TARTU ÜLIKOOL
LOODUS- JA TÄPPISTEADUSTE VALDKOND
Arvutiteaduse instituut
Informaatika õppekava

Helen Hendrikson

MOOCi „Programmeerimise alused“ ülesannete lahenduste analüüs

Bakalaureusetöö (9 EAP)

Juhendaja: Marina Lepp, PhD

Tartu 2018

MOOCi „Programmeerimise alused“ ülesannete lahenduste analüüs

Lühikokkuvõte:

Töös analüüsiti vaba juurdepääsuga e-kursuse „Programmeerimise alused” ülesannete lahendusi. Antud uurimus on koostatud, et näha, milliseid vigu osalejad teevad ja kuidas ülesandeid lahendavad, kuna esitatud ülesannete lahendusi hinnatakse automaakontrolli abil ning puudub ülevaade nii vigadest kui ka lahendustest. Töö eesmärgiks oli välja selgitada tingimuslause ja tsükli teemade keerulisemate ülesannete vead ja erinevad lahendused ning selle põhjal anda soovitusi materjalide täiendamiseks. Töös leiti vastused järgmistele uurimisküsimustele: milliseid lahendusi esines analüüsitud ülesannete lahenduste hulgas, milliseid vigu tehti analüüsitud ülesannete lahendamisel ning milliseid vigu esines analüüsitud lahendustes kõige rohkem.

Võtmesõnad:

MOOC, programmeerimise ülesanded, programmeerimise vead, ülesannete lahendamine

CERCS: P175, S280

The analysis of MOOC “Introduction to programming” exercises solutions

Abstract:

This paper describes the analysis of massive open online course “Introduction to programming” exercises solutions. As all the solutions are assessed automatically then the course supervisors are not aware of the solutions and mistakes made by the course participants. The purpose of this thesis was to figure out the mistakes and possible solutions of conditional statements and loop exercises and give helpful feedback of common problems for the supervisors to make adjustments as needed for the future courses. As a result of this analysis the following research questions were answered: which kind of solutions appeared amongst the solutions, which mistakes were made during problem solving and which mistakes were most common.

Keywords:

MOOC, programming assignments, programming mistakes, problem solving

CERCS: P175, S280

Sisukord

Sissejuhatus

MOOC (ingl *Massive Open Online Course*) on suuremahuline vaba ligipääsuga veebipõhine tasuta kursus, mis on praegu maailmas üks uuenduslikumaid arengusuundi kõrghariduses ning mille kiire areng sai alguse 2011. aastal [1].

Tartu Ülikoolis toimus esimene programmeerimise MOOC aastal 2014, mille nimeks sai „Programmeerimisest maalähedaselt" ning mis osutus ootamatult populaarseks. Kursusele oli jõudnud registreeruda 650 inimest, kui registreerumine otsustati liiga suure osaluse kartusel sulgeda [2]. 2016. aastal toimus teine MOOC, milleks oli „Programmeerimise alused".

Jälgides viimaste aastate statistilisi andmeid Tartu Ülikoolis pakutavate IT-alaste MOOCide kohta [3], võib öelda, et inimeste huvi infotehnoloogia ja programmeerimise vastu pole kahanenud. 2017. aasta sügissemestril toimunud kursuse „Programmeerimise alused" osalejate arv küündis 1518 huviliseni ning populaarsuselt esimese koha saavutas samal aastal kursus „Programmeerimisest maalähedaselt", kus osales 2247 inimest. Huviliste vanus antud kursustel jääb vahemikku 8 kuni 76 eluaastat [4]. Sellest tulenevalt võib eeldada, et inimeste ülesannetele lähenemine ja ülesannete lahendused võivad varieeruda väga suures ulatuses. Kuna aga kõik ülesanded hinnatakse automaatkontrolli abil, siis ei ole teada, milliseid vigu kursusel osalejad tegelikult teevad ning kuidas nad ülesandeid lahendavad. Sellepärast valis autor oma bakalaureusetööks just antud teema.

Töös analüüsitakse Tartu Ülikooli arvutiteaduse instituudi MOOCi „Programmeerimise alused" kahe kohustusliku ülesande lahendusi. Töö eesmärgiks on välja selgitada tingimuslause ja tsükli teemade keerulisemate ülesannete vead ja erinevad lahendused ning selle põhjal anda soovitusi vastavate teemade materjalide ja ülesannete tekstide täiendamiseks, et vähendada edaspidi osalejate muresid ja küsimusi. Eesmärgist lähtuvalt esitas autor järgmised uurimisküsimused:

- Milliseid lahendusi esines analüüsitud ülesannete lahenduste hulgas?
- Milliseid vigu tehti analüüsitud ülesannete lahendamisel?
- Milliseid vigu esines analüüsitud lahendustes kõige rohkem?

Töös analüüsitud kursuse „Programmeerimise alused" kestus on kaheksa nädalat ja kursus on mahus 3 EAP ehk 78 tundi iseseisvat tööd. Kursuse sihtrühmaks on õppijad, kellel kokkupuude programmeerimisega puudub või on vähene. Edukaks läbimiseks tuli osalejatel iga nädal lahendada üks kuni neli kohustuslikku ülesannet hindele arvestatud ja sooritada

kõik *Moodle'*i testid hindele arvestatud. Kursus on mitteeristava hindamisega ning kõiki ülesandeid hinnati automaatkontrolli abil. Esimene analüüsitud ülesanne kuulub teise nädala teema „Tingimuslaused" hulka ning teine analüüsitud ülesanne kuulub kolmanda nädala teema „Tsüklid" hulka. Mõlemad ülesanded olid kohustuslikud kursuse läbimiseks.

Töö esimeses osas annab autor ülevaate MOOCidest ja algajate programmeerijate vigadest ning neid mõjutavatest teguritest. Teises osas on kirjeldatud töö metoodikat. Lõpuks analüüsib autor kursuse „Programmeerimise alused" kahe valitud ülesande lahendusi ning esitab analüüsi tulemused.

1. Ülevaade MOOCidest

Peatükis antakse ülevaade MOOCide olemusest, Tartu Ülikooli programmeerimise MOOCidest ja MOOCist „Programmeerimise alused“.

