Rettferd i variantoppgåver med tilfeldig trekking: Erfaringar frå kvalitetssikringsarbeid

Guttorm Sindre [0000-0001-5739-8265]

Institutt for datateknologi og informatikk, NTNU, Trondheim, Noreg
guttorm.sindre@ntnu.no

Samandrag. I automatiserte prøver som studentar skal kunne ta om att mange gonger, er det behov for å lage store mengder oppgåver som blir trekte tilfeldig frå gong til gong. Ved formativ bruk av prøvene trengst variantar for å unngå at studentar etter kvart svarer rett på alt berre ved å pugge svar, og ved summativ bruk av prøvene trengst variantar for å hindre at gjentak gir ein urettvis fordel framfor å ta prøva på første forsøk. Eit mogeleg problem med tilfeldig trekking mellom mange variantar, er dersom variantane har ulik vanskegrad, slik at studentar kan vere meir eller mindre heldige med kva oppgåver som blir trekte for dei. Denne artikkelen ser på nokre variantoppgåver som blei brukte både i formative treningstestar og summative meistringstestar i eit innleiande programmeringskurs i Python for førsteårs universitetsstudentar, meir spesifikt oppgåver som skulle teste forståing av if-setningar i Python. Spørsmåla vi ser på er følgjande: 1) Kva analyser kan faglærar gjere for å sjekke om oppgåvevariantar er rettferdige? 2) Var oppgåvene rettferdige ved dette konkrete høvet? 2) I den grad det blei funne variasjon i vanskegrad mellom variantar, kva var årsaka til denne? Spørsmåla blei undersøkte ved statistisk analyse av anonyme studentsvar på oppgåvene. Funn tyder på at jamvel om kodekompleksiteten var lik i alle variantane, var det noko variasjon i vanskegrad. Mogeleg forklaring kan liggje i samanhengen mellom koden og utforminga av oppgåveteksten eller svaralternativa. Artikkelen avsluttar med idear til korleis oppgåvene kunne ha vore gitt på ein meir rettferdig måte.

Nøkkelord: programmering, kodeforståing, variantar, rettferd

1 Introduksjon

Det finst fleire motivasjonar for å utvikle spørsmålsbankar med mange variantar av dei same oppgåvene, både innan einskildkurs og for potensiell deling mellom universitet [1]. Ein motivasjon kan vere å motverke samarbeid når studentar sit ganske nær kvarandre under prøva, eller tar den som ein ukontrollert heimeeksamen [2-4]. Ein annan motivasjon kan vere å hindre at studentar får eksakt same oppgåver om att ved gjentak av prøva, anten det er summativt [5] eller formativt [6, 7].

Bakgrunnen for denne artikkelen er automatiserte testar i emnet IT1001 Informasjonsteknologi Grunnkurs ved NTNU, som var undervist første gong hausten 2023 for

ein klasse av 48 studentar i første år på eit 5-åring realfagslektorstudium. IT1001 er eit innleiande programmeringskurs med emnedesign basert på meistringslæring, og da mest likt på Keller sitt PSI [8, 9]. Pensum er delt opp i 9 modular, der kvar modul har ein automatisert test som må passerast med ein score på over 90% for å gå vidare til neste nivå. Dei 9 modulane er kalla I, H, …, A, der studenten må greie alle 9 for å ta karakteren A, 5 for å ta lågaste ståkarakter E. Det var mogeleg å ta ein teljande test kvar fredag, og feila test kunne takast på nytt neste fredag – og kvar einskild student kunne velje sitt eige tempo oppover denne teststigen – dvs., ein trong ikkje møte opp til test kvar fredag, berre når ein sjølv hadde lyst (vel å merke så sant ein greidde å stå testar minst 5 av dei 14 vekene i semesteret, elles ville ein ikkje rekkje fram til lågaste ståkarakter). Sjølvvald tempo kan vere éi løysing på at studentar kjem til intro programmeringskurs med svært ulike forkunnskapar, men slett ikkje den einaste [10].

I tillegg til desse teljande testane som studentane kunne ta kvar fredag, fanst det også formative treningstestar. For å gjere opplegget så transparent som mogeleg, var desse treningstestane heilt identiske med dei teljande testane, dvs. begge typar test trekte oppgåver frå dei same spørsmålsbankane. For at dette skulle kunne fungere, var det naudsynt å ha så mange variantar av kvar oppgåve at det ikkje ville vere realistisk for studentar å setje seg ned og berre pugge svar på alle oppgåvene. Typisk var det rundt 20 variantar av kvar deloppgåve, og 8-10 deloppgåver i kvar prøve, dermed 160-200 unike spørsmål per nivå – og med 9 testnivå dermed over 1600 unike oppgåver om ein skulle heilt til A.

Kvalitetssikring av emne med tanke på gradvis betring av emnedesign og læringsressursar er viktig i høgare utdanning. NTNU sitt standard verkemiddel for dette er referansegrupper [11, 12]. Vi fekk mange konstruktive innspel frå referansegruppa, men meir på overordna aspekt ved emnedesignet. Det er vanskelegare for studentar å vurdere oppgåver med mange variantar, der kvar student berre har sett somme av variantane. Dermed trengst meir inngåande analyser i kvalitetssikring av slike læringsressursar. For summative prøver er det viktig med rettferd mellom oppgåvevariantar i form av tilstrekkeleg lik vanskegrad [13, 14]. I denne artikkelen ser vi spesifikt på nokre oppgåver frå G-testen i emnet IT1001, der hovudtema var if-setningar og logiske uttrykk. Dei raskaste studentane i klassen (om lag 20%) kom til denne testen i veke 3 av semesteret, fleirtalet i veke 4-5, og ein hale av studentar med litt langsamare framdrift i vekene deretter. Det er interessant å sjå på desse oppgåvene fordi if-setningar er eit essensielt og typisk tidleg tema i programmeringskurs, samstundes som det er vanskeleg nok til at ein del studentar slit med det. Spørsmåla våre er:

1) Kva slags analyse kunne faglærar bruke for å sjekke vanskegrad av oppgåver med tanke på kvalitetssikring av emnet?
2) Var desse spesifikke oppgåvene rettferdige med tanke på vanskegrad?
3) Om det var ulik vanskegrad, kva kan vere mogelege årsaker?

