevde var i hovedsak syntaksfeil. I utdraget under ser det tilsynelatende ut til at studentene møter på en logisk feil, og de oppdager selv at svaret ikke gir mening ut ifra hva de forventet som resultat.

OG1:

«C: Jeg skjønner ikke, hvorfor adder det ikke opp til null prosent?

D: Nei, det gikk ikke helt opp.

A: Nei, for du får jo adda opp [tenkepause].

C: Det gir jo ikke mening.

D: Men tanken er jo at [blir avbrutt av student «B»].

B: Å, det gjør ikke det.

D: Egentlig det at det skal være så stor sjanse for at en blir syk.

B: Ah, fack.

D: Og så stor sjanse for at en blir frisk.

B: Det gjør den jo selvfølgelig ikke, det var jo, det er jo [blir avbrutt av student «D»].

D: Om du får tusen unger eller om du får en unge, det har ikke noe å si, men sjansen for at den blir syk eller frisk den er [tenkepause].*»*

```
In [19]: from pylab import *
         sykdom = ["RR", "Rr","Rr","rr"]
         i = 0
         a = 0
         b = 0
         while i <= 4:
             barn = choice(sykdom)
             if barn == "RR":
                 feno = "R, frisk"
                 b = b + 1
             elif barn == "Rr":
                 feno = "R, frisk"
                 b = b + 1
             else:
                 feno = "r, syk"
                 a = a + 1
             i=i+1
         print("sannsynlighet for å få et sykt", a/(i+1),"sannsynlighet for å få et friskt" b/(i+1))

         0.16666666666666666 0.6666666666666666
```

Figur 3 – Studentenes program, som ga et ugyldig svar på problemet. Feilen er markert i de blå boksene.

Den logiske feilen i utdraget over var egentlig en syntaksfeil, der studentene hadde lagt til en ekstra teller i programmet sitt. Denne måtte fjernes for å få forventet resultat. Studentene identifisere feilen, og rettet opp i denne som vist i utdraget under.

OG1:

«C: Hva mener, hva gjør du?

B: Jeg bare fjerner pluss en.

C: Åja, okay.»

```
In [22]: from pylab import *
         sykdom = ["RR", "Rr","Rr","rr"]
         i = 0
         a = 0
         b = 0
         while i <= 4:
             barn = choice(sykdom)
             if barn == "RR":
                 feno = "R, frisk"
                 b = b + 1
             elif barn == "Rr":
                 feno = "R, frisk"
                 b = b + 1
             else:
                 feno = "r, syk"
                 a = a + 1
             i=i+1
         print("sannsynlighet for å få et sykt", a/(i),"sannsynlighet for å få et friskt", b/(i))

         sannsynlighet for å få et sykt 0.4 sannsynlighet for å få et friskt 0.6
```

Figur 4 – Studentenes program, som her gir et gyldig svar på problemet. Feilen som ble rettet opp er markert med blå bokser.

Studentene sier selv at når de får feil i programmene sine, så skyldes det i stor grad syntaksfeil. Studentene i gruppe 2 syntes det er vanskelig å være presise hver gang, og opplever dette som en av hovedutfordringene ved programmering.

IG2:

«G: […] Jeg vil kanskje si de meste feilene jeg gjør er jo sånn, glemmer et skritt i while-løkken eller, veldig mange sånne små, små ting da. Du glemmer, du tar et for stort tall når du skal ta ut ting av en liste, eller at du [blir avbrutt av student «F»].

F: Skriver en variabel med stor forbokstav

G: Ja, mye sånt da.

F: Glemmer en S på slutten.

G: Ja ja ja, og da vil jo datamaskinen tolke det helt som bare feil, ikke sant. Og da, man må være ganske presis. På hva man mener og sånt.

S: Så det vanskeligste er å være presis? Er det det du sier nå?

G: Ja. Jeg vil kanskje si, det er jo kanskje det vanskeligste, bare liksom ha riktig hver gang da, på en måte. Være presis hver gang. Det er alltid noen slurvefeil inni der.»

Analysen i dette delkapittelet viser at studentene bruker en rekke strategier kjent fra matematikk og CT når de programmerer. De benytter seg av strategien *problemnedbryting* når de gjør problemet om til mer håndterlige biter. Når studentene henter ut relevant informasjon og gjenkjenner mønstre benytter de seg av strategien *abstraksjon*, og ved å identifisere hva som må til for å komme fra et sett med variabler til et ferdig resultat, bruker de strategier fra *algoritmisk tenkning*. De viser ulik grad av effektivisering og automatikk, men strategiene er til stede. De benytter seg tidvis av *eksempelprogrammer*, og driver med aktiv *feilsøking*. Syntaksfeil ser ut til å være den vanligste feilen, og for noen ser det ut til at den største utfordringen med programmering er å unngå denne typen feil. Funn knyttet til studentenes utfordringer med programmering vil bli presentert i det neste delkapittelet.

4.3 Studentenes utfordringer knyttet til bruk av strategier

Dette delkapittelet skal svare på forskningsspørsmålet

Hvilke utfordringer knyttet til bruk av problemløsningsstrategier møter studentene når de programmerer?

Alle gruppene klarte å løse oppgaven uten store problemer, og opplevde dermed denne oppgaven som mindre utfordrende enn andre oppgaver de har løst i BIOS1100. Dette kan

skyldes at oppgaven hadde en tydelig oppdeling, eller at oppgaven ble gitt på et tidspunkt i emnet der studentene hadde fått jobbet tilstrekkelig med programstrukturene de trengte for å løse oppgaven. Student «B» mente at dersom han måtte peke ut noe i oppgaven som var utfordrende, måtte det være planleggingsfasen, som vist i sitatet under.

IG1:

«B: Det var bare den første planleggingsfasen, hvordan man skal utforme den da. Men det var jo ikke sånn kjempeutfordrende sånn sett, mener jeg da. Fordi vi fikk for det meste alt hamra ned, så når vi først skrev den, så ble det bare småendringer. Også, ja, det gikk litt av seg selv egentlig. I forhold til mange andre av de andre oppgavene, som er vanskelige. Så står man gjerne bom fast på den første planleggingsfasen i hvordan man skal sette den opp. Her var det ikke så mye av det, syns jeg.»

Planleggingsfasen, der studentene skal bryte ned problemet og bestemme hvilke steg de skal gjennomføre for å løse oppgaven, kan ifølge denne studenten by på utfordringer. De andre studentene på gruppe 1 mente i tillegg at oppgave c) hadde vært utfordrende dersom de måtte løst den alene, hvilket kommer til uttrykk i utdraget under.

IG1:

«C: Hadde jeg gjort den for meg selv, så hadde jeg syns at c) hadde vært litt tricky kanskje.

S: Ja, når du skal legge inn while-løkke?

C: Mhm.

S: Og så med if etterpå.

[Student «A», «C» og «D» gir uttrykk for at de er enige i utsagnet mitt ved å si «Mhhm»].

[…]

S: Hvorfor tenker dere at det er akkurat den biten som blir mest vanskelig? Er det fordi det er ny kode [programstruktur]*, eller er det noen andre årsaker?*

C: Ja, det er jo å sette sammen en ny kode [et nytt program] *ja. De andre oppgavene var litt sånn bruk samme kode* [program]*, skift variabler eller tall, et eller annet. Og det gir jo mening. Det andre var liksom, biologien var jo ganske grei syns jeg.»*

Studentene syntes det er utfordrende med nye programstrukturer og store programmer når de skal programmere. Det kan være vanskelig å se for seg hva som skal inn i løkkene og når de ulike løkkene skal brukes, hvilket kommer frem i sitatet nedenfor. Utfordringer med å bestemme hva som skal inn i løkkene, og hvilke programstrukturer som skal brukes, er tegn på at studenten har utfordringer med å designe algoritmer.

IG2:

«E: Jeg kan syns det er, kan være vanskelig å egentlig vite litt hvor jeg skal putte ting inn, og hvilke formler jeg eventuelt skal bruke. Sånn, hva skal inn i while-løkken, hva skal stå utenfor while-løkken. Og når, ja, når man skal bruke for-loop og while-løkke, og litt sånne ting og. Kan jeg syns er litt vanskelig.»

Videre er det utfordrende for dem når løkker og funksjoner går i hverandre, som student «M» forteller i sitatet under. Dette ble det også gitt uttrykk for i et av sitatene knyttet til effektivisering i forrige delkapittel. Der ønsket studenten å holde løkkene adskilt, da det opplevdes komplisert for studenten å legge løkkene inn i hverandre.

IG4:

«M: Det [mest utfordrende] *er vel når forskjellige system går i hverandre. For eksempel når du har en løkke i en funksjon som igjen skal i en løkke. […] Det, det er ganske vanskelig.»*

Selv om studentene sammen klarte å løse denne oppgaven, og benyttet seg av flere problemløsningsstrategier kjent fra CT og matematikk for å løse den, ser det ut til at de møter en del utfordringer når de løser andre oppgaver i emnet. Dette kan skyldes at de har andre arbeidsvaner når de programmerer enn det som vises her, eller at oppgavene ikke har den samme tydelige oppdelingen. En oppgave som ikke har en tydelig struktur kan være mer utfordrende for studentene, fordi de selv blir nødt til å bryte ned problemet for å forstå hva oppgaven etterspør. Som student «H» forteller i sitatet under, kan det være vanskelig å forstå hvordan man skal begynne på en oppgave. Dette er et godt eksempel på at studentene syns det kan være vanskelig å bryte ned et problem.