1.1 MOOCide olemus

MOOC on haridusalane veebipõhine suuremahuline avatud kursus, mis on kõigile tasuta [1]. Avatud kursus tähendab, et osaleda saavad kõik, kellel on ligipääs internetile. MOOCides keskendutakse peamiselt algteadmiste ja teooria õpetamisele [5].

MOOC on tavaliselt üles ehitatud nädalate kaupa. Materjalid on esitatud erinevate dokumentide, videote ja viktoriinide abil keskkonnas, mida osalejad saavad külastada neile sobivatel aegadel. Nii osalejate omavahelise kui ka juhendajate ja osalejate vahelise suhtluse peamine vahend on foorum. Kuid siiski tuleks ära märkida, et MOOC on kursus, kus õppejõudude toetus on minimaalne.

Mitme allika põhjal saab väita, et MOOCide populaarsus hakkas kiiresti kasvama alates 2011. aastast [6]. 2016. aastal oli kõige populaarsem ja ühteaegu ka kõige suurem MOOCe pakkuv keskkond *Coursera*, millel oli 2016. aastal 23 miljonit registreerunud kasutajat ja mis andis valida rohkem kui 1700 erineva kursuse vahel [6].

1.2 Tartu Ülikooli programmeerimisalastest MOOCidest

Tartu Ülikool pakub MOOCe mitmes erinevas instituudis. Arvutiteaduse instituut pakub kolme programmeerimisteemalist MOOCi: „Programmeerimisest maaähedaselt”, „Programmeerimise alused” ning „Programmeerimise alused II”.

Esimese kahe kursuse sihtrühmaks on inimesed, kellel puudub eelnev kogemus programmeerimisega ning õppimist alustatakse baasteadmistest. Mõlema kursuse eesmärgiks on tutvustada üldiselt programmeerimist ja algoritmilist mõtteviisi. „Programmeerimisest maalähedaselt” toimus juba kaheksandat korda [7]. Kuna kursus kestab vaid neli nädalat ja on mahus 1 EAP ehk 26 tundi tööd, siis võib ka see olla üks põhjustest, miks kursus on Tartu Ülikooli programmeerimise MOOCide hulgas kõige populaarsem.

„Programmeerimise alused” toimus esimest korda 2016. aasta kevadsemestril ning viimane 2017. aasta sügissemestril, kokku on kursus toimunud neli korda. Kursusel osalemiseks ei ole kohustuslik läbida varasemalt kursus „Programmeerimisest maalähedaselt“, kuid see on

soovituslik, kuna kursuse jooksul käsitletakse põhjalikumalt kursusel „Programmeerimisest maalähedaselt“ läbitud teemasid ning lisaks õpitakse ka uusi teemasid [3].

Kursusele „Programmeerimise alused” pakub Tartu Ülikooli arvutiteaduse instituut ka jätkukursust „Programmeerimise alused II”, mis on suunatud inimestele, kellel eelnevalt on olemas kokkupuude programmeerimisega vähemalt kursuse „Programmeerimise alused” mahus [9]. Nii kursus „Programmeerimise alused” kui „Programmeerimise alused II” kestab kaheksa nädalat ja on mahus 3 EAP ehk 78 tundi iseseisvat tööd [8, 9]. Esimest korda toimus kursus 2017. aastal ja on kokku toimunud kaks korda. „Programmeerimise alused II” keskendub varasemalt omandatud teadmiste kordamisele, süvendamisele, programmide koostamisele ja testimisele [9].

1.3 MOOCi „Programmeerimise alused“ tutvustus

Antud töö osa kirjeldab kursuse „Programmeerimise alused” ülesehitust, teemasid, õppematerjale, ülesandeid, nõudeid, automaatkontrolli ja murelahendajat.

1.3.1 Ülesehitus

Kursuse materjalid on jaotatud nädalate kaupa, igal nädalal keskendutakse uuele teemale. Teemad on nädalate kaupa järjestatud järgnevalt:

1. Sissejuhatus programmeerimisse;
2. Tingimuslause;
3. Tsükkel;
4. Sõned ja graafika;
5. Järjend;
6. Funktsioon;
7. Andmevahetus ja lihtne kasutajaliides;
8. Kordamine.

Iga nädala teema on aga omakorda mitmeks osaks jaotatud, et teemat oleks lihtsam omandada. Näiteks sisaldab 2. nädala teema „Tingimuslause” järgmisi osasid [8]:

- Tingimuslause;
- Tingimuslause tingimuslause sees;
- Mitmeosaline tingimus. Loogilised tehted ja avaldised;
- Mitmeharuline tingimuslause *elif* abil;
- Juhuslik arv;

- Kilpkonnagraafika.

1.3.2 Õppematerjalid

Kursuse materjalid on saadavad Tartu Ülikooli õpikeskkonnas *Moodle* ja Tartu Ülikooli arvutiteaduse instituudi õppematerjalide keskkonnas *Courses*. Igal nädalal on vastava teema kohta kättesaadavad materjalid, mis on soovituslik iseseisvalt läbi töötada enne kontrollülesannete lahendamist. Materjalide hulgas on lisaks erinevatele programmeerimisalastele materjalidele ka videoid, selgitavaid skeeme, ilukirjanduslikke ja silmaringi avardavaid materjale. Lisaks on ka enesetesti küsimusi, et oleks võimalik kiiresti kontrollida, kas peatükist on õigesti aru saadud.

1.3.3 Kontrollülesanded ja nõuded

Iga nädal tuleb antud nädala teema kohta lahendada üks kuni neli iseseisvat ülesannet programmeerimiskeeles Python, mis tuleb esitada *Moodle'*i keskkonnas, nendest suurem osa on kohustuslikud ja viimane ülesanne on võimalik osalejatel valida pakutud ülesannete hulgast. Valikuid on tavaliselt kaks kuni kolm. Ülesandeid hindab automaatkontroll, hinne on kas arvestatud või mittearvestatud. Samuti on iga teema kohta vaja nädala lõpus lahendada *Moodle'*i test, mis koosneb suures osas valikvastustega küsimustest.