Resten av artikkelen er strukturert som følgjer: Seksjon 2 diskuterer relatert arbeid. Seksjon 3 viser dei analyserte oppgåvene og forklarer hypotesane vi vil teste. Seksjon 4 motiverer for valet av statistisk analysemetode, og seksjon 5 presenterer resultata. Til slutt kjem seksjon 6 med ein avsluttande diskusjon og konklusjon.

2 Relatert arbeid

Det finst fleire andre innleiande programmeringskurs på universitetsnivå som har nytta eit emnedesign basert på meistringslæring / PSI, både i Noreg [15] og elles i verda. Garner et al. [16] presenterer ei samanlikning av 12 slike undervisningsopplegg, og på andre er blitt publisert seinare, t.d. [17-21]. Mest likt vårt undervisningsopplegg er det som blei gjort av Toti et al. [21], der det også var ein serie testar som meir eller mindre direkte hang saman med karaktertrinn. Vårt opplegg var likevel monaleg meir transparent ved at dei teljande testane hadde korresponderande treningstestar som trekte oppgåver frå dei eksakt same spørsmålsbankane. Dette skapte behov for mange variantar av kvar oppgåve, medan andre opplegg for meistringslæring har greidd seg med kanskje 3-4 variantar av kvar test for å tilby gjentak til studentar som feila testen. Såleis er det ambisjonen om å oppnå høg transparens som fører til behovet for mange nesten like oppgåvevariantar. Dette skil seg frå mange andre autoretta prøver, til dømes den nasjonale forkunnskapstesten i programmering [22], der ideen tvert om er at alle prøvetakarane får same oppgåver.

Fleire andre har studert mogeleg ulikskap i vanskegrad mellom oppgåver, somme i samanheng med at ulike variantar blei brukte som middel for å redusere samarbeid ved ukontrollerte heimeeksamenar under Covid-19-pandemien. Oppgåvevariantar som alle testar same læringsutbytte og der koden i dei ulike variantane har identisk struktur (anten det er kode studenten skal forstå eller lage) blir gjerne kalla *isomorfe* i litteraturen. Denny et al. [14] analyserte om ei samling fleirvalsoppgåver for kodeforståing var rettferdige, der koden var den same i kvar variant, men svaralternativa varierte. Dei fann at dei fleste variantane hadde tilnærma lik vanskegrad, men somme med uvanlege svaralternativ skilde seg ut. Butler et al. [23] såg på isomorfe variantar av kodeskrivingsoppgåver i Java, og fann at mange var tilnærma like men somme skilde seg ut med høgare vanskegrad eller høgare tidsbruk. Fowler og Zilles [24] undersøkte diverse små variasjonar i kodeskrivingsoppgåver for å lage ein større spørsmålsbank – dels som mottiltak mot fusk. Endring av namn på variable og funksjonar hadde ingen effekt, medan endring av rekkjefølgje på parametrar og reversert polaritet (t.d. finne største vs. minste, positive vs. negative) somme gonger påverka vanskegraden. Emeka og Zilles [25] undersøkte ein eksamen der studentane eksplisitt hadde uttrykt tvil om rettferd for variantoppgåvene. Somme variantar viste seg å vere urettferdige, men i dei fleste høve kunne lågare testscore betre forklarast ved veikare kunnskap hos kandidaten. I ein påfølgjande studie analyserte Fowler et al. [13] rett-

ferd for ein serie av eksamenar som nytta isomorfe variantar av oppgåver og fann at dei fleste skilnader i vanskegrad var små nok til at det var akseptabelt, særleg om nokre få spørsmål som skilde seg ut, vart fjerna. Ein vidare studie av Fowler et al. [26] diskuterer ulike overflatepermutasjonar som kan brukast for å lage isomorfe oppgåver, og i kva grad dei påverkar vanskegraden. Parker et al. [27] såg på isomorfe blokkprogrammeringsoppgåver på ungdomsskulenivå, der elevane skulle velje rett kodealternativ for å flytte eit dyr til ønskt posisjon i eit rutenett. Somme variantar skilde seg ut med betre eller dårlegare score enn det som var vanleg, og forfattarane såg samsvar med normal leseretning og rekkjefølga på flytt og snu som plausible forklaringar på skilnadene. Liknande utfordringar knytt til leserekkjefølgje kan spele inn i våre oppgåvevariantar òg. Fromont et al. [28] undersøkte variantar av Parsons-problem der løysingskoden var den same, men der ulike kodeliner var gjort om til dra-element. Dei fann at variantar der vilkårssetningar var gjort om til dra-element (t.d. if, elif) var vanskelegare enn ein del andre variantar. Våre oppgåver handla også om if-setningar og den eine er eit Parsons-problem – men dei same kodelinene (både vilkårssetningar og handlingar) var dra-objekt i alle variantane, og variasjon måtte skapast på andre måtar, til dømes ved bruk av ulike logiske operatorar.