IG3:

«H: Noen ganger så syns jeg det kan være vanskelig […] å vite hva oppgaven egentlig ber om. Eller, ber om. Eller, jeg skjønner hva den ber om, men hva jeg skal gjøre for å svare på det. Det er ikke alltid, men noen tilfeller så har jeg syntes det har vært litt vanskelig å vite hvordan jeg skal svare på det da. Ja, så er det jo å vite, kanskje spesielt ved hva er det jeg skal sette i løkka mi, siden du har jo sånn while i mindre [i<]*, og på en måte hva jeg skal ha der. Ja det blir mer avansert når du skal legge inn if-else tester inni der da. Egentlig hvor jeg skal plassere de ulike.»*

Sitatet over viser at det i stor grad er vanskelig å planlegge hvordan oppgaven skal løses. Det ser ut til at hovedutfordringene for studentene er å bryte ned problemet, og deretter designe algoritmer, altså identifisere hvilke steg de må gjøre for å løse oppgaven. I tillegg er abstraksjon av oppgavene og programstrukturene som inngår krevende. Studentene kan ha utfordringer med å finne ut hva oppgaven spør om, som kommer frem i sitatet over, eller utfordringer med å forstå hva den enkelte programstrukturen gjør. Nedenfor vil jeg vise et

eksempel på utfordringer med abstraksjon av programstrukturer som studentene kan møte på. Utdraget er knyttet til oppgaven studentene jobbet med, men er kun representativt for student «A». Jeg har likevel valgt å inkludere sitatet, da det illustrerer godt hvilke utfordringer knyttet til abstraksjon studentene generelt kan møte på.

OG1:

«A: Nei jeg bare skjønner ikke, altså må man ikke legge inn hva sannsynligheten er for å få et friskt eller et sykt barn for hver gang? Altså det er jo sånn man regner sannsynligheten for å få gutt eller jente da, siden det er 50/50. Og legger man ikke inn det [blir avbrutt av student «B»].

B: Men ja, ikke sant, den sjansen er bygd inn i choices, siden i, på, du ser sykdom på toppen der?

A: Ja.

B: Da har vi fire forskjellige variabler, ikke sant?

A: Mhm.

B: Og choices tar da og velger en av dem.

A: Ja, så skjønner den derre choice-funksjonen det at det er en fjerdedel på hver av de?

B: Ja. Jaja, fordi det er jo en fjerdedel, det er helt random. Altså det er like stor sjanse for å få hver av dem.

A: Jaja, det er random.

B: Og det er derfor det er en fjerdedels sjanse, det er fire av dem.

A: Okei, så choice-funksjonen skjønner det at [blir avbrutt av student «B»].

B: Ja.

A: Hvis du har fire variabler så er det en fjerdedel?

B: Ja.»

Utdraget viser at student «A» i forkant av denne oppgaven ikke har forstått hvordan *choice*-funksjonen fungerer, annet enn at den velger et tilfeldig element. Dette viser at studenten hadde utfordringer med abstraksjonen mellom nivået *hva gjør funksjonen?* og *hvordan virker funksjonen?* Det er ikke usannsynlig at studentene på BIOS1100 møter på liknende utfordringer i emnet som illustrert i dette sitatet, da de forteller at de syntes det er utfordrende med nye programstrukturer, store programmer og anvendelse av de ulike løkkene.

4.4 Studentenes holdninger til krav om matematikk R2

I dette delkapittelet vil jeg presentere funn knyttet til forskningsspørsmålet

Hvordan vurderer studentene relevansen av kompetansen de fikk fra R2 inn i BIOS1100?

Resultatene i dette delkapittelet hører utelukkende til teamet «R2-kravet». Studentene hadde flere meninger knyttet til det nye opptakskravet. Av noen trekkes det frem at matematikken de møter i BIOS1100 i seg selv kan være krevende, som vist i sitatet under.

IG2:

«F: Men spesielt når det kommer til R2, så tror jeg egentlig det er. Jeg tror egentlig det er ganske bra, fordi mye, eller det lille matten vi har hatt hittil er, det kan være ganske vanskelig å forstå konseptene hvis du ikke har […] en grei nok forståelse av vektorer og slike ting fra før av da.»

Studentene på gruppe 1 legger til at forkunnskaper i matematikk er fordelaktig når man skal lære programmering, fordi programmeringen i seg selv består av mye matematikk.

IG1:

«A: Jeg tror det er bra [krav om R2], *fordi mye er jo matte, og spesielt når vi har jobba med differensiallikninger eller hva de heter […]*

B: Ja, differential equations, ja.

A: Ja. Så er jo det matte, og jeg vet ikke, jeg husker ikke helt hvorvidt det var en del av R1 eller ei. Men i hvert fall hvis du har hatt R2, så tror jeg du kommer, at du har lettere for å forstå det eller, ja, eller at du hvert fall har litt bakgrunn for å forstå det.»

Ut ifra det studentene forteller i utdragene over tolker jeg det slik at de har hatt behov for de matematiske kunnskapene de har fått gjennom R2, og at de har brukt disse når de har programmert. Samtidig forteller noen av studentene at de syns matematikkravet er unødvendig, særlig fordi de opplever at det er lite matematikk i BIOS1100. Dette illustreres godt av det følgende sitatet, og er et eksempel på et sitat kodet med «negativ til R2».

IG3:

«I: Jeg er ikke helt sikker om det jeg, jeg syntes jo at, at det gir mening at det kreves på så mange studier. For på mange studier så er på en måte matte, fokusere på matte gir jo mening. Men akkurat i det studiet her, så vet jeg ikke hvor viktig det er. Matten vi har lært her har jo på en måte, det har ikke trengt, vi har ikke trengt noe kunnskap fra R2 spesielt for å lære den matten vi har lært i det kurset her. Vi har gått igjennom matten ganske raskt, så man må på en måte ha en forståelse for tall. Men R2 i seg selv har ikke noe særlig å si for det kurset her, slik jeg har forstått det. Eller slik jeg, føler jeg i hvert fall.»

Jeg tolker dette som at studenten ikke anser kompetansen fra R2 som en nødvendig forkunnskap for BIOS1100, eller for studieprogrammet i sin helhet. Dette kan henge sammen med forestillingen om at biologi er en realfaglig disiplin som er unntatt matematikken, som jeg presenterte kort i innledningen av denne studien. Denne forestillingen er synlig i flere av gruppene, og vises tydelig i sitatet fra student «B».

IG1:

«B: Jeg tror også at, hvert fall de fleste jeg snakker med på linja [studiet] *sier jo at det er en grunn, en av de grunnene til at de tar biologi er at der man kommer lengst vekk fra matte også. Det er i hvert fall litt sånn som jeg føler også, siden jeg var jo ikke noe fan av matte. Og, og, hvert fall, jeg liker fysikk, men jeg orker ikke matte delen av det. Og biologi er jo egentlig, langt vekk jeg kan komme fra det. Og, så det blir jo litt sånn. Hvorfor ha det da, når ikke biologi er så viktig for.. eller, matte er så viktig for biologi egentlig.»*

I dette sitatet sier studenten eksplisitt at matematikk ikke er en viktig del av biologien. Jeg antar at dette har en sammenheng med at elevene i liten grad eksponeres for matematiske modeller og beregninger i biologifaget på videregående skole (Utdanningsdirektoratet, 2006a), som er den biologien de fleste studentene har kjennskap til før de starter på biovitenskap. Ut ifra sitatet over ser det også ut til at noen studenter velger biovitenskap fremfor andre realfaglige utdanninger for å unngå matematikk.

Utover synspunktene knyttet til matematikkforkunnskaper i BIOS1100 og studiet for øvrig, hadde studentene også tanker rundt andre aspekter ved matematikkompetansen som er viktige for programmering og for studiet. Ifølge studentene på gruppe 1 åpner R2-kravet for at mer avanserte emner kan legges til bachelorprogrammet i biovitenskap, da majoriteten av de realfaglige emnene har krav om R2. I andre grupper hevdes det at R2-kravet er innført for å innsnevre søkermassen, som starten på det neste sitatet illustrerer. Dette er et eksempel på et sitat som er kodet med både «negativ til R2» og «positiv til R2».

IG4:

«K: Det er vel kanskje bare mer for å sette en slags terskel til å finne menneskene som ville gått en, et så logisk fag R2 matematikk er da. […] For hvis du føler at du tar P-matte for eksempel, fordi matten er for vanskelig i R2, eller i T-matte til og med. Så kanskje du ikke burde ta et programmeringsfag til å begynne med. For kanskje det ikke bare, kanskje det ikke går overens med hvordan det er du tenker. Kanskje du vil da slite veldig mye med det, siden alt sammen handler bare om logikk. Så jeg ville ikke si at matten i seg selv er så nødvendig, for vi bruker egentlig veldig lite matte. [..] Men vi bruker jo all den logiske tankegangen som du blir trent opp i fra matten. Jeg vil kanskje si at det er derfor.»

I dette sitater reflekterer studenten over viktigheten av den logiske tankegangen, og dermed strategiene for problemløsning, man læres opp i dersom man har full fordypning i realfagsmatematikk. Som studenten selv sier, så er det ikke matematikkunnskapene i seg selv som er hovedpoenget med opptakskravet, men heller de strategiske ferdighetene man får gjennom å lære logisk tenkning. Dette reflekteres det også over i utdraget fra gruppe 1, og er et eksempel på et utdrag kodet med «positiv til R2».

IG1:

«S: Fint. Jeg vil bare høre, når du sier logisk tenkning, hva tenker du da? Fordi du brukte det begrepet to ganger […].

D: Ja. Ja men det er, det stemmer det. Jeg er opptatt av den derre logiske tenkninga. […] jeg sier jo ofte at «du må bare tenke!». Bruk hodet. Fordi at ja, jeg mener at mange problemer, altså hvis du har mye matte fra før, så har du lært deg til å tenke, i hvert fall jeg har lært det da, til å tenke strategisk og liksom. Du skal løse en oppgave, og få ett svar på en måte. Og det gjør du jo i programmering også. Du må jo en måte skjønne hva de spør om, hente ut spørsmålet, og. Og du må tenke hvordan du skal sette sammen, det er mye logikk i det for eksempel. Skal du ha det og det elementet i løkka, utenfor, hvordan. Hvis ikke du ser for deg det, så må du kjøre den mange ganger for å se hva som skjer. Hvis du klarer å tenke hvordan du skal plassere ting i forhold til hverandre, og se for deg hva som kommer ut, så sparer du mye tid på det. Det er det jeg mener med logisk tenkning. Tror jeg. Strategi, problemløsning. Sånne ting, og det. Hvis du har litt matte, så tror jeg du er bedre, du har bedre forutsetninger for å klare å løse oppgavene med å bruke logikk da.

A: I hvert fall hvis matten du har hatt er problemløsende, eller hva skal jeg si, har fokus på å løse problemer.

D: Og det er jo den realfagsmatten.»