1.3.4 Automaatkontroll ja murelahendaja

Õppurite lahendusi hinnatakse automaatkontrolli programmi VPL (ingl *Virtual Programming Lab*) abil, kuna suure osalejate arvu tõttu ei ole ajaliselt võimalik ja mõistlik kõiki lahendusi manuaalselt läbi vaadata. VPL on *Moodle'*i keskkonna jaoks ehitatud moodul, mis võimaldab lihtsalt toimetada erinevate programmeerimisülesannetega. VPL-i abil saab hinnata, grupeerida, salvestada, kopeerida osalejate esitatud ülesandeid [10]. VPL toetab erinevaid programmeerimiskeeli nagu näiteks Python, Java, C++, Ruby ja Haskell [10]. Ülesannete automaatkontroll on seadistatud nii, et vigade esinemisel saab õpilane tagasisidet automaatkontrollilt: milliste parameetritega käivitati programm ning milline oli oodatud ja tegelik väljund. Antud kursusel ei olnud automaatkontrollile ülesande esitamiste arv piiratud.

Lisaks automaatkontrolli tagasisidele on võimalus osalejatel kasutada murelahendajat (ingl *troubleshooter*). Murelahendaja on Vello Vaherpuu koostatud programm, mis annab abistavaid vihjeid ülesannete lahendamisel ja mille vihjed on koostanud Kaspar Hollo [11]. Programmi vihjed on koostatud kursusel osalejate e-kirjade põhjal, et ennetada tekkivaid

probleeme ning vähendada õppejõududele laekuvate meilide hulka [11]. Murelahendajat on kõige efektiivsem kasutada probleemi tekkimisel, kuid samuti on soovituslik võrrelda peale ülesande lahendamist murelahendaja ja enda lähenemist probleemile [12]. Kaspar Hollo bakalaureusetöö raames koostatud vihjete abil vähendati õppejõududele laekunud kirjade arvu rohkem kui kolmandiku võrra [11]. Kui 2016. aastal toimunud kursuse „Programmeerimisest maalähedaselt" osalejad saatsid õppejõududele üle 1250 meili, siis 2017. kursusel „Programmeerimisest maalähedaselt" oli kirjade arv 750 [11].

2. Programmeerimise ülesannete lahendamisest

Antud peatükis annab autor ülevaate programmeerimise ülesannete lahendamisest ja seda mõjutavatest teguritest.

2.1 Programmeerimise õppimist mõjutavad tegurid

Teadaolevalt on programmeerimine seotud nii väga paljude teadus- kui ka eluvaldkondadega. Viimastel aastatel on huvi ja nõudlus programmeerimise õppimise järele oluliselt suurenenud ning samuti on suurenenud programmeerimisalaste MOOCide osalejate arv [14].

Vaatamata suurele huvile on enamikul algajatest programmeerijatest raskusi juba esimeste programmeerimisse sissejuhatavate kursustega, mida näitab ka programmeerimise MOOCide ja sissejuhatavate kursuste pidevalt kõrge väljalangemise protsent. Kui Tartu Ülikooli kursustel on see ligikaudu 40-50% [13], siis välismaistel kursustel võib see tõusta 80-95%ni [14]. Järgnev lõik annab ülevaate, mis tegurid võivad mõjutada õpilaste tulemusi programmeerimise sissejuhatavas kursuses.

Al-Imam Mohammad Ibn Saud Islamic ülikoolis avaldatud artikli kohaselt võib õpilaste programmeerimiseoskust mõjutavad faktorid jagada kaheks kategooriaks: õpilasega seotud tegurid ja õpilasest mittesõltuvad ehk õpikeskkonnaga seotud tegurid [15].

Õpilasega seotud mõjurite all on eelkõige mõeldud faktoreid, mida saab õpilane ise kontrollida ning mille alla Wilson liigitas 2002. aastal järgmised tegurid: õpilase taust, selle juures matemaatiline ja programmeerimise taust, vähene arvuti kasutamise oskus, inglise keele oskus ja halb varasem arvuti kasutamise kogemus [16]. Halva varasema kogemusega võib tihti kaasneda ebakindlus edasiste ettevõtmiste juures. Wilson väidab, et palju suuremat rolli õpilaste programmeerimisalaste soorituste juures kui matemaatiline taust omab õpilaste mugavus ja enesekindlus õppida programmeerimist ja seda just naissoost õpilaste hulgas.

Hispaania Castilla-La Mancha ülikoolis tehtud uuringu käigus arvasid õppejõud, et õpilased ei kirjuta häid programme, sest nad püüavad probleeme lahendada nii kiiresti kui võimalik, mõtlemata nende kvaliteedile ja erinevatele võimalikele lahendustele [20]. Ka see võib olla üks õpilasest tulenev probleem.

Lisaks on leitud, et õpilaste tulemuste ja motivatsiooni vahel on tugev seos [15]. Courney jt väitsid uurimuses, et vähene motivatsioon viib läbikukkumiseni ja halvad tulemused kursuse alguses viivad vähese motivatsioonini [17]. Uurimus tehti Queenslandi ülikooli

programmeerimisse sissejuhatava kursuse ümberstruktureerimisest eesmärgiga motiveerida õpilasi programmeerima ja vähendada kursuse kõrget väljalangevuse protsenti. Motivatsiooni tõstmiseks muudeti kursuse ülesehitust nii, et kursus keskenduks vähem programmeerimiskeelele ja selle süntaksi õppimisele ning oleks rohkem fokuseeritud reaalse eluga seonduvate probleemide lahendamisele ja erinevate tehnoloogiate tutvustamisele. Hiljem, kui asuti praktilisemate ülesannete juurde, võeti kasutusele ka paarisprogrammeerimine.

Tulemused peale kursuse muutmist olid positiivsed, õpilaste tagasisidest tuli välja, kui olulised on nende jaoks teadmised, miks programmeerimine on kasulik, ja langes ka kursuselt väljalangemise protsent. Tänu sellele, et kursuse sissejuhatav osa tehti pikemaks ja üldisemaks, selgitamaks õpilastele probleeme, mida saab lahendada programmeerimise abil, tõsteti õpilaste motivatsioonitaset lahendada programmeerimise ülesandeid ja osaleda kursusel lõpuni.