3 Oppgåver og hypotesar

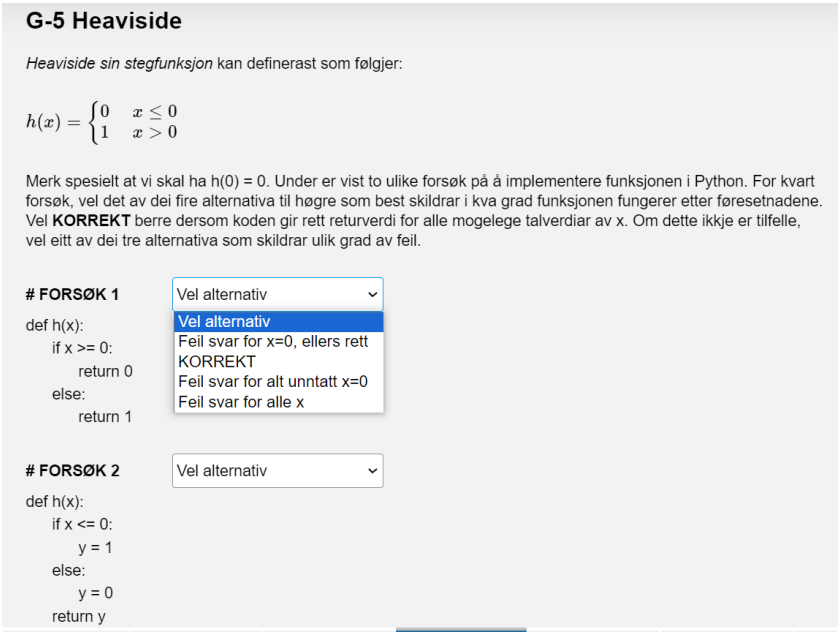
G-testen inneheldt 9 oppgåver, oversikt vist i tabell 1. Kolonnen Øving% gir gjennomsnittleg prosentvis score på kvar deloppgåve frå treningstestane studentane tok, medan Test% gir tilsvarande snittscore frå teljande testar, og Var.-kolonnane gir variansen for kvar av dei. Som ein kan sjå var dei fire første oppgåvene forholdsvis lette, alle med snittscore over 90% for teljande testar. Dette er enkle oppgåver, men konteksten her var meistringstestar der studentane måtte score over 90% for å stå – da må også snittscoren på dei fleste oppgåvene liggje rundt eller over 90, elles ville dei fleste ha stroke. Litt lågare score på treningstestar er naturleg sidan studentane i ein tidleg fase gjerne har dårlegare kunnskapar, men så får ei gradvis betring.

Tabell 1: Oppgåver i G-testen, analyserte med feit skrift

No.	Theme	Sjanger	Øving%	Var.	Test%	Var.
#1	Relasjonsuttrykk	Fleirval	91	3	95	1
#2	Strengindeksering	Paring	92	3	97	1
#3	Strengsamanlikning	Fleirval	89	6	96	2
#4	Konvertering bool	Paring	83	6	92	2
#5	**if…else**	**Fleirval**	**78**	**11**	**89**	**5**
#6	**if…elif…else**	**Plasser-i-tekst**	**84**	**8**	**92**	**3**
#7	and, or, not, ( )	Sant/Usant	92	4	95	1
#8	Nøsta if-else	Dra-og-slepp	64	14	75	8
#9	f-strengar	Fleirval (i kode)	84	5	93	2

Vi valde å analysere oppgåve #5 og #6 fordi desse hadde noko høgare varians enn dei tidlegare oppgåvene, og dessutan fokuserte på forståing av kodesnuttar som inkluderte if-setningar, medan tidlegare oppgåver såg på berre éi kodeline av gongen. Oppgåve #8 kunne også vere interessant å analysere, gitt at den hadde enda høgare vanskegrad og varians enn #5 og #6, men tre oppgåver ville blitt for plasskrevjande for denne artikkelen.

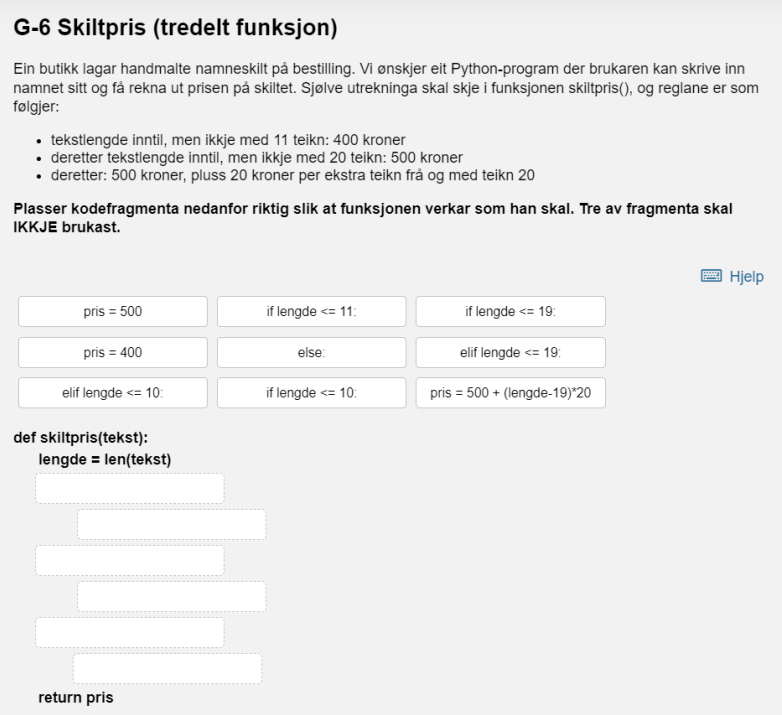
Figur 1 viser ein tilfeldig trekt variant av oppgåve 5 slik den såg ut i Inspera. Teksten forklarer ein matematisk stegfunksjon og viser to Python-funksjonar som er forsøk på å implementere funksjonen. Det var 21 ulike variantar. Python-koden var strukturelt lik i alle dei 21 variantane, der første forsøk alltid er ein if-else som returnerer direkte i if-setninga, og andre forsøk alltid bruker ein variabel y som blir returnert etter if-setninga. Det som endrar seg mellom variantane er kva operator som blir brukt i if-setninga, kva tal det blir testa mot i if-setninga (0 eller 1) og rekkjefølgja på kva som blir returnert i if- og else-delen av koden (0 … 1 eller 1 … 0). Somme av desse kombinasjonane vil gi funksjonar som oppfører seg korrekt, medan andre vil gi ulike slags feil. Som ein kan sjå av figuren, har studenten valet mellom fire ulike alternativ for korleis koden verkar: (i) korrekt returverdi for alle x, (ii) feil returverdi for alle x, (iii) korrekt returverdi for x = 0, feil for andre x eller (iv) korrekt returverdi for alle x unnateke x = 0. Dei to forsøka var sett saman slik at det aldri var same av dei fire alternativa som var korrekt for begge delane av oppgåva.