Selv om ikke alle studentene ser på den logiske tenkningen og matematikkravet i samme metaperspektiv som i dette sitatet, viser resultatene i delkapittel 4.2 at studentene i stor grad bruker strategier kjent fra matematikk og CT når de programmerer. Studentene evner å bryte ned problemet, designe algoritmer og håndtere kompleksiteten i problemet ved hjelp av abstraksjon, hvilket er de tre strategiene i CT som er direkte relatert til matematikk. Dette kan tyde på at kravet om kompetansen fra R2 har vært positivt for studentene på bachelorprogrammet i biovitenskap.

4.5 Studentenes tanker om arbeid med programmering og biologi

Dette delkapittelet skal svare på forskningsspørsmålet

Hvilke tanker har studentene om bruk av strategier i arbeid med fagstoff i BIOS1100 og BIOS1110?

Som beskrevet i innlendingen av denne studien er det på mange måter den logiske tankegangen og strategiene for problemløsning som har vært hovedpoenget med R2-kravet. Dette er strategier som er nødvendige ferdigheter i programmering, men som i større og større grad også er nødvendige ferdigheter i biologi. BIOS1110 er et mer tradisjonelt emne i biologi enn BIOS1100, og jeg vil i dette delkapittelet sette studentenes arbeid med de to emnene opp mot hverandre, for å se på hvilke tanker de har om det klassiske biologiemnet BIOS1110, og det beregningsorienterte emnet BIOS1100. Sitatene og utdragene som presenteres i dette delkapittelet er alle kodet med «strategier i biologi», og tilhører temaet *overføring av strategier til biologi.*

Det ble presentert to sitater i forrige delkapittel der studentene ga uttrykk for at matematikk ikke er en viktig del av biologien, selv om dette er en realfaglig disiplin. Disse holdningene kommer også til uttrykk når studentene snakker om hvordan de jobber i emnene BIOS1100 og BIOS1110, som de tar parallelt det første semesteret av studiet. Som student «D» gir uttrykk for i sitatet under, ser det ikke ut til at studentene benytter seg av de samme strategiene for problemløsning når de jobber med Celle- og molekylærbiologi.

IG1:

«D: I Celle- og molekylærbiologi så er det veldig mye lesing. Du må bare lese og skjønne, det er ikke så mye tenking sånn sett. Mens her [BIOS1100] *er det jo, du må jo tenke logisk da […].»*

Det ser ut til at studenten har en forestilling om at måten man lærer seg, og dermed arbeider med det biologifaglige, er ved hjelp av pugging og passiv tilnærming til fagstoffet. Studenten skiller mellom arbeidsmåter i biologi og programmering, og det ser ut til at studenten har en oppfatning om at man i programmering i større grad må tenke og forstå, enn hva man må i biologi. I de andre gruppene gir de også uttrykk for at de to emnene er svært ulike, og at pugging er en mye brukt strategi i BIOS1110, som vises i utdraget under.

IG4:

«L: Nei, det er to veldig forskjellige fag da. Jeg føler at, ja, det andre [Celle- og molekylærbiologi] *er jo egentlig bare puggefag. Mens her* [BIOS1100] *så jobber man jo litt mer med det.*

K: Absolutt.

L: Jeg syns. Sånn sett så syns jeg det her, ja. Det er jo litt artigere sånn sett.

K: Her kan du, kan du finne frem til svaret ved å bare sitte å gå inn ditt eget hode.

L: Ja.

K: Mens i BIOS1110 [Celle- og molekylærbiologi] *så må du ofte gå inn i boken, og ikke inn i deg selv for å finne riktig svar.»*

Det kan se ut til at studentene ikke har overført de strategiske kunnskapene de kjenner fra matematikk, og etter hvert programmering, til det biologifaglige. Studentene ser ut til å arbeide med fagstoffet svært ulikt i de to emnene. I biologi vil de gjerne lese, pugge og bli forelest for, mens de i programmering vil ha mengdetrening og tid til å prosessere. Dette illustreres godt i de to neste sitatene.

IG2:

«G: Ja, jeg syns BIOS1110 er mye lettere å forklare, mens i sånn som programmering [i BIOS1100]*, så er det litt sånn man lærer å gjøre, du må liksom på en måte ha gjort det selv litt da, før du faktisk forstår det. Også kan man jo kanskje si det at sånn som Celle- og molekylærbiologi er veldig mye mer litt sånn puggefag enn kanskje for eksempel programmering, som er da mer, ikke nødvendigvis mer forståelse da men, det er mer prosessering da.»*

IG2:

«E: Ja, jeg trenger i hvert fall å få en oversikt selv over begreper og sånne ting, og liksom få systematisert det litt for meg selv og skrive notater. Mens i BIOS1100 så må jeg få gjort oppgaver i hvert fall. For det hjelper jo ikke å pugge hvordan jeg skal skrive en while-loop [while-løkke]*, jeg må liksom skjønne hvilke ting jeg skal ha med inn i loopen* [løkken]*.»*

Ut ifra det studentene forteller om arbeidet med fagstoff i emnene BIOS1100 og BIOS1110, får jeg et inntrykk av at de bruker færre strategier for problemløsning kjent fra matematikk og CT i biologiemnet enn hva de gjør når de programmerer. Dette kan som nevnt skyldes at studentene har en oppfatning om at biologi er unntatt matematikken, og at de har lite erfaring med bruk av disse strategiene i biologi fra tidligere. Jeg må understreke at dette inntrykket baserer seg på hva studentene selv fortalte under intervjuet, og inkluderer ikke undersøkelser om hvordan studentene faktisk arbeider i BIOS1110.

5 Diskusjon

I dette kapittelet vil jeg først drøfte svarene på mine fire forskningsspørsmål i lys av relevant teori. Deretter vil jeg drøfte betydningen av mine funn, og hvilke implikasjoner de kan få for valg av opptakskrav og undervisning på utdanningen i biovitenskap. Kapittelet avsluttes med en refleksjon over studiens begrensninger, og forslag til videre forskning.

5.1 Studentenes strategier for problemløsning

I dette delkapittelet vil jeg drøfte hvordan resultatene svarer på forskningsspørsmålet

Hvilke strategier kjent fra matematikk og computational thinking bruker studentene når de programmerer?

Resultatene fra min analyse tilsier at studentene benytter seg av strategiene kjent fra matematikk og CT når de programmerer, men at strategiene benyttes i ulik grad. I tillegg brukte studentene feilsøkingsstrategier, og tidvis også eksempelprogrammer. I de neste avsnittene tar jeg for meg de ulike strategiene studentene brukte, og hvordan de ble brukt.

5.1.1 Problemnedbryting og abstraksjon

Når man skal løse et problem med utgangspunkt i strategier fra CT og matematikk er problemnedbryting og abstraksjon første steg (Shute et al., 2017). Som jeg viste i delkapittel 4.2, var det første studentene gjorde da de skulle løse oppgaven å skaffe seg en oversikt over hvilken informasjon de hadde tilgjengelig. Evnen til å hente ut relevant informasjon fra en oppgave er en sentral del av abstraksjon (Shute et al., 2017), og jeg vil argumentere for at studentene i min studie behersket denne strategien. De noterte ned hvilke variabler de har fått oppgitt, skaffet oversikt over hvilken informasjon de manglet og tolket hva oppgaven spurte om. Da de tolket oppgaveteksten viste de tegn til problemnedbryting ved at de diskuterte hva de skulle finne ut av, men som problematisert tidligere, så bar oppgaven studentene jobbet med preg av en tydelig oppdeling. Dette gjorde at studentene i liten grad fikk vist hvordan de går frem for å bryte ned et problem.

For å løse oppgave a), og finne informasjon som var nødvendig for å løse resten av oppgaven, lagde gruppene et krysningsskjema. Dette er en modell som brukes mye i genetikken, og studentene har blitt introdusert for krysningsskjemaer i naturfag på VG1, programfaget biologi og BIOS1110. Å forenkle problemet ved hjelp av modeller er en viktig del av

abstraksjon (Shute et al., 2017), men om studentene bruker denne strategien aktivt når de programmerer er noe usikkert. En av studentene forklarte eksplisitt hvordan han bruker matematikk til å lese grafer og modellere, mens de andre studentene ikke ga uttrykk for at de brukte modellering aktivt i sin problemløsning. I og med at krysningsskjemaer er såpass kjent for studentene, kan det tenkes at de brukte modellen fordi de først og fremst gjenkjente den biologiske tematikken i problemet. Det er derfor diskutabelt hvorvidt modellering er en bevisst strategi hos studentene.

Da studentene jobbet med oppgaven, gjenkjente de en rekke mønstre fra andre oppgaver de hadde jobbet med tidligere. Dette er et tegn på bruk av problemløsningsstrategien mønstergjenkjenning, som innebærer at man gjenkjenner deler av problemet som likner på noe man har løst tidligere, og anvender denne kunnskapen på det nye problemet (Grover & Pea, 2018). Å kjenne igjen situasjoner der de ulike løkkene skal anvendes er et godt eksempel på en mønstergjenkjenning som studentene brukte for å løse oppgaven. Mine funn tyder på at studentene i stor grad gjenkjenner mønstre når de programmerer, og at de bruker denne informasjonen når de løser oppgaven. Ofte fører mønstergjenkjenningen til at studentene vet hva de skal gjøre, som i eksempelet med løkkene, mens i noen tilfeller fører det til at de benytter seg av eksempelprogrammer. Bruk av eksempelprogrammer vil diskuteres i avsnitt 5.1.3.

5.1.2 Algoritmer

Når man designer en algoritme planlegger man et sett med strukturerte instruksjoner som skal gi løsningen på problemet (Shute et al., 2017). Da studentene jobbet med oppgaven begynte de å designe algoritmer, samtidig som de hentet ut informasjon, ved at de så for seg hva sluttproduktet skulle bli. En algoritme vil alltid ha et definert start og sluttpunkt, der sluttpunktet er løsningen på problemet (Anderson, 2016). Da studentene hadde forstått oppgaven og fått en oversikt over hvilket resultat de kunne forvente, begynte de å planlegge hvordan de skulle gå frem for å løse oppgaven. Det er i denne planleggingsfasen de i hovedsak designer algoritmen, og de gjør dette ved å finne hvilke programstrukturer de skal bruke. I denne oppgaven valgte studentene å designe algoritmer som blant annet innebar å bruke programstrukturen *choice* i kombinasjon med ulike løkker.