Teisalt võib õpilaste suhtumist õppetöösse mõjutada Wilsoni arvates õpikeskkond [16]. Keskkonnaga on näiteks seotud õppejõud, kes määravad kursuse ülesehituse, õpetamise stiili ning kursuse keerukuse [16]. MOOCide õpetamise stiili juures tuleks arvestada, et õpetamise stiil ei mõjuta õppureid nii palju, kui seda teeb õppejõud klassiruumis, kuid õppematerjalide esitusviis mõjutab õpilaste õppimise stiili. Kursuse ülesehituse osas on olulisel kohal ka programmeerimiskeele valik, mis peaks olema valitud kursuse eesmärgist lähtuvalt [15]. Rääkida tavalisest koolitunni keskkonnast võib öelda, et suur roll on ka õpilastel endil, kes mõjutavad üksteise suhtumist ja käitumist, kui ka ülikoolil ja klassil kui tervikul [15].

2.2 Algajate programmeerijate vigadest ja probleemidest

Usutakse, et programmeerimine on oskus, mis vajab palju keskendumist ja harjutamist [18]. Sarnaselt on leitud ka 2005. aastal läbi viidud küsitluses, mis viidi läbi programmeerimise baaskursuse läbinud erinevate ülikoolide tudengite hulgas [19], mille tulemusel selgus, et õpilased hindasid üksinda õppimist efektiivsemaks kui loengutes osalemist ja üksi kursuse materjalidega töötamist kasulikumaks kui seminarides osalemist. Kusjuures tegu oli üliõpilastega, kellest üle poolte olid enne kursuse algust kokku puutunud programmeerimisega.

Teisisõnu võib öelda, et programmeerimist ei ole võimalik „ära õppida" lühikese perioodi jooksul. Programmeerimise oskus kujuneb välja pikema aja jooksul pidevalt uute probleemidega tegelemise, analüüsi ja lahendamise oskuse arendamise abil.

2004. aastal Garneri jt [21] poolt koostatud Java sissejuhatava kursuse analüüsi eesmärgiks oli uurida õpilastel kursuse jooksul tekkinud probleeme. Antud uuring viidi läbi kursuse juhendajate abiga, kes märkisid üles kõik õpilaste poolt küsitud küsimused, mis hiljem kategoriseeriti. Iga seminari kohta sõnastati lõpuks maksimaalselt kolm erinevat probleemi. Tulemused näitasid, et kõige rohkem probleeme esines kursuse jooksul põhiteadmistega, mille alla kuuluvad erinevad lihtsamat tüüpi vead. Vastupidiselt ootustele, et baasteadmised võiksid aja jooksul paraneda, esines enamikus seminarides sarnaseid elementaarseid vigu hoopis sagemini.

Järgnesid probleemid programmi disaini ja struktuuri koostamisega ning ülesande tekstist aru saamisega [21]. Samuti toodi välja probleem tsüklite ja järjenditega, mis tuleb väga selgelt välja vastava seminari analüüsigraafikust [21]. Kõige rohkem esines probleeme järjenditega, mille alla loeti õpilaste järgnevad küsimused: järjend kui andmestruktuur, järjendi sisu, järjendi defineerimine, järjendi indekseerimine. Tsüklitega seotud probleemide hulka arvestati nii tsükli päise, keha kui ka tsükli plokkidega seonduvaid vigu.

Nimetatud teemade probleemid olid kõikides seminarides loendatud vigadest kõige suurema esinemiste arvuga ühe seminari kohta, millest võib järeldada, et seminar „Tsüklid ja järjendid" valmistas õpilastele kõige rohkem probleeme.

3. Metoodika

Järgnevas peatükis kirjeldab autor uurimuses kasutatud valimit, analüüsitud ülesandeid ja analüüsiks kasutatud meetodeid.

3.1 Ülesannete kirjeldused

Antud töös on vaatluse all Tartu Ülikooli 2017. aasta sügissemestril toimunud MOOCi „Programmeerimise alused" kaks kohustuslikku ülesannet. Ülesanded on valitud erinevate nädalate teemade hulgast. Esimene analüüsiks valitud ülesanne „Spämm" kuulub 2. nädala teema „Tingimuslaused" kohustuslike ülesannete hulka. Teine analüüsiks valitud ülesanne "Lillede arv" kuulub 3. nädala teema „Tsükkel" kohustuslike ülesannete hulka. Kuna nende ülesannete lahenduste esituste arv oli kõrgem antud nädalate kohustuslike ülesannete omadest, võib eeldada, et ülesanded oli keerulisemad kui teised ning sellepärast valis autor kokkuleppel juhendajaga need kaks ülesannet töös analüüsimiseks. Järgnevalt saab lugeda kahe valitud ülesande püstitusi [22], mis on esitatud samamoodi nagu kursuse materjalides.

3.1.1 Ülesande „Spämm" püstitus

Tabel 1. Ülesande „Spämm" püstitus.

Kirjade seast rämpsposti (spämmi) leidmiseks saab kasutada filtreid, mis filtreerivad välja konkreetsetele tingimustele vastavaid kirju. Kalmer teeb filtrit, kus filtreeritakse välja kirjad, mille kohta on vähemalt üks järgmistest tingimustest tõene:

- kirjal ei ole teema pealkirja,
- kiri sisaldab manusena faili ja kirja suurus ületab 1 MB.

Koostada Kalmeri jaoks programm, milles

1. küsitakse kirja suurust megabaitides (kasutaja sisestab ujukomaarvu),
2. küsitakse kirja teema pealkirja (kasutaja sisestab teema pealkirja või kasutaja sisestus on tühi),
3. küsitakse, kas kirjaga on kaasas fail (kasutaja sisestab "*jah*" või "*ei*"),
4. väljastatakse ekraanile "*Kiri on spämm*", kui kiri filtreeritakse välja, vastasel juhul väljastatakse "*Kiri ei ole spämm*".

Proovige kirjutada programm, kasutades ainult ühte tingimuslauset. Kui see ei õnnestu, siis võib ka mitmega.

NB! Kasutaja käest peab kindlasti küsima kolm korda.