Figur 1. Tilfeldig trekt variant av oppgåve 5.

Figur 2 viser ein tilfeldig trekt variant av oppgåve 6, som gjorde bruk av Inspera-sjangeren «Plasser i tekst». Studentane skal her trekkje kodeliner til rett posisjon, slik

at funksjonen skiltpris() fungerer slik teksten forklarer. Det er 6 ledige plassar som må fyllast i koden, men 9 kodeliner tilgjengeleg – der 3 dermed er distraktorar som ikkje skal brukast. Løysingskoden er strukturelt lik i alle variantane, i form av ei if-elif-else-setning med tre mogelege utfall for pris av skilt, men det varierer mellom variantane kva som er gitt som grenseverdi mellom intervalla, og om denne varianten var oppgitt som til og med, eller til men ikkje med. Dessutan varierer det kva relasjonsoperator som var brukt i if- og elif-testane. Det var 16 variantar av oppgåva, med 4 variantar for kvar av relasjonsoperatorane <, <=, >=, >. Dette vil også avgjere retning på returverdiane, sidan variantar med < eller <= må starte med minste pris i if-delen og ende med høgaste pris i else-delen, medan > eller >= må gi motsett rekkjefølgje.



Figur 2. Tilfeldig trekt variant av oppgåve 6

Nullhypotesar for både oppgåve 5 og 6 vil vere at det ikkje er signifikante skilnader på prestasjonar for ulike variantar av oppgåvene. Faglæraren sitt utgangspunkt var også å prøve å lage variantar som hadde same vanskegrad, men samstundes med nok variasjon til at studentar ikkje berre kunne pugge svar. Om ein ser på oppgåvene med eit kritisk blikk, er det likevel mogeleg å tenkje seg faktorar som påverke vanskegraden. For oppgåve 5 er den matematiske definisjonen av stegfunksjonen alltid identisk, medan det er koden i dei to forsøka som varierer. Den matematiske definisjonen bruker alltid operatoren $\leq$ som vilkår for å returnere 0, og > for å returnere 1. Ein kan dermed mistenkje at forsøk på Python-kode som nyttar anten <= eller > i if-setninga, vil vere meir like den matematiske definisjonen enn forsøk som nyttar < eller >=. Eit

ytterlegare moment i dette er at eit forsøk med <= eller > anten vil returnere korrekt verdi for alle x, eller feil verdi for alle x, medan forsøk med < eller >= derimot vil hamne på svaralternativa «Feil svar for alt unntatt x=0» eller «Feil svar for x=0, elles korrekt». Desse to mellomalternativa er kanskje konseptuelt meir forvirrande enn dei reine alternativa heilt rett eller heilt feil. Ei plausibel alternativ hypotese for oppgåve 5 blir dermed *H1: oppgåvevariantar som bruker <= eller >, med svar korrekt eller feil for alle x, vil ha betre score enn variantar som bruker < eller >=.*

Sidan den matematiske definisjonen har returverdi 0 øvst, så 1, kan ein også tenkje seg at variantar der koden har 0 i if-delen og 1 i else-delen, kan vere enklare å vurdere enn kode der det er omvend. Altså *H2: oppgåvevariantar med resultat 0 på if og 1 på else, har betre score enn oppgåvevariantar med 1 på if og 0 på else.*

For oppgåve 6 er det største logiske skiljet mellom variantar at halvparten av variantane tar prisintervalla i stigande rekkjefølje i if-elif-else-setninga (dersom operator var < eller <=), og den andre halvparten tar dei i søkkande rekkjefølgje (dersom operator var > eller >=). I og for seg treng kanskje ikkje det eine vere lettare enn det andre, men ein kan merkje seg at oppgåveteksten for alle variantar hadde tre kulepunkt som tok for seg prisintervalla i stigande rekkjefølgje. Dermed vil variantar der løysingskoden har stigande rekkjefølgje, ha betre samsvar med leserekkjefølgja i oppgåveteksten. Ei plausibel alternativ hypotese for oppgåve 6 blir dermed *H3: oppgåvevariantar med stigande rekkjefølgje på prisintervalla har betre score enn oppgåvevariantar med søkkande rekkefølgje.*

4 Metode for statistisk analyse

Rimelegvis fins det også andre metodar som kunne ha vore brukt for å undersøkje relativ vanskegrad av oppgåvevariantar, til dømes intervju med studentar om korleis dei opplevde vanskegraden, eller også observasjon av studentar under løysing av oppgåver, men dette ville vere langt meir krevjande, og gå langt forbi det ein faglærar vanlegvis har tid til å kvalitetsoppfølgjing av eit emne frå eitt år til det neste. Ein naturleg start vil vere å først sjå på enkel statistikk for oppgåvevariantar om somme skil seg ut som lettare eller vanskelegare, og bruke dette som utgangspunkt for å vurdere om somme oppgåver bør reviderast. Ein vanleg test å bruke for å finne om ei gruppe (her: av oppgåvevariantar) har betre score enn ei anna, er ANOVA, men denne har som føresetnad at data er normalfordelte. Dette var langt ifrå tilfelle for score på oppgåvene våre – og normalfordeling var neppe å forvente sidan det var snakk om meistringstestar der studentane måtte score over 90%, som gjorde at det var ein topp på full score på begge oppgåvene. Med unormale data blir Kjikvadrattest eit mogeleg alternativ for oppgåver som berre har mogeleg score 0 eller 1 (som er tilfelle for kvart av dei to fleirvalsspørsmåla i Oppgåve 5, og Kruskal-Wallis blir ein aktuell test når det finst fleire enn to utfall, som for oppgåve 6 der det var mogeleg å score alt frå 0 til 6 poeng. Dette er same bruk av statistiske metodar som i Parker et al. [27], som på liknande vis samanlikna variantar av oppgåver der somme var fleirval med score 0, 1 og somme var Parsons problem der det var mogeleg å score fleire poeng.