Ved bruk av algoritmer er det mange måter man kan løse et problem, så lenge start- og sluttpunktet forblir det samme (Anderson, 2016). Dette er særlig relevant i programmering, da det finnes mange måter å lage et program som gir samme resultat. Dette ble synlig da

studentene skulle løse oppgaven jeg ga dem, ved at de hadde ulik grad av effektivisering og bruk av løkker. Jeg vil argumentere for at det i et godt algoritmisk design er høy grad av effektivisering, automatikk og parallell gjennomføring av steg, og at programmet som skrives er kortfattet og oversiktlig. Studentene viste at de evnet å programmere effektivt ved at de de gjorde flere operasjoner i samme steg. De valgte for eksempel å printe fenotypen samtidig som genotypen i oppgave d), fremfor å gjøre dette i to separate operasjoner. De viste også at de evnet å automatisere systemet ved å lage løkker som gjennomførte samme operasjon flere ganger. Både effektiviseringen og automatiseringen gjorde at programmene som ble skrevet ble kortfattet, men gruppene brukte strategiene i ulik grad. Dette kan skyldes at de har ulik trening i å tenke algoritmisk, hvilket er sannsynlig, i og med at flere av studentene ikke hadde tidligere programmeringserfaring.

5.1.3 Feilsøking og bruk av eksempelprogrammer

Under mine intervjuer med studentene kom det frem at de bruker feilsøkingsstrategiene ganske likt. Den vanligste strategien er å kjøre programmet ofte for å finne ut om det virker som det skal og for å identifisere mulige feil. Denne strategien fungerer godt dersom feilen studentene gjør er syntaksfeil, som skyldes feil plassering av symboler (Qian & Lehman, 2017). Dersom studenten møter på syntaksfeil forsøker de å rette den opp, og klarer stort sett dette på egenhånd, ettersom grensesnittet de arbeider i forklarer hvor feilen ligger. Hva studentene gjør når de møter på logiske feil i programmet kommer ikke tydelig frem i min undersøkelse. Dette skyldes først og fremst at studentene tilsynelatende ikke møtte på denne typen feil. Gruppe 1 fikk en logisk feil som skyltes en syntaksfeil i sitt program, og forsto at de hadde en feil fordi svaret ikke ga mening fra et matematisk perspektiv. Ut ifra det studentene sier i intervjuene, er en strategi for å identifisere logiske feil å vurdere svaret ut ifra en biologisk eller matematisk kontekst, men denne strategien vil neppe fange opp alle logiske feil som studentene gjør. Jeg kan derfor ikke si med sikkerhet hvilke andre strategier studentene bruker når de møter på, eller forsøker å identifisere, logiske feil.

Det var kun én av gruppene som benyttet seg av eksempelprogrammer da de løste oppgaven. Gruppen brukte eksempelprogrammet som templat ved at de kopierte over deler av programmet, og la i tillegg inn en *if-else* test. De andre studentene fortalte i intervjuet at de tidvis også benytter seg av eksempelprogrammer, men at de bruker disse som referanse fremfor templat. Det ser dermed ut til at de fleste studentene programmerer relativt selvstendig når de arbeider med oppgaver i BIOS1100.

Håland (2019) fant i sin studie at de vanligste strategiene hos studentene på BIOS1100 var prøv- og feilstrategier og bruk av eksempelprogrammer. Disse strategiene ble også brukt av studentene i min studie, men det ser ut til at de benytter seg av strategiene ulikt. Håland sine studenter benyttet seg av prøv- og feilstrategier som feilsøkingsstrategi og som en kompensasjon for manglende strategier innen algoritmisk tenkning. Prøv- og feilstrategien blir av studentene i min studie i hovedsak brukt som en feilsøkingsstrategi. Bruken av eksempelprogrammer er også ulik blant studentene, der studentene i Hålands studie hadde bruk av eksempelprogrammer som hovedstrategi i sin programmering. Studentene som deltok i min studie ser ut til å programmere mer selvstendig, der de i større grad benytter seg av strategier utover bruken av eksempelprogrammer. Dette kan henge sammen med at studentene i min studie har mer erfaring med bruk av problemløsningsstrategeier, fordi de har hatt matematikk R2 på videregående. Det er også mulig at forskjellen i bruk av eksempelprogrammer skyldes endringer av undervisningen i BIOS1100, eller at det ble gjort små endringer i oppgaven studentene arbeidet med.

5.2 Studentenes utfordringer knyttet til bruk av strategier

Her vil jeg drøfte hvordan resultatene svarer på forskningsspørsmålet

Hvilke utfordringer knyttet til bruk av problemløsningsstrategier møter studentene når de programmerer?

Studentene løste oppgaven uten store problemer, og opplevde ikke denne oppgaven som spesielt utfordrende. I analysen påpekte jeg at dette kan skyldes at oppgaven hadde en tydelig oppdeling, eller at studentene hadde fått tilstrekkelig tid til å arbeide med programstrukturene de trengte for å løse oppgaven. I intervjuene fortalte studentene hva de opplever som utfordrende når de arbeider med oppgaver i BIOS1100, der de særlig trakk frem at planleggingsfasen kan være utfordrende. Planleggingsfasen består av komponentene problemnedbryting, abstraksjon og algoritmisk tenkning, strategier kjent fra CT og matematikk (Grover & Pea, 2018; Shute et al., 2017; Sneider et al., 2014). Som jeg diskuterte i forrige delkapittel, var dette strategier studentene benyttet seg av da de løste oppgaven jeg ga dem. Likevel gir studentene uttrykk for at å bruke disse strategiene er vanskelig.

Problemnedbryting er en grunnleggende ferdighet for å forstå og lære programmering. Dersom studentene ikke har trening i denne delen av problemløsning blir det vanskelig å bestemme hvilke deler problemet skal brytes ned til, og dermed løse problemet (Keen &

Mammen, 2015). I og med at problemnedbryting er det første steget i planleggingsfasen, og at oppgaven studentene løste hadde gjort dette steget for dem, kan det tenkes at planleggingsfasen var enklere enn i andre oppgaver studentene jobber med. Dette gjorde også at det ble mindre utfordrende å hente ut informasjon fra oppgaveteksten, som igjen førte til at det ble enklere å designe algoritmer. Dersom oppgaven hadde blitt gitt uten den tydelige oppdelingen, kan det tenkes at studentene hadde hatt større utfordringer med å løse den.

Studentene fortalte også at de syntes det er utfordrende med nye programstrukturer og store programmer, hvilket henger sammen med den konseptuelle kunnskapen om hvordan systemet fungerer (Bayman & Mayer, 1988). Studentene fortalte at de har utfordringer knyttet til *når* de skal bruke programstrukturene, og *hvordan* de skal bruke programstrukturene. Utfordringer med den konseptuelle forståelsen kan påvirke studentenes evner til å designe algoritmer, fordi det blir vanskelig å bestemme fremgangsmåten for å løse problemet (Ettles et al., 2018). Dette henger igjen sammen med utfordringer knyttet til planleggingsfasen av en oppgave.

I delkapittel 4.3 viste jeg at en av studentene hadde misoppfattet hvordan *choice*-funksjonen virker. Dette var et eksempel på hvordan ulike nivåer av abstraksjon kan være utfordrende for studentene. Dersom studentene arbeider i et høyere nivå i Python, kan de ignorere hvordan de lavere nivåene i programmet virker (Wing, 2008), som i eksempelet med *choice*-funksjonen. I og med at studentene syntes det er vanskelig å avgjøre hva som skal inn i programmet, kan det hende at de ikke klarer å forholde seg til de ulike nivåene i abstraksjonen. De ender derfor med å skrive lengre programmer enn det som er nødvendig, hvilket øker kompleksiteten i oppgaven, som igjen bidrar til holdningen om at det er utfordrende med store programmer.

5.3 Kompetansen fra R2

I dette delkapittelet vil jeg drøfte studentenes holdninger til kompetansen fra matematikk R2, som svarer på forskningsspørsmålet

Hvordan vurderer studentene relevansen av kompetansen de fikk fra R2 inn i BIOS1100?

Resultatene fra min analyse tilsier at det er særlig to aspekter ved R2-kravet som studentene vurderte som relevante for BIOS1100. Disse er knyttet til de matematiske kunnskapene de får i R2, og de strategiske kunnskapene de får i form av problemløsningsstrategier.

Ut ifra det studentene fortalte i intervjuet, ser ut til at de fleste har hatt nytte av den rent matematiske kompetansen de fikk fra R2. De fleste studentene var enige om at matematikken de møter i BIOS1100 kan være vanskelig dersom man ikke har kjennskap til konsepter som vektorer og likningssett. Majoriteten av studentene opplever at det er mye matematikk i programmering, og anser derfor forkunnskapene fra R2 som relevante for BIOS1100.

En av studentene mente at det ikke gir mening å ha matematikkrav på bachelorutdanningen i biovitenskap, fordi matematikken som brukes gjennomgås som en del av BIOS1100. En liknende holdning ble også uttrykt av en annen student, som mente at biologi er det realfagsfeltet som er lengst vekk fra matematikken. Som jeg presenterte i innledningen til denne studien, er dette holdninger som har eksistert i lang tid, fordi biologi tradisjonelt sett har vært et kvalitativt fagfelt. Dette reflekteres blant annet i kompetansemålene i videregående skole og i yrkesbeskrivelsene av biologen, der for eksempel økologi og artsmangfold hovedsakelig presenteres som kvalitative temaer (Utdanning.no, 2020; Utdanningsdirektoratet, 2006a). Dersom studentene som søker seg til utdanningen har et inntrykk av at matematikk ikke er relevant for fagfeltet, er det forståelig at de opplever R2-kravet som unødvendig. Det er dog bekymringsverdig at flere av studentene opplever matematikk som lite relevant i biologi, når fagfeltet tar i bruk stadig mer kvantitative metoder som krever en god matematisk forståelse.