Näited programmi tööst:

```
>>> %Run yl2.2.py

  Sisestage kirja suurus: 0.7
  Sisestage kirja teema pealkiri: Ülesanne 2.2
  Kas kirjaga on kaasas fail? jah
  Kiri ei ole spämm

>>>

>>> %Run yl2.2.py

  Sisestage kirja suurus: 0.8
  Sisestage kirja teema pealkiri:
  Kas kirjaga on kaasas fail? ei
  Kiri on spämm

>>>
```

3.1.2 Ülesande „Lillede arv“ püstitus

Tabel 2. Ülesande „Lillede arv“ püstitus.

On traditsioon, et rõõmsatel puhkudel kingitakse paaritu arv lilli. Lillepoel on sünnipäev ja pood otsustas klientidele kinkida lilli nii, et päeva esimene ostja saab ühe lille, teine ei saa ühtegi, kolmas ostja saab kolm lille, neljas ei saa midagi, viies ostja saab viis lille jne.

Koostada programm, mis

- küsib kasutajalt klientide arvu (mittenegatiivne täisarv);
- arvutab while-tsükli abil lillede koguarvu, mida pood kingib;
- väljastab saadud lillede arvu ekraanile.

Vihje: lillede koguarvust võib mõelda kui summast, milles liidetavad on paaritud arvud alates 1 kuni esimese paaritu arvuni, mis pole suurem kui klientide arv.

Näiteks, kui kasutaja sisestas 7, siis paaritute arvude summa on 16, sest 1 + 3 + 5 + 7 = 16. Kui kasutaja sisestas 8, siis on summaks samuti 16, sest 1 + 3 + 5 + 7 = 16.

Näited programmi tööst:

```
>>> %Run yl3.2.py

  Sisesta ostjate arv: 7
  Lillede koguarv on 16.

>>>

>>> %Run yl3.2.py

  Sisesta ostjate arv: 8
  Lillede koguarv on 16.

>>>
```

3.2 Valim

2017. aasta sügissemestril osales „Programmeerimise alused" kursusel kokku 1518 õppurit. Osalejate vanus jäi vahemikku 14 kuni 74 eluaastat ning kursusel oli 759 naissoost ja 759 meessoost osalejat. Töö valimiks on need osalejad ja vastavalt nende osalejate ülesannete „Spämm" ja „Lillede arv" lahendused, mis ei saanud esimese *Moodle*'i keskkonda esitatud esitusega automaatkontrollilt arvestavat hinnet. Ülesande „Spämm" puhul oli 810 õppurit, kes esitasid oma lahenduse *Moodle*'isse enam kui üks kord ja ei saanud kohe hindeks arvestatud ja ülesande „Lillede arv" puhul oli neid 243. Ülesannet „Spämm" lahendas kokku 1315 õppurit ja ülesannet „Lillede arv" lahendas 1213 õppurit.

3.3 Protseduur

Antud töös kasutas autor ülesannete lahenduste analüüsiks kvantitatiivset uurimismeetodit. Andmete kogumiseks kasutati 2017. aasta sügissemestril kursuse „Programmeerimise alused" läbinud osalejate kahe erineva ülesande lahendusi.

Vigade väljaselgitamiseks ja liigitamiseks vaatas autor läbi 50 lahendust kummagi ülesande puhul ning seejärel sõnastas vigade tüübid. Liigitatud vigade analüüsiks ning statistika koostamiseks koostas autor kummagi ülesande jaoks kaks andmetabelit, millest üks oli arvestatud lahenduste ja teine mittearvestatud lahenduste jaoks. Analüüsi käigus vaadati läbi kummagi ülesande puhul kõik lahendused, mille esituste arv *Moodle*'is oli suurem kui üks kord, seega vaatluse alla ei kuulunud esmasel esitusel automaatkontrolli poolt arvestatuks loetud lahendused. Andmetabelitesse märgiti kõik vastavas esituses esinenud vead. Ülesannete lahenduste vead on autori ja juhendaja kokkuleppel andmetabelitesse märgitud iga õppuri enda lahendusest lähtuvalt, kuna mõlema analüüsitud ülesande puhul on olemas palju võimalikke lahendusi ning õiged lahendused võisid olla vastandlikke tingimusi või

muutujaid nõudvad. Kuigi autor vaatas läbi ka mittearvestatud lõpphindega lahendused, ei ole tehtud nende lahenduste kohta eraldi analüüsi (vt Lisa I, Lisa II), kuna lahendustes esinenud vead olid samad nii arvestatud kui mittearvestatud tööde juures ja lõplikult mittearvestatud lahendusi oli mõlema ülesande puhul vähe (Ülesande „Spämm" juures 22 tükki ja ülesande „Lillede arv" juures 11 tükki).

4. Tulemused

Järgnevas peatükis antakse ülevaade analüüsitud ülesannete lahendustest, esinenud vigadest ja nende statistikast. Samuti annab autor soovitusi kursuse materjalide täiendamiseks.

4.1 Ülesande „Spämm“ lahendused

Kokku lahendas ülesannet „Spämm” 1315 osalejat, kellest arvestatud hinde said 1293 osalejat ja mittearvestatud 22 osalejat. 1293-st osalejast sai esimese esituse korraga mittearvestava hinde 788 osalejat. Analüüsi käigus vaatas autor läbi ligi 5400 lahenduse esitust. Ülesanne oli kõige kõrgema keskmise lahenduste esituste arvuga ülesanne kursusel, keskmine esituste arv kõigi lahenduste esituste peale oli 4,6 korda ja esmalt mittearvestatud hinde saanud osalejate esituste keskmine arv oli 6,7 korda.