Sjølve testinga blei gjort ved at prestasjonsdata på dei to oppgåvene vart lasta ned frå Inspera til eit anonymt datasett utan noko informasjon knytt til student. Data blei analysert i Excel, med formelen KJIKVADRAT for fleirvalsspørsmåla og RANG.GJSN pluss innskrivne formlar for Kruskal-Wallis-testane av oppgåve 6. Sidan formative treningstestar og summative teljande testar føregjekk under svært ulike forhold (formative: ingen kontroll, mogeleg å bruke hjelpemiddel eller diskutere med medstudentar; summative: under tilsyn med bruk av SEB) blir desse testa kvar for seg heller enn å vere samla i eitt datasett.

Det er viktig å merke seg at samanlikninga av oppgåvevariantar her *ikkje* kan seiast å vere noko kontrollert eksperiment. Den ideelle måten for å samanlikne vanskegrad mellom variantar av oppgåver ville vere å la to eller fleire jamgode grupper av studentar løyse variantane i ein kontrollert situasjon, på eitt fast tidspunkt. Det vi har av svardata er derimot samla over eit lenger tidsrom. Særleg for dei formative øvingstestane kan studentane ha løyst oppgåvene på fleire ulike måtar, med eller utan hjelpemiddel, og med eller utan hjelp frå medstudentar eller læringsassistentar, og studentar kan ha gjort mange forsøk på testen, der ein ville vente at svara blir gradvis betre. Sidan data er fullstendig anonymiserte, er det umogeleg for oss å sjekke om eit gitt svar er ein student sitt første eller n'te forsøk på oppgåva, eller kva vedkomande elles presterte i emnet. Det er også umogeleg å vite om ein student somme tider har hatt «flaks» og fått identisk variant på teljande test som ein såg på ein tidlegare øvingstest. Samstundes er det verd å merke seg at trekkinga av variantar var fullstendig tilfeldig, så alle variantar hadde lik sjanse for å bli gitt til både ambisiøse og mindre ambisiøse studentar, og alle variantar hadde lik sjanse til å bli gitt som første forsøk eller som n'te forsøk på ei oppgåve. At nokon variantar kan ha vore «heldige» og fått flinkare studentar er såleis ikkje særleg annleis enn at det tilfeldigvis kan førekome at ei gruppe tilfeldigvis får flinkare studentar enn den andre i eksperiment der ein fordeler forsøkspersonar tilfeldig i to grupper.

5 Resultat

Tabell 2 viser resultat for analysen av oppgåve 5. Venstre del av tabellen viser resultat for dei formative treningstestane medan høgre del viser for dei summative testane under tilsyn. Dei to øvste radene er ikkje knytt til noka hypotese men samanliknar score på "Forsøk 1" der koden hadde return-setningar rett i if- og else-delen og «Forsøk 2», der det blei brukt ein lokal variabel y som blei returnert etter if-setninga. Mellom desse var det ingen signifikant skilnad. Det ville heller ikkje ha vore noka urettvise om det var ein skilnad, sidan alle studentar fekk éin av kvar kodesnutt.

Dei to radene med lyseblå bakgrunn viser samanlikninga knytt til H1. Som ein kunne mistenkje, var det betre score for variantar som brukte same relasjonsoperatorar som den matematiske definisjonen, og skilnaden var sterkt signifikant både for formative og summative prøver. For rekkjefølgja på returverdi 0 og 1 i koden (knytt til H2) var det derimot ingen signifikant skilnad.

Tabell 2: Resultat for oppgåve 5

Oppgåvevariant	Trening (N=290)	χ^2	p	Teljande (N=58)	χ^2	p
"Forsøk 1", alle variantar	0.78	--	--	0.88	--	--
"Forsøk 2", alle variantar	0.79	--	--	0.90	--	--
Variantar med <= eller >	**0.89**	**26.8**	**0.00001**	**0.97**	**7.37**	**0.007**
Variantar med < eller >=	**0.67**			**0.81**		
Variantar med 0 før 1	0.80			0.90		
Variantar med 1 før 0	0.77			0.88		

I tillegg til å sjå på skilnad i poengscore, kan det også vere interessant å undersøkje kva feil som typisk blei gjort på dei ulike oppgåvevariantane. Tabell 3 indikerer at det mest vanlege (som markert i grått) var at det blei svart «Feil for alle x» der det riktige ville ha vore «Feil svar for alt unntatt x=0», og dernest at det blei svart «Feil svar for alt unntatt x=0» der det riktige var «Feil svar for x=0, elles rett». På treningstestane var det også ein del høve der det blei svara «Korrekt for alle x» når dette ikkje var rett, jamfør tala 13, 11, 5 i venstre kolonne i tabellen, men denne typen feil forsvann tydelegvis i løpet av treninga, da ingen slike feil blei gjort på dei teljande testane.

Tabell 3: Fordeling av feilsvar for oppgåve 5

N = (290, 58)	Svar gitt			
Løysing	A	B	C	D
A) Korrekt svar for alle x	--	(5, 0)	(6, 0)	(5, 1)
B) Feil svar for x=0, elles rett	**(13, 0)**	--	**(18, 3)**	(9, 2)
C) Feil svar for alt unntatt x=0	(11, 0)	(2, 0)	--	**(33, 6)**
D) Feil svar for alle x	(5, 0)	(9, 0)	(9, 1)	--

Tabell 4 viser resultat for Oppgåve 6, igjen med kolonnene i venstre halvdel for formative øvingstestar, medan kolonner i høgre halvdel er for summative prøver under tilsyn. Maksimalscore på oppgåva var 6 poeng (alle seks kodeliner plassert riktig), og snittscore for alle prøveforsøk var 5.13 for øvingstestane, 5.50 for dei teljande under tilsyn. Som vi kan sjå, viste Kruskal-Wallis-testen signifikant skilnad mellom variantar avhengig av kva operator som var brukt i if- og elif-setningane. Variantar med < og <=, der prisintervalla kom i stigande rekkjefølgje, hadde monaleg betre score enn snittet på 5.13, medan variantar med prisintervalla i søkkande rekkjefølgje hadde monaleg dårlegare score. Denne skilnaden var signifikant for dei formative øvingstestane, men ikkje for dei teljande testane. Radene nedst i tabellen samanliknar score for dei andre variasjonsaspekta i oppgåva: om terskelverdien var inkludert eller ikkje (til og med, vs. til men ikkje med), og om talet 10 eller 11 var brukt for terskelverdien. Her var det berre små skilnader, langt under det som kunne vere signifikant.