Som jeg diskuterte i delkapittel 5.1 benytter studentene seg av flere av strategiene kjent fra CT og matematikk når de programmerer. Studentenes matematiske forutsetninger vil påvirke hvor godt de mestrer disse strategiene, da det er en stor overlapp mellom matematikk og CT (Grover & Pea, 2018; Sneider et al., 2014). Et konkret eksempel på hvordan forkunnskaper i matematikk påvirker problemløsningsstrategiene i programmering er hvordan algoritmiske feil oppstår. Algoritmiske feil skjer når man velger feil fremgangsmåte for å løse problemet (Ettles et al., 2018), og oppstår ofte fordi programmereren sliter med å overføre problemet til en matematisk form (Gomes et al., 2006). Dette var den største utfordringen for studentene i Hålands (2019) studie, men det ser ikke ut til at studentene som deltok i min studie hadde samme utfordring da de løste oppgaven. Studentene i min studie nevnte heller ikke andre utfordringer knyttet til matematikk, selv om utfordringene de møtte på ellers var de samme som i studien til Håland, og i andre introduksjonskurs til programmering (Håland, 2019). Dette kan være en indikasjon på at studentene har hatt nytte av kompetansen de fikk fra R2.

Flere av studentene reflekterte over hvordan de strategiske ferdighetene de lærer i matematikken kan bidra til å løse oppgaver i programmering. De trekker frem at det er den

logiske tankegangen man lærer i matematikk som må være årsaken til at det har blitt krav om R2. Ut ifra resultatene i analysen ser det ut til at studentene har hatt større nytte av de strategiske kunnskapene fra matematikken, som for eksempel bruk av abstraksjon og algoritmisk tenkning, enn de rent matematiske kunnskapene, som hvordan en differensiallikning ser ut, når de skal lære å programmere. Dette henger sannsynligvis sammen med at man bruker mange av de samme fremgangsmåtene i matematikk og programmering, hvilket gjør problemløsningsstrategiene i CT og matematikk svært like (Wilkerson & Fenwick, 2017).

5.4 Studentenes arbeid med programmering og biologi

Her vil jeg drøfte resultatene som skal svare på forskningsspørsmålet

Hvilke tanker har studentene om bruk av strategier i arbeid med fagstoff i BIOS1100 og BIOS1110?

Hvordan studentene arbeider i henholdsvis BIOS1100 og BIOS1110 er et relativt stort tema, og jeg kan kun diskutere dette forskningsspørsmålet i lys av hva studentene fortalte i intervjuene. Under intervjuene bemerket jeg meg særlig en ting som er interessant for tematikken i denne studien. Det ser ut til at problemløsningsstrategiene kjent fra matematikk og CT er fraværende hos studentene når de arbeider med BIOS1110, som de tar parallelt med BIOS1100. Studentene beskriver BIOS1110 som et emne der man må «lese og skjønne», og at man lærer fagstoffet ved å pugge. De virker til å være svært avhengige av læreboken, og det ser ut til at de har et inntrykk av at det kun finnes ett riktig svar når de arbeider med oppgaver. Dette er en kontrast til hvordan de arbeider i BIOS1100, der de bruker flere problemløsningsstrategier, som vist i denne studien, og forteller at de kan «finne svaret ved å gå inn i eget hode».

Skillet mellom arbeidsmåtene i de to emnene kan skyldes erfaringene studentene har med biologi fra videregående skole, der man i løpet av kort tid møter mange begreper man må huske. I tillegg har programfaget i biologi båret lite preg av matematiske modeller og aktiv modellering i den nåværende læreplanen (Utdanningsdirektoratet, 2006a). Modellene elevene presenteres for er ofte bilder i læreboka, for eksempel av proteinsyntesen, som inneholder mye informasjon. Det er ikke uvanlig at elever memorerer rekkefølgen på de ulike syntesestegene, fremfor å forstå hvordan prosessen fungerer (Cimer, 2012). Dette ser også ut til å gjelde for førsteårsstudentene på biovitenskap. Studentene ser ut til å ha en forestilling

om at måten man arbeider med det biologifaglige er å skaffe en oversikt over pensum, og deretter pugge dette. Dette er en tilnærming som kan fungere sammen med andre strategier, men som ikke er tilstrekkelig i seg selv dersom studentene skal anvende kunnskapen på nye problemer (Chiou, Liang & Tsai, 2012).

Jeg har flere ganger i denne studien trukket frem at det finnes godt forankrede holdninger om at biologi er unntatt matematikken. Disse holdningene blir også synlige når studentene forteller om sin tilnærming til fagstoffet i BIOS1110 versus BIOS1100. I BIOS1100 er de opptatt av mengdetrening og tid til å prosessere tematikken de jobber med i oppgavene, hvilket er arbeidsmåter som er svært like dem de kjenner fra arbeidet med matematikk. I BIOS1110 derimot, uttrykker de ikke et behov for den samme mengdetreningen og prosesseringen, til tross for at tematikken de jobber med er svært lik den i BIOS1100. Dersom studentene aldri har vært eksponert for en kvantitativ tilnærming til biologi er det ikke overraskende at de ikke bruker problemløsningsstrategier kjent fra CT og matematikk når de arbeider med biologioppgaver i BIOS1110. Dette hadde ikke vært problematisk dersom de hadde benyttet seg av andre problemløsningsstrategier, men når arbeidet med det biofaglige ser ut til å bygge på puggestrategier er det grunn til bekymring.

5.5 Implikasjoner

Ettersom denne studien har sett på hvordan kompetansen fra R2 gjør seg gjeldende i biologistudenters problemløsningsstrategier, kan denne studien få betydning for den pågående debatten om opptakskrav på utdanningen. Studien vil også være relevant for undervisningen på biovitenskap, der problemløsningsstrategier og kvantitative metoder i større grad bør vektlegges som en del av studiet, og i beregningsorienterte emner som BIOS1100. Jeg vil adressere disse implikasjonene i de tre neste avsnittene.

5.5.1 Implikasjoner for valg av opptakskrav til bachelorutdanningene i biovitenskap

Ut ifra funnene i denne studien mener jeg at full fordypning i realfagsmatematikk vil være positivt for studentene på biovitenskap, fordi de vil ha et bedre utgangspunkt for å arbeide med beregningsorientert biologi. Fagfeltet benytter seg stadig mer av kvantitative metoder, hvilket er hovedargumentet for å sette R2 som opptakskrav på bachelorprogrammet (Mørken, intervjuet i Hystad & Tønnesen, 2019). Matematikk er et fag som lærer oss å tenke logisk, og vil dermed tilrettelegge for bruk av problemløsningsstrategier som er nyttige når biologer

møter nye utfordringer knyttet til store datasett. Disse utfordringene vil kun la seg løse dersom matematikken integreres med biologien, slik vi har sett i forbindelse med koronasituasjonen, der matematisk modellering har informert en hel verden om utbrudd, smitte og spredning av viruset våren 2020.

For studentene på biovitenskap vil ikke disse kvantitative metodene utelukkende brukes i emner som BIOS1100, men gjennom hele studiet, og senere også i arbeidslivet. Det er derfor noe foruroligende at anerkjente studiesteder, som Universitetet i Bergen, anser R2-kravet som unødvendig for studentene på biovitenskap. Universitetet i Bergen valgte å trekke seg fra prøveordningen fordi det var langt færre søkere til utdanningen etter at R2-kravet ble innført i 2019, men søkertallet gikk ned hos alle studiestedene som tilbyr en bachelorutdanning i biovitenskap (Samordna Samordna, 2019). Denne trenden har snudd, og i 2020 var det nettopp bachelorprogrammet i biovitenskap ved Universitetet i Oslo som var det mest populære realfagsstudiet (målt i antall enkeltsøkere) i Norge, til tross for at R2-kravet ble stående (Valberg, 2020). Dette indikerer at søkerne er villige til å ta R2 for å komme inn på bachelorprogrammet.

Universitetet i Bergen argumenterer for at de kan se bort ifra R2-kravet, fordi studentene kan lære matematikken *etter* at de har begynt på studiet (Walderhaug, intervjuet i Hystad & Tønnesen, 2019). Det er fullt mulig å integrere matematikken som en del av utdanningen, men om dette vil gi det samme grunnlaget for arbeid med beregningsorientert biologi som det R2-kravet gjør, er usikkert. Dette skyldes at det i stor grad er de strategiske ferdighetene hos studentene, i tillegg til de rent matematiske, R2-kravet har tilrettelagt for. Dersom studiestedene ikke vil sette krav om R2, men i stedet lære studentene matematikken de trenger som en del av studiet, fordrer dette at de i større grad integrerer problemløsningsstrategier som en del av utdanningen. Denne implikasjonen vil diskuteres i neste avsnitt.

5.5.2 Implikasjoner for undervisningen på biovitenskap

Mye av det jeg har presentert i denne studien indikerer at studentene har hatt nytte av kompetansen de fikk fra R2, ved at de bruker problemløsningsstrategier kjent fra matematikk og CT i arbeid med beregningsorientert biologi. Jeg må likevel understreke at full fordypning i realfagsmatematikk ikke er nok i seg selv for at studentene skal anvende problemløsningsstrategiene fra CT og matematikk, men det gir et godt utgangspunkt. Både CT og matematisk tenkning involverer strategier for hvordan man løser et problem (Wing, 2008), og vil være nyttige for studentene når de skal arbeide med biologi i en tid der fagfeltet

blir stadig mer kvantitativt. Derfor bør utdanningen legge til rette for at studentene lærer å bruke disse problemløsningsstrategiene, slik at de bedre kan møte de problemstillingene vi står ovenfor i dag.

Jeg har i denne studien blant annet sett på hvilke strategier studentene bruker når de arbeider med programmeringsoppgaver, og hvilke tanker de har om arbeid med biologi og programmering. Ut ifra min analyse ser det ut til at studentenes fordypning i matematikk har gitt dem et godt utgangspunkt for arbeid med beregningsorientert biologi, der de har vist at de kan anvende problemløsningsstrategier kjent fra matematikk og CT når de arbeider med oppgaver i programmering. Samtidig forteller studentene at de ikke bruker disse strategiene når de arbeider med andre typer biologioppgaver. Jeg mener derfor at strategier for problemløsning bør integreres som en del av undervisningen for studentene på biovitenskap, både i emner som fokuserer på beregningsorientert biologi, som BIOS1100, men også i de øvrige emnene.