Üks ülesande tingimus oli, et programm küsiks kasutaja käest kolme sisendit: kirja suurust, kirja teema pealkirja ja kas kirjaga on kaasas fail (vt alapeatükk 3.1.1). Antud osaga said suur osa lahendajatest hästi hakkama ja sisendid küsiti õiges järjekorras. Sobivaid tüüplahendusi oli kolm:

- ```
  suurus = float(input("Sisestage kirja suurus: "))
  pealkiri = input("Sisestage kirja teema pealkiri: ")
  fail = input(Kas kirjaga oli kaasas fail?")
  ```

- ```
  print("Sisestage kirja suurus: ")
  suurus = float(input())
  print("Sisestage kirja teema pealkiri: ")
  pealkiri = input()
  print("Kas kirjaga oli kaasas fail?")
  fail = input()
  ```

- ```
  print("Sisestage kirja suurus: ")
  suurus = input()
  suurus = float(suurus)
  print("Sisestage kirja teema pealkiri: ")
  pealkiri = input()
  print("Kas kirjaga oli kaasas fail?")
  fail = input()
  ```

Analüüsitud programmi põhiosa lahendused saab liigitada nelja erinevasse kategooriasse (vt Tabel 3). Tingimused on esitatud:

1. ühe *if*- ja ühe *else*-haru abil;
2. *if*-, *elif*- ja *else*-harude abil;
  ```

3. *if*- ja *else*-harude abil, mille sees on *if*- ja *else*-harud;
4. paljude kombinatsioonide abil: kasutades ühte *if*- ja ühte *else*-haru, kus *if*-harus on palju tingimusi esitatud või kasutades *if*-, *else*- ja palju *elif*-harusid.

Tabel 3. Ülesande „Spämm“ põhiosa tüüplahendused.

Lahendus 1

Lahendus 1a

```
if pealkiri == "" or fail == "jah" and suurus > 1:
    print("Kiri on spämm.")
else:
    print("Kiri ei ole spämm.")
```

Lahendus 1b

```
if pealkiri != "" and (fail != "jah" or suurus <= 1):
    print("Kiri ei ole spämm.")
else:
    print("Kiri on spämm.")
```

Lahendus 2

```
if pealkiri == "":
    print("Kiri on spämm.")
elif fail == "jah" and suurus > 1:
    print("Kiri on spämm.")
else:
    print("Kiri ei ole spämm.")
```

Lahendus 3

```
if len(pealkiri) == 0:
    print("Kiri on Spämm")
elif fail == "jah":
    if suurus > 1:
        print("Kiri on spämm"
    else:
        print("Kiri ei ole spämm")
else:
    print("Kiri ei ole spämm")
```

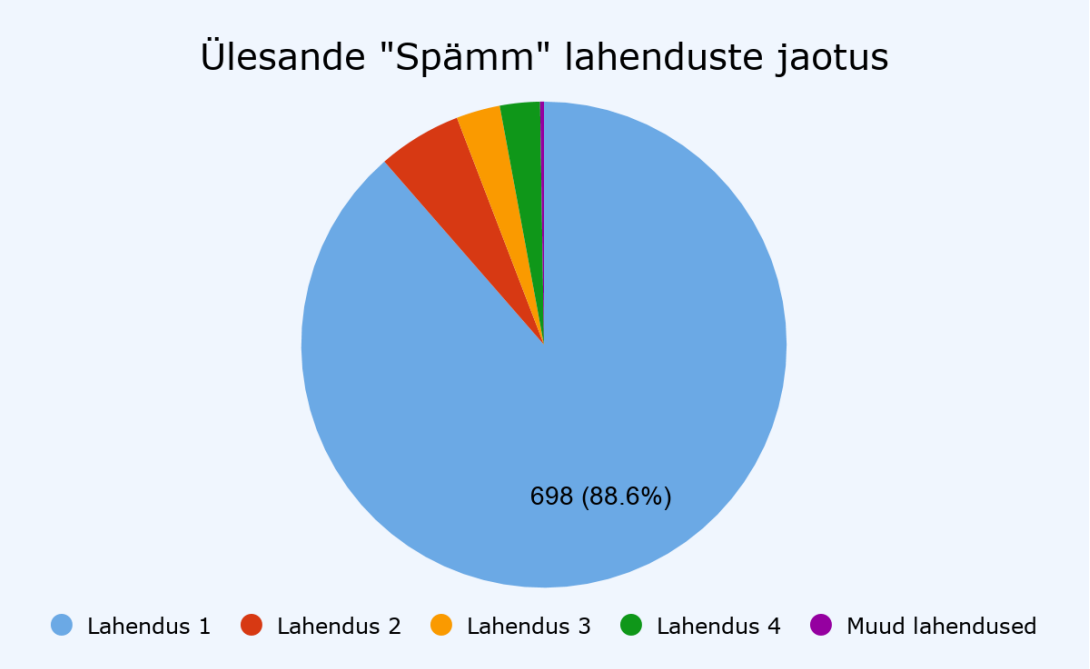
Lahendus 4

```
if suurus > 1 and fail == "jah" and pealkiri == "":
    print("Kiri on spämm")
elif suurus > 1 and fail == "ei" and pealkiri == "":
    print("Kiri on spämm")
elif suurus > 1 and fail == "jah" and pealkiri == pealkiri:
    print("Kiri on spämm")
elif suurus > 1 and fail == "ei" and pealkiri == pealkiri:
```