Ein meir inngåande analyse av kva feil som førte til poengtap på oppgåve 6, illustrerer også at stigande rekkjefølgje synest å ha vore lettare. Talet på prøvesvar som feilaktig sette saman if-elif-else-setninga i søkkande rekkjefølgje der fasit skulle vere

stigande, var berre 6 på øvingstestane og 1 på teljande test. Feilaktig bruk av stigande rekkjefølgje der fasit skulle ha vore søkkande var derimot heile 31 på øvingstestar, 5 på teljande test. Med andre ord var det om lag 5 gonger så vanleg å ha invertert rekkjefølgje i forhold til fasit der fasit skulle ha vore søkkande. I tillegg var det sjølvsagt ein del andre feil, mellom anna relatert til vald terskelverdi, men desse feila hadde ikkje tilsvarande nokon markant tendens for spesifikke variantar av oppgåva.

Tabell 4: Resultat for Oppgåve 6 (if-elif-else, max score 6 poeng)

Variantgruppe	Øving (N=290)	H	p	Prøve (N=58)	H	p
Alle	5.13	--	--	5.50	--	--
Stigande, <	**5.51**	**17.8**	**0.0005**	5.58	1.99	0.58
Stigande, <=	**5.64**			6.00		
Søkkande, >=	**4.68**			5.20		
Søkkande, >	**4.58**			5.36		
Inkl. terskel	5.13			5.64		
Ekskl. terskel	5.14			5.39		
Terskel 10	5.25			5.50		
Terskel 11	5.03			5.50		

6 Diskusjon

For dei tre alternative hypotesane vi hadde vart resultata som følgjer:

- H1 vart akseptert, da det var signifikant skilnad mellom variantar av oppgåve 5. Variantar der Python-koden hadde operatorar <= eller >, i tråd med den matematiske funksjonen, hadde betre score enn variantar med dei motsette operatorane < eller >=. Denne skilnaden var til stades både for formative treningstestar og for dei teljande testane gjort under tilsyn.
- H2 vart derimot ikkje akseptert, da det ikkje var nokon signifikant skilnad mellom variantar som returnerte 0 på if, 1 på else, og variantar som hadde den motsette rekkjefølgja.
- H3 vart akseptert da det var signifikant skilnad på dei formative treningstestane: Variantar der if-elif-else gjekk i stigande rekkjefølgje gjennom prisintervalla, hadde betre score enn variantar der rekkjefølgja var søkkande. For dei teljande testane var skilnaden ikkje lenger signifikant.

For H1 og H2 tyder dette på at samsvar mellom kode og matematisk definisjon gjorde det lettare for studentane å forstå om koden var rett eller ikkje, og at samsvar i bruk av relasjonsoperatorar var meir nyttig enn samsvar i rekkjefølgje på retur av 0 og 1. Med andre ord, studentane greidde lettare å omstille seg til ei motsett rekkjefølgje versus den matematiske definisjonen enn til bruk av andre operatorar. Eit medverkande element her kan også vere at variantar som brukte same operatorar som den matematiske definisjonen, ville leie til eitt av dei to «reine» svara: at funksjonen returnerte korrekt verdi for alle x, eller feil verdi for alle x, medan motsette operatorar ville leie til blandingsalternativ der utfallet for akkurat x=0 var annleis enn for andre x-verdiar.

For H3 kan det vere fleire årsaker til at stigande rekkjefølgje var enklare:

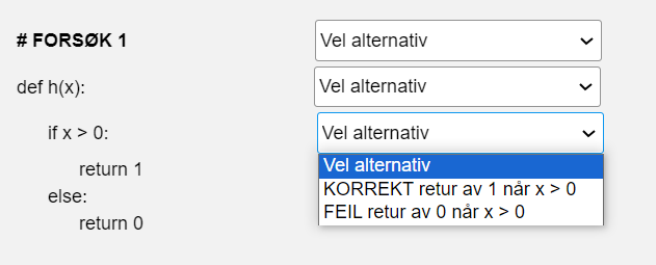
- Stigande rekkjefølgje var også brukt for kulepunktlista i oppgåveteksten.
- Kanskje er stigande rekkjefølgje meir vanleg i dagleglivet når slike prisintervall skal skildrast?

At skilnaden ikkje lenger var signifikant for dei teljande testane, kan tyde på at studentane gradvis greidde å lære seg at rekkjefølgja måtte hengje saman med kva relasjonsoperator som vart brukt – som nettopp var hovudpoenget oppgåva håpa å få fram. Samstundes kan manglande signifikans for teljande testar også skuldast at N var mindre her (58. vs. 216) slik at det trongst mykje større skilnad for å gi signifikans.