Uavhengig av hvordan studiestedene velger å integrere CT og matematisk tekning som en del av bachelorutdanningen i biovitenskap, bør de ta hensyn til hvilke forkunnskaper studentene har før de begynner på studiet. Dersom studentene har bestått R2 vil de sannsynligvis ha et bedre utgangspunkt, fordi de har arbeidet mer med problemløsningsstrategiene tidligere. Studiestedene må derfor ta stilling til om studentene primært skal arbeide med problemløsningsstrategiene etter at de har begynt på studiet, eller om de skal begynne på studiet med noen forkunnskaper fra R2.

5.5.3 Implikasjoner for BIOS1100

I forrige avsnitt argumenterte jeg for at implementering av problemløsningsstrategier bør foregå i alle emner på bachelorutdanningen i biovitenskap. Hvordan undervisningen skal legges opp for å integrere CT og matematisk tenkning i utdanningen har ikke vært en del av denne studien, og vil derfor ikke adresseres her. Det jeg har sett på i denne studien er hvordan studentene arbeider med programmeringsoppgaver i BIOS1100, og funnene fra analysen kan derfor få betydning for undervisningen av problemløsningsstrategier i dette emnet.

Da studentene arbeidet med oppgaven, viste de at de mestret flere av strategiene kjent fra CT og matematikk. Samtidig fortalte de i intervjuene at de syns det er utfordrende med planleggingsfasen i en oppgave, selv om planleggingsfasen består av mange av de samme strategiene som studentene brukte da de løste oppgaven. Som jeg diskuterte i delkapittel 5.2,

kan det tenkes at studentene hadde hatt større utfordringer med å løse oppgaven dersom den ikke hadde hatt den tydelige oppdelingen. Jeg mener derfor at studentene bør få trening i å anvende problemløsningsstrategiene på mer sammensatte oppgaver, slik at de får øvd på å planlegge hvordan de skal løse et problem.

Undervisningen på BIOS1100 bør også tilrettelegge for at studentene får arbeide med abstraksjon på ulike nivåer. En av hovedutfordringene for studentene er å avgjøre hvilke programstrukturer som skal inngå i programmet, som kan henge sammen med at de ikke forstår hvordan den virker, som i tilfellet med *choice*-funksjonen. Dersom studentene får en bedre oversikt over de ulike nivåene av abstraksjon, vil de forhåpentligvis få et bedre utgangspunkt for å skrive kortere og mer konsise programmer, fordi de kan se bort ifra forklarende informasjon i programmet (Weintrop et al., 2016; Wing, 2014). Som jeg diskuterte i delkapittel 5.2, vil dette videre kunne møte utfordringen studentene har med store programmer.

5.6 Begrensninger og forslag til videre forskning

I denne studien har jeg undersøkt hvilke problemløsningsstrategier et lite utvalg studenter bruker når de arbeider med programmeringsoppgaver i et emne i beregningsorientert biologi. For å kunne generalisere funnene fra denne studien måtte jeg ha sett på et langt større utvalg, men dette var dessverre ikke mulig grunnet studiens omfang og tidsbegrensning. Studien har vært deskriptiv, og jeg har forsøkt å identifisere hvilke problemløsningsstrategier studentene bruker, hvilke utfordringer de møter, hvordan de vurderer kompetansen fra R2 og hvilke tanker de har om bruk av problemløsningsstrategier i arbeid med biologi og programmering. Problemstillingen i denne studien har vært relativt vid, og det har derfor ikke vært mulig å gå i dybden på hvert enkelt forskningsspørsmål. Jeg tror derfor det vil være mye forskning som gjenstår for å forstå studentenes bruk av problemløsningsstrategier i biologi.

Denne studien er gjennomført på et tidspunkt der konseptet «computational thinking in biology» fremdeles er svært ungt. Jeg håper at min studie blir et bidrag for videre forskning på feltet, og jeg har følgende forslag til videre studier:

- *Computational thinking in Biology:* Hvilke problemløsningsstrategier bruker studentene når de arbeider i mer tradisjonelle biologiemner?

- *Fokus på strategier:* Hvordan kan undervisningen på bachelorprogrammet i biovitenskap tilrettelegge for utvikling av studentenes problemløsningsstrategier i biologi?

- *Fagfornyelsen:* Fra 2021 vil det være nye kompetansemål for programfaget i biologi i videregående skole. Hvordan vil dette påvirke de strategiske ferdighetene til studentene? Og hvordan vil dette påvirke holdningene om biologi som et kvalitativt fagfelt?

6 Konklusjon

I denne studien har jeg undersøkt hvordan kompetansen fra R2 gjør seg gjeldende i studenters bruk av problemløsningsstrategier når de arbeider med programmering i BIOS1100. Jeg har forsøkt å svare på problemstillingen gjennom fire forskningsspørsmål. I dette kapittelet vil jeg gi en kort oppsummering av funnene fra denne studien.

De to første forskningsspørsmålene handler om hvilke problemløsningsstrategier kjent fra matematikk og CT studentene benytter seg av, og hvilke utfordringer de møter når de skal bruke disse strategiene. Funnene fra denne studien viser at studentene bruker strategien abstraksjon ved at de henter ut relevant informasjon fra oppgaven og gjenkjenner mønstre. De benytter seg av algoritmer ved at de tenker ut et sett med strukturerte steg for å løse problemet, der de tidvis programmerer effektivt og automatiserer programmene sine. Videre viser de i noen grad at de benytter seg av problemnedbryting, men denne strategien var vanskelig å undersøke grunnet oppgavens tydelige oppdeling. Oppgaven som ble brukt i denne studien var ikke spesielt utfordrende for studentene, men de viste likevel til utfordringer knyttet til bruk av problemløsningsstrategier i programmering. Studentene trakk frem planleggingsfasen som særlig utfordrende når de skal løse oppgaver i BIOS1100. I planleggingsfasen brukes strategien problemnedbryting, samt komponenter fra strategiene abstraksjon og algoritmisk tenkning, de samme strategiene studentene brukte da de løste oppgaven de ble gitt i denne studien. Utfordringer med planleggingsfasen tyder på at studentene syntes det er vanskelig å bruke problemløsningsstrategiene kjent fra matematikk og CT, særlig når de skal løse oppgaver uten tydelig oppdeling.

Det tredje forskningsspørsmålet tok for seg hvordan studentene vurderer relevansen av kompetansen de fikk fra R2, inn i BIOS1100. Det ble trukket frem to aspekter ved R2-kravet som studentene mente var relevante for emnet. Disse knytter seg til den matematiske kompetansen i form av regning, og den strategiske kompetansen i form av problemløsningsstrategier. Det var delte meninger blant studentene om hvorvidt matematiske forkunnskaper er nødvendig, i og med at mange av de matematiske konseptene studentene møter i BIOS1100 undervises som en del av emnet. Flere av studentene trakk frem at det er den logiske tankegangen man lærer i matematikken som må være årsaken til at det har blitt krav om R2, og at den strategiske kompetansen de har fått fra matematikken har vært nyttig når de skal arbeide med oppgaver i BIOS1100.

Avslutningsvis undersøkte jeg hvilke tanker studentene har om bruk av strategier når de arbeider med fagstoff i BIOS1100 og BIOS1110. Resultatene fra dette forskningsspørsmålet baserer seg utelukkende på studentenes selvrapportering under intervjuet, og inkluderer ikke undersøkelser om hvordan studentene faktisk arbeider i Celle- og molekylærbiologi. Ut ifra hva studentene fortalte under intervjuet, ser det ikke ut til at de benytter seg av problemløsningsstrategiene kjent fra matematikk og CT når de arbeider med oppgaver i BIOS1110. Det ser derimot ut til at studentene tilnærmer seg fagstoffet ved å først og fremst pugge det som står i læreboka, og deretter løser oppgaver ved å reprodusere denne kunnskapen. Dette er en kontrast til hvordan de arbeider i BIOS1100, der funnene fra denne studien tilsier at de benytter seg av flere problemløsningsstrategier, og i større grad ønsker å forstå og prosessere fagstoffet de arbeider med.

Biologi som fagfelt er i endring, og den moderne biologien formes av både statistikk og nye kvantitative analysemetoder. Ny kunnskap gjøres tilgjengelig gjennom beregninger og modellering, og tverrfaglig kunnskap om biologi, matematikk og datavitenskap blir stadig viktigere. I denne studien har jeg undersøkt hvordan kompetansen fra R2-matematikk gjør seg gjeldende i biologistudenters problemløsningsstrategier når de skal løse oppgaver i beregningsorientert biologi. Funnene fra denne studien tyder på at kompetansen fra R2 har vært relevant for studentene, og at de har tatt i bruk problemløsningsstrategier kjent fra matematikk og CT. Samtidig viser denne studien at studentene har utfordringer knyttet til bruk av disse strategiene, hvilket tydeliggjør et behov for økt fokus på arbeid med problemløsningsstrategier i biologi.

Litteraturliste

Altadmri, A. & Brown, N. C. (2015). 37 million compilations: Investigating novice programming mistakes in large-scale student data. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (s. 522-527).

Ambrósio, A., Costa, F., Almeida, L., Franco, A. & Macedo, J. (2011). Identifying cognitive abilities to improve CS1 outcome. *Proceedings - Frontiers in Education Conference*. https://doi.org/10.1109/FIE.2011.6142824

Anderson, N. D. (2016). A call for computational thinking in undergraduate psychology. *Psychology Learning & Teaching, 15*(3), 226-234. https://doi.org/10.1177/1475725716659252

Bayman, P. & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM, 26*(9), 677-679. https://doi.org/http://dx.doi.org/10.1145/358172.358408

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

Befring, E. (2015). *Forskningsmetoder i utdanningsvitenskap*. Oslo: Cappelen Damm akademisk.

Blikstad-Balas, M. (2017). Key challenges of using video when investigating social practices in education: contextualization, magnification, and representation. *International Journal of Research & Method in Education, 40*(5), 511-523. https://doi.org/10.1080/1743727X.2016.1181162

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

Chiou, G.-L., Liang, J.-C. & Tsai, C.-C. (2012). *Undergraduate Students' Conceptions of and Approaches to Learning in Biology: A Study of Their Structural Models and Gender Differences*.

Cimer, A. (2012). What makes biology learning difficult and effective: Students' views. *Educational Research and Reviews, 7*(3), 61-71.

Cohen, L., Morrison, K. & Manion, L. (2007). *Research methods in education* (6th ed. utg.). London: Routledge.

Creswell, J. W. & Miller, D. L. (2000). Determining Validity in Qualitative Inquiry. *Theory Into Practice, 39*(3), 124-130. https://doi.org/10.1207/s15430421tip3903_2

Dalen, M. (2011). *Intervju som forskningsmetode*. Oslo: Universitetsforlaget.

Descartes, R. (1956). *Discourse on method* (2. ed. utg.). New York: Macmillan Publishing.

Dvergsdal, H. (2019). Python - programmeringsspråk. Hentet fra https://snl.no/Python_-_programmeringsspr%C3%A5k

Dæhlen, M. (2016, 7. mars 2016). Nye opptakskrav fra 2018/2019 [Blogginnlegg]. Hentet fra https://titan.uio.no/blogg/2020/nye-opptakskrav-fra-20182019

Engen, S. (2020). Koronaepidemien - litt matematisk oppklaring midt i depresjonen *Universitetsavisa*. Hentet fra https://www.universitetsavisa.no/koronavirus/2020/03/24/Koronaepidemien-litt-matematisk-oppklaring-midt-i-depresjonen-21423468.ece

Ettles, A., Luxton-Reilly, A. & Denny, P. (2018). Common logic errors made by novice programmers. *Proceedings of the 20th Australasian Computing Education Conference* (s. 83-89).

Everett, E. L. & Furseth, I. (2012). *Masteroppgaven : hvordan begynne - og fullføre* (2. utg. utg.). Oslo: Universitetsforl.

Gomes, A., Carmo, L., Bigotte, E. & Mendes, A. (2006). Mathematics and programming problem solving. *3rd e-learning conference–computer science education* (s. 1-5): Citeseer.

Grover, S. & Pea, R. (2018). Computational Thinking: A competency whose time has come. I S. Sentance, E. Barendsen & C. Schulte (Red.), *Computer science education: Perspectives on teaching and learning in school* (bd. 19, s. 20-38). London: Bloomsbury Academic.

Hystad, J. & Tønnesen, E. (2019, 02.12.2019). Fjerner mattekrav på realfag. *Khrono*. Hentet fra https://khrono.no/fjerner-mattekrav-pa-realfag/423904

Håland, L. E. (2019). *Studenters arbeid med programmering i biovitenskapelige problemstillinger* (Masteravhandling). Universitetet i Oslo, Oslo. Hentet fra https://www.duo.uio.no/bitstream/handle/10852/70499/Masteroppgave_H-land.pdf?sequence=8&isAllowed=y

Institutt for Biovitenskap & Universitetet i Oslo. (2017). BIOS1100 – Innføring i beregningsmodeller for biovitenskap. Hentet fra https://www.uio.no/studier/emner/matnat/ibv/BIOS1100/

Johannessen, A., Christoffersen, L. & Tufte, P. A. (2016). *Introduksjon til samfunnsvitenskapelig metode* (5. utg. utg.). Oslo: Abstrakt.

Johnson, B. & Christensen, L. B. (2017). *Educational research : quantitative, qualitative, and mixed approaches* (Sixth edition. utg.). Thousand Oaks, California: SAGE Publications.

Keen, A. & Mammen, K. (2015). Program decomposition and complexity in CS1. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (s. 48-53).

Kleven, T. A. (2014). Data og datainnsamlingsmetoder. I T. A. Kleven (Red.), *Innføring i pedagogisk forksningsmetode* (s. 17-31). Bergen: Fagbokforlaget.

Kunnskapsdepartementet. (2016). *Rundskriv om endringer i opptaksforskriften 2016* (F-01-16). Hentet fra https://www.regjeringen.no/contentassets/4a8519c4fbe84b328be587c5044fe34e/rundskriv2016-17-l1006955.pdf

Larsen, A. K. (2017). Om samfunnsvitenskapekig metode. I A. K. Larsen (Red.), *En enklere metode. Veiledning i samfunnsvitenskapelig metode* (bd. 2, s. 17-31). Bergen: Fagbokforlaget.

Lerdal, A. & Karlsson, B. (2008). Bruk av fokusgruppeintervju. *Sykepleien forskning, 3*(3), 172-175.

Markowetz, F. (2017). All biology is computational biology. *PLoS biology, 15*(3), e2002050.

Medeiros, R. P., Ramalho, G. L. & Falcão, T. P. (2018). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education, 62*(2), 77-90.

Müller, L., Silveira, M. S. & de Souza, C. S. (2018). Do I Know What My Code is" Saying"? A study on novice programmers' perceptions of what reused source code may mean. *Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems* (s. 1-10).

Nortvedt, G. A. & Bulien, T. (2018). *Norsk Matematikkråds forkunnskapstest* Hentet fra https://matematikkradet.no/rapport2017/NMRRapport2017.pdf

Omholdt, S. W. (2008). Fronter biologiens nye retning *UMB nytt,* (2), 18-19. Hentet fra http://www.umb.no/statisk/alumni/2008/umb_nytt_0208.pdf

Papert, S. (1980). *Mindstorms : children, computers, and powerful ideas*. New York: Basic Books.

Priami, C. (2007). Computational Thinking in Biology. I C. Priami (Red.), *Transactions on Computational Systems Biology VIII* (1st ed. 2007. utg., s. 63-77). Berlin, Heidelberg: Springer Berlin Heidelberg : Imprint: Springer. Hentet fra https://link-springer-com.ezproxy.uio.no/content/pdf/10.1007%2F978-3-540-76639-1.pdf

Qian, Y. & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE), 18*(1), 1-24.

Sadler, P. & Tai, R. H. (2007). Transitions - The two high-school pillars supporting college science. *Science, 317*(5837), 457-458. https://doi.org/10.1126/science.1144214

Samordna opptak. (2019). *Hovedopptaket til høyere utdanning ved universiteter og høyskoler gjennom Samordna opptak*. Hentet fra https://www.samordnaopptak.no/info/om/sokertall/sokertall-2019/faktanotat-hovedopptaket-2019.pdf

Sandelowski, M. (1995). Qualitative analysis: What it is and how to begin. *Research in Nursing & Health, 18*(4), 371-375. https://doi.org/10.1002/nur.4770180411

Searls, D. B. (2018). Computational biology. Hentet fra https://www.britannica.com/science/computational-biology

Shute, V. J., Sun, C. & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142-158.

Sneider, C., Stephenson, C., Schafer, B. & Flick, L. (2014). Computational thinking in high school science classrooms. *The Science Teacher, 81*(5), 53-59.
Tjora, A. H. (2017). *Kvalitative forskningsmetoder i praksis* (3. utg. utg.). Oslo: Gyldendal akademisk.
Utdanning.no. (2020). Yrkesbeskrivelse - Biolog. Hentet fra https://utdanning.no/yrker/beskrivelse/biolog#intervju
Utdanningsdirektoratet. (2006a). *Læreplan i biologi - programfag i utdanningsprogram for studiespesialisering (BIO1-01)*. Hentet fra https://www.udir.no/kl06/BIO1-01/Hele/Formaal
Utdanningsdirektoratet. (2006b). *Læreplan i matematikk for realfag - programfag i utdanningsprogram for studiespesialisering (MAT3-01)*. Hentet fra https://www.udir.no/kl06/MAT3-01
Utdanningsdirektoratet. (2019). *Høring - læreplaner i matematikk*. Utdanningsdirektoratet på oppdrag fra Kunnskapsdepartementet. Hentet fra https://hoering.udir.no/Hoering/v2/343?notatId=656
Valberg, A. (2020). Mattekrav stopper ikke søkerne. Hentet fra https://www.mn.uio.no/om/aktuelt/aktuelle-saker/2020/sokertall-2020.html
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127-147.
Wilkerson, M. H. & Fenwick, M. (2017). Using mathematics and computational thinking. I C. V. Schwarz, C. Passmore & B. J. Reiser (Red.), *Helping students make sense of the world using next generation science and engineering practices* Arlington: National Science Teachers Association.
Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366*(1881). https://doi.org/10.1098/rsta.2008.0118
Wing, J. M. (2014). Computational thinking benefits society. *40th Anniversary Blog of Social Issues in Computing*.

Vedlegg

Vedlegg A – Intervjuguide

Velkommen til denne økten med en gruppeoppgave, etterfulgt av en uformell samtale om programmering og modellering i biologi!

Dere har nå samtykket til dette forskningsprosjektet som handler om programmering og modellering i biologi. Prosjektet er et masterprosjekt med veiledere fra institutt for biovitenskap og fysisk institutt. Målet med prosjektet er å undersøke hvilke utfordringer studenter kan møte på når de programmerer i biologi og å finne ut hvordan studentene velger å løse ulike de ulike problemstillingene. Gjennom prosjektet ønsker vi å bidra til bedre læring og motivasjon i beregningsorientert biologi.

Nå skal dere først arbeide med en oppgave der dere skal programmere i biologi. Deretter tar vi et litt uformelt intervju sammen, hvor vi snakker om hvordan dere løste oppgaven og hvilke utfordringer som dere eventuelt møtte på.

Det er frivillig å delta i denne diskusjonen, og dere kan når som helst, uten begrunnelse, trekke dere. Vi ønsker å gjøre lydopptak av både oppgave og diskusjonen. Opptaket skal bare brukes til forskningsformål og vil ikke innvirke på vurdering eller karaktersetting av dere i noen av emnene dette semesteret. Opptaket vil behandles konfidensielt; dere vil ikke identifiseres med navn eller kunne gjenkjennes på annen måte i rapporter fra forskningen. Ønsker noen å trekke seg? Hvis ikke, starter vi lydopptaket nå. Vi ønsker at dere i størst mulig grad skal **diskutere med hverandre ut fra relativt åpne spørsmål og temaer som vi tar opp.**

1. **Litt generelt om dere**
    - Fortell litt om dere selv og hvorfor dere har valgt biovitenskap.
    - Hvor gamle er dere?
    - Har dere studert før?
    - Har dere programmert før BIOS1100?
    - Hvor ofte jobber dere med programmering i BIOS1100?
    - Jobber dere sammen med andre når dere jobber med programmering? Hvorfor/hvorfor ikke?
    - Hvor ofte spør dere hjelpelæreren deres?
    - For hver oppgave, eller noen ganger i løpet av økta?
    - Hva tenker dere om å programmere med biologiske problemstillinger?
        - Hvordan påvirker det motivasjonen i BIOS1100?
        - Synes dere at det blir det enklere å programmere når det er en biologisk problemstilling?
    - Hva tenker dere om BIOS1100 som biologiemne?
    - Hvordan er det å være student BIOS1100 i sammenligning med BIOS1110?