Som diskutert i seksjon 4 har vi avgrensa kontroll over data fordi dei er fullstendig anonyme, det er umogeleg å vite om nokon variantar har vore «heldigare» ved å bli tatt av flinkare studentar, og vi veit heller ikkje kva forsøk som kjem frå same student. Ein vil forvente betre score når studentar får eksakt same oppgåve som dei har sett før [26]. Gitt at variantar blei trekte heilt tilfeldig skal det likevel mykje til at dette kan ha vore avgjerande, særleg sidan vi samanliknar oppgåvevariantar, ikkje studentar. Kvar variant vil ha hatt lik sjansar til å vere «heldig» med å få same studentar om att. Dei største skilnadene er sterkt signifikante, særleg for øvingstestane der talet på svar var større ($p=0.00001$, $p=0.0005$). Det verkar derfor rimeleg å konkludere at det var ulik vanskegrad på variantar både for oppgåve 5 og 6, og for begge oppgåvene synest skilnaden å vere knytt til bruken av relasjonsoperatorar. For oppgåve 5 gav visse operatorar betre samsvar med den matematiske definisjonen av funksjonen gitt i oppgåveteksten, og leidde dessutan til «reinare» svaralternativ («korrekt for alle x», «feil for alle x»), medan dei motsette operatorane leidde til alternativ der funksjonen var delvis korrekt. For oppgåve 6 gav visse operatorar stigande rekkjefølgje som var i samsvar med kulepunkta i oppgåveteksten, medan motsette operatorar gav søkkande rekkjefølgje som kanskje dermed vart lettare for studentane å feile på.

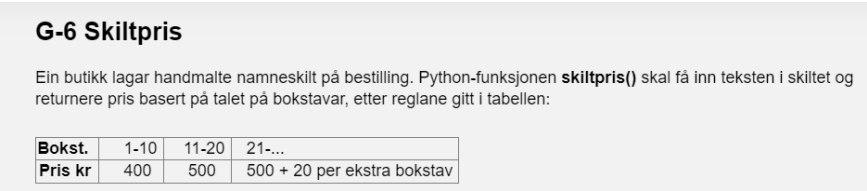
Potensiell urettferd i variantar er ikkje ønskeleg, så eit interessant spørsmål er da korleis ein eventuelt kan redusere dette ved å endre oppgåvene. Oppgåve 5 kunne ved ganske enkle grep ha vore meir rettferdig sidan den alltid inneheldt to spørsmål, eitt relatert til «Forsøk 1» og eitt relatert til «Forsøk 2». Gitt at operatorar `<=` og `>` er enklare og `<` og `>=` vanskelegare, kunne ei mogeleg løysing ha vore at éi av oppgåvene alltid brukte ein av dei enkle operatorane og den andre ein av dei vanskelegare – men med variasjon om dette var Forsøk 1 eller 2, så det ikkje skulle bli for openbert. Dette var dessverre ikkje gjort i 2023 – einaste avgrensing var at ingen av variantane hadde same svar for Forsøk 1 og 2, men somme variantar kunne da ha to enkle («rett for alle x», «feil for alle x»), somme kunne ha ein lett og ein vanskeleg, og somme to vanskelege. Ein annan refleksjon rundt oppgåve 5 er at fleirval med 4 alternativ, der to alternativ er konseptuelt «reinare» enn to andre, var uheldig. Ein betre måte å gje oppgåva på kan vere med fleire spørsmål med 2 alternativ på kvart. Ei slik endring av spørsmålet er vist i Figur 3. Her er den nedste «Vel alternativ»-boksen opna, denne gjeld når $x > 0$ og gir studenten 2 alternativ: «FEIL retur av 0 når $x > 0$» og «KORREKT retur av 1 når $x > 0$». Desse to skulle vere mykje likare når det gjeld kompleksitet enn dei gamle alternativa, t.d. «Feil svar for alle x» vs. «Feil svar for x=0, elles rett». Dei to «Vel alternativ»-boksane ovanfor gir likeins tilsvarande alternativ for $x < 0$ og $x = 0$. Dermed må ein alltid ta stilling til 3 spørsmål, uansett kva kodevarianten er, og ingen svaralternativ skal vere kortare enn andre. Vonleg vil dette redusere urettferd mellom

variantar av oppgåve 5. Den nye måten er også meir pedagogisk å spørje på med tanke på å lære studentane korleis ein bør tenkje ved testing eller inspeksjon av kode med if-setningar, nemleg dekkje alle mogelege utfall og sjå spesielt på grensetilfella.



Figur 3. Ny versjon av «Forsøk 1» i oppgåve 5.

For oppgåve 6 var ein potensiell årsak til urettferd at kulepunkta i oppgåveteksten forklarte prisintervall i stigande rekkjefølgje, som kan ha gitt fordel til variantar med same rekkjefølgje i if-elif-else-setninga, og ulempe til motsett rekkjefølgje. Figur 4 viser ein ny versjon som har bytt ut kulepunktlista med ein tabell for prisintervalla. Her viser vi berre spørsmålsteksten sidan interaksjonsdelen av oppgåva med kodeliner som skal trekkjast på plass, er den same som før, jfr. fig. 2. Tabellrepresentasjonen har fleire fordelar: Oppgåveteksten blir langt meir konsis, og det blir tydelegare eksakt kva tal på bokstavar som skal leie til kva prisintervall, som kunne vere meir krevjannde å forstå i ein lengre tekst med fraser som «inntil men ikkje med» eller «til og med». Dessutan er samsvaret mellom tekst og variantar med stigande if-elif-else iallfall noko svekka, sidan det ikkje lenger er ei loddrett liste av kulepunkt som stemmer perfekt med ei loddrett liste av aksjonar på if, elif og else. Det *kan* likevel tenkjast at stigande rekkjefølgje framleis vil ha ein fordel med denne versjonen også – sidan naturleg leseretning for tabellen nok vil vere frå venstre mot høgre for dei fleste.



Figur 4. Ny versjon av oppgåve 6.

Ein lærdom å trekkje frå dette er at jamvel om faglærar prøver å lage variantar som er så rettferdige som mogeleg, så er dette utfordrande sidan jamvel ganske subtile skilnader kan påverke vanskegraden i oppgåver. Nettopp derfor er det viktig å analysere spørsmålsbankane år for år på jakt etter oppgåver som kan bli betre. Dette gjeld sjølvsagt ikkje berre i spørsmål om rettferd, men også andre aspekt ved oppgåvene, til dømes å gjere oppgåveformuleringane så presise som mogeleg, og undersøkje om oppgåvene testar dei læringsutbytta som vi er interesserte i.