2. **Hva tenker dere om oppgaven dere jobbet med nå?**
    - Hvordan var den å arbeide med?
    - Hvordan var den i forhold til hva dere pleier å arbeide med i kurset?
    - Bruker dere kunnskaper dere har fra før når dere løser oppgaver i programmering? Altså ting dere lærte på videregående, eller kan fra fritiden

3. **Hvordan gikk dere frem for å løse oppgaven?**
    - Hva er det første dere gjør når dere begynner å løse en oppgave?
    - Bruker dere noen bestemte strategier eller metoder?
    - Har dere lært noen bestemte strategier eller fremgangsmåter for å løse programmeringsoppgaver?
    - Hva tenker dere om oppgaveteksten?
    - Hvordan gikk dere frem for å finne ut hva teksten spurte om?
    - Hvordan gikk dere frem for å finne genotypen til de ulike fenotypene i oppgaven?
    - Hvilke koder valgte dere å bruke? Hvorfor det?
    - Opplevde dere noen feil i kodene? Hvordan løste dere dem?
    - Er det noen sammenhenger i hvordan dere tenker når dere løser oppgave a og b, i forhold til resten av oppgaven?
    - Hvorfor valgte dere å bruke/ ikke bruke hjelpelærer da dere løste oppgaven?
    - Da dere følte dere ferdige med programmeringen, hvordan vurderte dere koden? Virket svaret koden ga riktig ut ifra oppgaven?

4. **Var det noe dere opplevde som spesielt utfordrende i oppgaven?**
    - Hvor opplevde dere at utfordringen oppstod?
    - Hva er det som gjør dette utfordrende å arbeide med?
    - Hvordan var oppgaveteksten? Var det noen ord eller uttrykk dere lurte på?
    - Hvordan var det å finne genotypen i oppgave a?
    - Hvordan var det å regne ut sannsynligheten for arv i oppgave b?
    - Hvordan var det å finne frem riktige koder og funksjoner i oppgave c?
    - Kan dere si noe generelt om hva dere synes er vanskeligst å arbeide med når dere programmerer?
    - Hva med oppgaven vil dere si var mest utfordrende? Hva var minst utfordrende?

5. **Spørsmål lagt til intervjuguiden under det første intervjuet**

    - Hva syntes dere om at det har blitt krav om R2 på bachelorutdanningen i biovitenskap?

Vedlegg B – Samtykkeskjema

Forespørsel om deltakelse i forskningsprosjektet

Programmering og modellering i biolog: Intervju og lydopptak

Bakgrunn
Det matematisk- naturvitenskapelige fakultet ved Universitetet i Oslo la om alle sine studieprogrammer høsten 2017, og programmering og modellering er nå blitt en integrert del i hele studieløpet. Fakultetet har også et senter for fremragende utdanning, Centre for Computing in Science Education (CCSE), som støtter og forsker på innføringen av programmering og modellering i realfagsstudiene. På bachelorstudiet i biovitenskap møter studentene programmering og modellering allerede første semester i et eget emne, BIOS1100 – Innføring i beregningsmodeller for biovitenskap.

Forskningsprosjektet
Formålet med prosjektet er å undersøke studenters holdninger og motivasjon for programmering og modellering i biologi, samt finne ut hvordan studentene velger å løse ulike biologiske problemstillinger ved hjelp av programmering. Gjennom prosjektet ønsker vi å bidra til bedre læring og motivasjon i beregningsorientert biologi.

Hva innebærer deltakelse i studien?
Datainnsamling vil skje i form av dybdeintervju med enkeltstudenter samt lydopptak fra deler av gruppeundervisningen i BIOS1100. Intervjuene vil bli benyttet til å få en bedre forståelse av resultater fra spørreskjemaundersøkelsen (eget samtykkebrev). Lydopptak i gruppeundervisning vil bli benyttet for å kunne få en bedre forståelse for hvordan du/dere arbeider når du/dere skal løse biologiske problemstillinger med programmering og modellering.

Hva skjer med informasjonen vi samler inn?
Alle data som kan være personidentifiserende vil lagres på sikre servere ved UiO. Det er kun ansvarlige for studien som vil ha tilgang til dataene. Dataene vi samler inn vil være bakgrunnsdata for flere fagartikler publisert i vitenskapelige tidsskrift og for presentasjoner på vitenskapelige konferanser. Ingen personidentifiserende data skal publiseres.

Det er tenkt å samle data fra emnet i 3 år for å kunne sammenlikne og se på endring i holdninger og motivasjon over denne perioden. Inkludert analyser og publisering er prosjektet tenkt å ha en varighet på inntil 5 år, til 2023.

Frivillig deltakelse
Det er frivillig å delta i studien, og du kan når som helst trekke ditt samtykke uten å oppgi noen grunn.

Dersom du ønsker å delta i studien, signerer du skjema og lever direkte til oss. Har du spørsmål til studien, ta kontakt med Tone Fredsvik Gregers (tlf. 996 97 154). E-post t.f.gregers@ibv.uio.no.

Studien er meldt inn til Personvernombudet for forskning, NSD - Norsk senter for forskningsdata AS.

Tone Fredsvik Gregers (Førstelektor, Institutt for biovitenskap)

Samtykke til deltakelse i forskningsprosjektet «Programmering og modellering i biologi»

Jeg har mottatt informasjon om forskningsprosjektet «Programmering og modellering i biologi», og er villig til å delta i intervju og/eller lydopptak i undervisningen.

☐ Lydopptak i undervisningen

☐ Intervju

_____

Signatur

Vedlegg C – Oppgave med løsningsforslag

Autosomal recessive disorder appears in individuals with two abnormal copies of a gene. The genes are located on an autosome, which is a chromosome that is not a sex chromosome. Humans usually have 22 pairs of autosomes and one pair of sex chromosomes. The inherited trait of an autosomal recessive disorders usually skip a few generations, while dominant traits are usually present in every generation. Therefore, the individual that inherits an autosomal recessive disorder must inherit the abnormal alleles from parents that are carriers or from parents where one of them is a carrier and the other has the disorder. Examples of autosomal recessive disorders are cystic fibrosis, sickle cell disease, and phenylketonuria.

This exercise is about a three generation family: Grandparents, parents and grandchildren. The grandparents are the parents of the parent mother.

a) The parents and grandparents show no signs of the autosomal recessive condition, however one of the grandchildren is affected by an autosomal recessive disorder. Given this information, can you identify the genotypes of the parents and the grandparents?

Answer. Based on the given information where neither parents nor grandparents show any symptoms but one of the grandchilden is affected by the disorder, we know that both parents must be carriers (even though we don't have any information about the father's parents) and at least one of the grandparents must be a carrier. The parents are therefore heterozygous carriers with genotypes: ["A", "a"] and ["A", "a"]. The grandparents' genotypes are either ["A", "A"] and ["A", "a"] or ["A", "a"] and ["A", "a"].

b) Given that the parents are carriers and we do not know anything about the grandchildren, what is then the probability of having three affected grandchildren? Calculate the probability using Python and print the answer to screen.

Answer. Since we know that the parents are carriers, the chance that they have an affected child is 1/4 (["a", "a"]).

The probability of three affected grandchildren is therefore 1/4 * 1/4 * 1/4 = 1/64.

Solution.

In [31]:
```
prob_affected = (1/4 * 1/4 * 1/4)**3 * 100
print("The probability of three affected grandchildren is", prob_affected)
```

```
The probability of three affected grandchildren is 0.0003814697265625
```

c) Now that you are familiar with the probabilities of inheriting the disease allele, you are going to check the genotypes and phenotypes of 5 grandchildren. You only need to consider the parental alleles and their offspring.

Write a program that iterates over the 5 children and randomly generates the child's genotype from the parental alleles. Remember, if the child has the dominant allele present it will possess a normal phenotype. Print the genotypes and corresponding phenotypes for each child to screen.

Answer. For example, output could be:

```
aa affected
aA normal
AA normal
aa affected
Aa normal
```

Solution.

In [32]:
```
from pylab import *

parent_1 = ["A", "a"]  # carrier
parent_2 = ["A", "a"]  # carrier

children = 5
i = 0

while i < children:
    allele_1 = choice(parent_1)
    allele_2 = choice(parent_2)

    genotype = allele_1 + allele_2

    if "A" in genotype:
        phenotype = "normal"
    else:
        phenotype = "affected"

    print(genotype, phenotype)
    i += 1
```

```
aa affected
Aa normal
aa affected
AA normal
aA normal
```

d) d) Run the program you wrote a couple of times. Do you get the same output each time? Is this expected?

e) Add a counting step to your program where you count the number of affected and normal offspring (you are free to reuse your previous code). Print the probability of having affected children and the probability of having normal children to screen. Do they correlate with the true probabilities?

Try this for 1000 children and calculate the same probabilities, were they different for 5 and 1000 children?

Answer. The probabilities become more accurate if the parents produce more offspring.

1 affected and 4 normal phenotypes: Probability of affected children: 20.0 % Probability of normal children: 80.0 %

Or 252 affected and 748 normal phenotypes: Probability of affected children: 25.2 % Probability of normal children: 74.8 %

Solution.

In [33]:
```
from pylab import *

parent_1 = ["A", "a"]  # carrier
parent_2 = ["A", "a"]  # carrier

children = 5  # or change to 1000
i = 0

affected = 0
normal = 0

while i < children:
    allele_1 = choice(parent_1)
    allele_2 = choice(parent_2)

    genotype = allele_1 + allele_2

    if "A" in genotype:
        normal = normal + 1
    else:
        affected = affected + 1
    i += 1

print("Probability of having affected children:", affected / children * 100, "%")
print("Probability of having normal children:", normal / children * 100, "%")
```

```
Probability of having affected children: 20.0 %
Probability of having normal children: 80.0 %
```

f) Again, run the program you wrote a couple of times. Do you get the exact same output each time? Is this expected?