Referansar

1. Sanders, K., Ahmadzadeh, M., Clear, T., Edwards, S.H., Goldweber, M., Johnson, C., Lister, R., McCartney, R., Patitsas, E., Spacco, J.: The Canterbury QuestionBank: Building a repository of multiple-choice CS1 and CS2 questions. Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports, pp. 33-52 (2013)
2. Jeffries, B., Baldwin, T., Zalk, M.: Online Examinations in a Large Australian CS1 Course. Proceedings of the 24th Australasian Computing Education Conference, pp. 20-26 (2022)
3. Rusak, G., Yan, L.: Unique exams: designing assessments for integrity and fairness. Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, pp. 1170-1176 (2021)
4. Sindre, G., Haugset, B.: Techniques for detecting and deterring cheating in home exams in programming. 2022 IEEE Frontiers in Education Conference (FIE), pp. 1-8. IEEE (2022)
5. Emeka, C.A., Zilles, C., West, M., Herman, G.L., Bretl, T.: Second-Chance Testing as a Means of Reducing Students' Test Anxiety and Improving Outcomes. 2023 ASEE Annual Conference & Exposition, (2023)
6. Beerepoot, M.T.: Formative and Summative Automated Assessment with Multiple-Choice Question Banks. Journal of Chemical Education 100, 2947-2955 (2023)
7. Zamprogno, L., Holmes, R., Baniassad, E.: Nudging student learning strategies using formative feedback in automatically graded assessments. Proceedings of the 2020 ACM SIGPLAN Symposium on Splash-E, pp. 1-11 (2020)
8. Keller, F.S., Sherman, J.G.: PSI, the Keller Plan Handbook: Essays on a personalized system of instruction. WA Benjamin Advanced Book Program (1974)
9. Eyre, H.L.: Keller's Personalized System of Instruction: Was it a Fleeting Fancy or is there a Revival on the Horizon? The Behavior Analyst Today 8, 317 (2007)
10. Sandstrak, G., Klefstad, B., Styve, A., Raja, K.: Analyzing Pedagogic Practice and Assessments in a Cross-Campus Programming Course. IEEE Transactions on Education (2024)
11. Sandstrak, G., Klefstad, B.: Referansegrupper som verktøy for kontinuerlig kvalitetsforbedring. Norsk IKT-konferanse for forskning og utdanning, (2021)
12. Klefstad, B., Sandstrak, G.: Hvordan skape økt bevissthet og oppslutning fra studentene i kvalitetsutvikling i høyere utdanning. Norsk IKT-konferanse (NIKT), (2023)
13. Fowler, M., Smith IV, D.H., Emeka, C., West, M., Zilles, C.: Are We Fair? Quantifying Score Impacts of Computer Science Exams with Randomized Question Pools. Proceedings of the 53rd ACM Technical Symposium on Computer Science Education, pp. 647-653 (2022)
14. Denny, P., Manoharan, S., Speidel, U., Russello, G., Chang, A.: On the fairness of multiple-variant multiple-choice examinations. Proceedings of the 50th ACM technical symposium on computer science education, pp. 462-468 (2019)
15. Purao, S., Sein, M., Nilsen, H., Larsen, E.Å.: Setting the Pace: Experiments With Keller's PSI. IEEE Transactions on Education 60, 97-104 (2017)

16. Garner, J., Denny, P., Luxton-Reilly, A.: Mastery learning in computer science education. Proceedings of the Twenty-First Australasian Computing Education Conference, pp. 37-46 (2019)
17. Campbell, J., Petersen, A., Smith, J.: Self-paced mastery learning CS1. Proceedings of the 50th ACM Technical Symposium on Computer Science Education, pp. 955-961 (2019)
18. Izu, M., Ng, D., Weerasinghe, H.: Mastery Learning and Productive Failure: Examining Constructivist Approaches to teach CS1. (2023)
19. Aggarwal, A., Puthanveetil, N., Gardner-Mccune, C.: Who Attempts Optional Practice Problems in a CS1 Course? Exploring Learner Agency to Foster Mastery Learning. Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, pp. 1055-1061 (2023)
20. Ott, C., McCane, B., Meek, N.: Mastery learning in cs1-an invitation to procrastinate?: Reflecting on six years of mastery learning. Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1, pp. 18-24 (2021)
21. Toti, G., Chen, G., Gonzalez, S.: Teaching CS1 with a Mastery Learning Framework: Impact on Students' Learning and Engagement. Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, pp. 540-546 (2023)
22. Bolland, S.: National Prior Knowledge Test in Programming-How proficient are incoming higher education students? Norsk IKT-konferanse for forskning og utdanning, (2023)
23. Butler, L., Challen, G., Xie, T.: Data-Driven Investigation into Variants of Code Writing Questions. 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), pp. 1-10 (2020)
24. Fowler, M., Zilles, C.: Superficial Code-guise: Investigating the Impact of Surface Feature Changes on Students' Programming Question Scores. Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, pp. 3-9 (2021)
25. Emeka, C., Zilles, C.: Student perceptions of fairness and security in a versioned programming exam. Proceedings of the 2020 ACM conference on international computing education research, pp. 25-35 (2020)
26. Fowler, M., Smith, D.H., Zilles, C.: Quickly Producing "Isomorphic" Exercises: Quantifying the Impact of Programming Question Permutations. Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, pp. 178–184. Association for Computing Machinery, Milan, Italy (2024)
27. Parker, M.C., Garcia, L., Kao, Y.S., Franklin, D., Krause, S., Warschauer, M.: A Pair of ACES: An Analysis of Isomorphic Questions on an Elementary Computing Assessment. Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1, pp. 2-14 (2022)
28. Fromont, F., Jayamanne, H., Denny, P.: Exploring the Difficulty of Faded Parsons Problems for Programming Education. Proceedings of the 25th Australasian Computing Education Conference, pp. 113-122 (2023)