```
    print("Kiri ei ole spämm")
elif suurus <= 1 and pealkiri and fail == "jah":
    print("Kiri ei ole spämm")
elif suurus <= 1 and pealkiri and fail == "ei":
    print("Kiri ei ole spämm")
elif suurus <= 1 and pealkiri == "" and fail == "ei":
    print("Kiri on spämm")
elif suurus <= 1 and pealkiri == "" and fail == "jah":
    print("Kiri on spämm")
```

4.1.1 Lahenduste analüüs

Iga läbivaadatud lahenduse juures märkis autor lisaks tehtud vigadele andmetabelisse vastavalt sobiva lahendustüübi (vt Lisa I). Lahendustüüp määrati igale lahendusele *Moodle*'isse osaleja viimase esitatud arvestatud lahenduse põhjal. Näiteks kui õppur oli lahendamist alustanud lahenduse 4 suunas, kuid viimane arvestatuks loetud lahendus oli lahendus 1, siis märgiti andmetabelisse vastavalt tüübiks lahendus 1. Samuti tuleks ära märkida, et *if*-harude hulga ja paigutuse järgi kategooriatesse liigitatud lahenduste hulgas esines tingimuste osas ka osaline variatiivsus.



Joonis 1. Ülesande „Spämm“ lahenduste jaotus.

Esitatud lahenduste tüübid jagunesid järgnevalt (vt Joonis 1): lahendust 1 esines 698 korda (88,6%), lahendus 2 esines 44 korda (5,6%), lahendus 3 esines 23 korda (2.9%) ning lahendus 4 esines 21 korda (2,7%). Lisaks esines veel kaks lahendust (0,3%), mis ei liigitunud ühegi lahendustüübi alla. Kõige rohkem esines esimest tüüpi lahendusi, mille

hulka arvestati lisaks ülesande tingimustele vastavatele lahendustele ka vastupidiseid tingimusi kasutavad lahendused (vt Tabel 3, Lahendus 1b). Sellest võib järeldada, et paljud õppurid siiski jõudsid lõpuks kõige lühema lahenduseni ning said oma vigadest aru, vaatamata suurtele esituste arvudele.

4.2 Vigade liigitus ülesandes „Spämm“

Autori ja juhendaja ühise arutluse järel sõnastati ülesande „Spämm” lahenduste läbivaatlusel 12 erinevat veatüüpi. Veatüübid antud ülesandes olid järgnevad:

1. **Valede loogiliste operaatorite kasutus** – teise tingimuse (faili olemasolu ja kirja suuruse tingimus) osad olid ühendatud vale operaatori abil või ülesande esimene ja teine tingimus olid omavahel kombineeritud vale operaatori abil või operaator puudus. Näiteks:
   ```
   if suurus > 1 or fail == "jah" and pealkiri == ""
   ```
2. **Valede väärtustega võrdlemine** – selle vea alla kuulusid: pealkirja võrdlus vale väärtusega, kirja olemasolu võrdlus vale väärtusega, kirja suuruse võrdlus vale väärtusega. Näiteks:
   - `pealkiri == " "`
   - `fail == ""`
   - `suurus < 0.9`
3. **Vale võrdlusoperaatori kasutus** – kirja suuruse tingimuses vale võrdlusoperaator kasutamine. Näiteks:
   - `suurus < 1`
   - `suurus == 1`
   - `suurus >= 1`
4. **Tingimuste vale kombineerimine** – ülesande „Spämm” püstituses kirjeldatud pealkirja tingimus oli valesti kombineeritud teise ülesandes esitatud kirja suuruse ja faili tingimusega. Näiteks:
   ```
   if suurus > 1 and pealkiri == "" or fail == "jah":
   ```
5. **Vastupidine väljund vastavalt tingimusele** – *if*- või *else*-tingimusele oli vastavusse pandud vastupidine väljund. Näiteks:
   ```
   if pealkiri == "" or fail == "jah" and suurus > 1:
       print("Kiri ei ole spämm.")
   else:
       print("Kiri on spämm.")
   ```

6. **Tingimuste vale grupeerimine sulgudega või *if*-harudega** – valesti tingimuste grupeerimine, seejuures ka tihti tingimuste kordamine, *if*- ja *elif*- tingimuste või sulgude abil. Näiteks:

```
if (pealkiri == "" or suurus > 1) and fail == "jah"
```

7. **Automaatkontrolli nõudmistest tulnud vead** – lahendus ei läinud automaatkontrollist läbi automaatkontrolli nõudmiste tõttu. Näiteks:

```
if pealkiri == "" or fail =="Jah" and suurus > 1:
    print("Kiri on spämm")
else:
    print("Kiri ei ole spämm")
```

8. **Tingimuste osaline puudumine** – ülesande püstituses esitatud tingimustest oli mõni puudu. Näiteks:

```
if pealkiri == "" or suurus > 1:
    print("Kiri on spämm")
else:
    print("Kiri ei ole spämm")
```

9. **Probleem muutujate tüüpidega** – tingimuse päises püüti võrrelda erinevat tüüpi muutujaid. Näiteks:

```
suurus = int(float(input("Sisesta faili suurus: ")))
pealkiri = input("Sisesta pealkiri: ")
fail = input("Kas fail on olemas?")
if pealkiri == "" or fail == False and suurus > "1":
    print("Kiri on spämm")
```

10. **Trüki- või süntaksivead.** Näiteks:

```
suurus = float(input("Sisesta kirja suurus:")
```

11. **Küsitud sisendite üleväärtustamine enne *if*-plokki** – enne programmi *if*-tingimustesse sisenemist toimus kasutaja käest küsitud sisendite üleväärtustamine. Näiteks:

```
suurus = float(input("Sisesta kirja suurus:"))
pealkiri = input("Sisesta pealkiri:")
fail = input("Kas fail on olemas?")
fail = "jah"
pealkiri == "Programmeerimine"
suurus = 0.5
```

12. **Probleemid muutujatega** – kasutajalt küsitud sisendeid ei salvestatud muutujasse või puudus muutuja, mida kasutati tingimuses. Näiteks:

```
pealkiri = " "
suurus = float()
fail = "jah"
```

```
print(input("Sisestage kirja teema pealkiri:"))
print(input("Sisestage kirja suurus:"))
print(input("Kas kirjaga on kaasas fail?"))
if pealkiri == str(" ") or (suurus > 1 and fail == "jah"):
    print("Kiri on spämm")
else:
    print("Kiri ei ole spämm")
```

4.2.1 Vigade analüüs

Ülesande „Spämm" arvestatud lahenduste andmetabelis loeti kokku 2183 erinevat viga kõigi selle ülesande läbi vaadatud lahenduste peale kokku. Järgnevalt on toodud ülesande „Spämm" vigade jaotus (vt Tabel 4).

Tabel 4. Ülesande „Spämm" vead.

Veatüüp	**Kokku**	**Protsent**
Viga 1 – valede loogiliste operaatorite kasutus	486	22,3%
Viga 2 – valede väärtuste võrdlemine	443	20,3%
Viga 3 – vale võrdlusoperaatori kasutus	320	14,7%
Viga 4 – tingimuste vale kombineerimine	302	13,8%
Viga 5 – vastupidine väljund vastavalt tingimusele	169	7,7%
Viga 6 – tingimuste vale grupeerimine sulgudega või *if*-harudega	150	6,9%
Viga 7 – automaatkontrolli nõudmistest tulnud vead	118	5,4%
Viga 8 – tingimuste osaline puudumine	100	4,6%
Viga 9 – probleem muutujate tüüpidega	64	2,9%
Viga 10 – trüki- või süntaksivead	19	0,9%
Viga 11 – küsitud sisendite üleväärtustamine enne *if*-plokki	10	0,5%
Viga 12 – probleem muutujatega	2	0,2%

Kõige sagedasem oli viga 1, mida märgiti andmetabelisse 486 korda (22,3%). Antud viga seisnes ülesandes esitatud tingimuste valede loogiliste operaatorite abil kombineerimises. Võib öelda, et see viga on seotud veaga 4 ehk tingimuste vales järjekorras kombineerimisega, sest tingimuste vale järjekorra tõttu on valed ka loogilised operaatorid. Tingimuste vale kombineerimist esines 302 korda (13,8%).

Ülesande tekst ütles, et kiri on spämm siis, kui vähemalt üks järgmistest tingimustest on tõene:

- kirjal ei ole teema pealkirja;
- kiri sisaldab manusena faili ja kirja suurus ületab 1 MB.

Üle kolmandiku lahendajatest on esmalt üritanud pealkirja tingimust lisada teise kaheosalise tingimuse sisse, vaatamata sellele et tingimused olid loetletud. Võib oletada, et osalejad ei osanud ülesande teksti põhjal järeldada, et tingimused peavad olema ka programmis samas järjekorras kirjutatud (või vastupidises), kui need ei ole numbrite abil järjestatud. Võimalik, et õppuritele oleks numbrite abil järjestatud tingimused olnud arusaadavamad ja vähem segadust tekitavad.

Võib oletada, et valede operaatorite kasutamise suur esinemiste arv tulenes osaliselt sellest, et tingimuste järjekord oli vale, kuid probleemi võis ka tekitada ülesande tekst, kus on küll kasutatud sõna „ja" teise tingimuse osade ühendamisel, kuid sõna „või", millega peaks olema ühendatud esimene ja teine tingimus, ei ole ülesande tekstis kasutatud [23]. Alternatiivselt on kasutuses väljend „kus vähemalt üks tingimustest on tõene", mis võis jätta õppurid segadusse, kuigi ka selle lause kohta on peatüki materjalis „Mitmeosaline tingimus" avaldise näide toodud [23]. Sellest võib järeldada, et osalejad, kes pole materjali korralikult lugenud, ei tea, millise operaatoriga sellise sõnastuse puhul ülesande kaks põhitingimust tuleks kombineerida. Kuigi paljudele õppuritele sai esmalt saatuslikuks üks operaatoritest, juhtis see viga katseeksitusmeetodini, st katsetati läbi veel kõik läbi proovimata kombinatsioonid. Võimalik, et abi oleks ülesandesisestest viidetest õppematerjalidele, siis oleks suurem tõenäosus, et ka need osalejad, kes materjale pole varem lugenud või ei lugenud nii korralikult, saavad neid ülesande lahendamise ajal lugeda ja tekitada seoseid. Seega kui lahendamise käigus peaks tekkima küsimusi, siis materjali viide tuletab õppuritele meelde, kust võib lisaks murelahendajale abi leida.

Sageduselt järgmine oli viga 2, mida märgiti andmetabelisse 443 korda (20,3%). Antud viga seisnes tingimuses valede väärtuste kasutamises. Seega, kui lahenduses oli vähemalt üks kolmest (pealkirja, faili ja kirja suuruse) muutujast võrreldud vale väärtusega, märgiti viga andmetabelisse. Kuna analüüsi käigus esines kõikides nõutud tingimustes valede väärtustega võrdlusi, siis liigitati need ühe veatüübi alla. Pealkirja tingimus põhjustas kolmest
tingimusest kõige rohkem probleeme. Sellele viitab ka kõige suurema variatiivsusega

väärtuste hulk, mida õppurid püüdsid pealkirja tingimuse võrdlusel kasutada (vt Lisa V). Autor on veendunud, et kõige enam prooviti rakendada võrdlust `pealkiri == " "`, mille põhjal saab järeldada, et paljud lahendajad ei erista, mis on tühi sõne ja mis on tühikuga sõne.

Sageduselt järgmine oli viga 3, mida märgiti andmetabelisse 320 korda (14,7%). Antud viga seisnes vale loogilise operaatori kasutuses kirja suuruse tingimuses. Kõikide lahenduste puhul olid valedeks operaatoriteks >=, < ja ==. Õigeks operaatoriks võis sõltuvalt lahendusest olla vastavalt ülesandes nõutud tingimusele kas > või vastandlahenduse korral eelmisele vastupidine tingimus ehk <=. Kuna võrdlusoperaatoreid õpitakse juba põhikooli algklassides, ei saanud viga tekkida sellest, et ei tuntud nende märkide tähendust. Probleem võib olla põhjustatud sellest, et esmasel ülesande teksti lugemisel ei pööratud piisavalt tähelepanu väiksematele programmi detailidele ja ei mõeldud või ei saadud aru sõna „ületab" tähendusest.

Viga 5 ehk vastupidise väljundi kasutamise kohta vastavalt tingimusele tehti andmetabelisse 169 märget (7,7%). See viga märgiti tabelisse siis, kui vähemalt ühele tingimustest oli vastavusse pandud väljund, mis pidi olema väljundiks vastandtingimuse puhul. Selle vea tekitaja võib olla aga vastupidine eelmise vea põhjusele ehk ülesande teksti lugemisel ei ole aru saadud ülesande teksti peamisest mõttest ehk mis tingimusel on kiri spämm ja mis tingimustel ei ole kiri spämm.

Viga 6, tingimuste vale grupeerimine sulgude või *if*-plokkide abil, esines 150 korda (6,9%). Antud viga süvenes sageli juhul, kui esimene esitatud lahendus ei läinud automaatkontrollist läbi. Tulemuseks oli, et kasutusele võeti katseeksitusmeetod ehk erinevate kombinatsioonide väljakirjutamine sulgude või *if*-plokkide kasutamise abil. Kuid samuti esines ka lahendusi, mis vastasid täpselt automaatkontrolli testides kasutatud kombinatsioonidele. Kuna automaatkontroll näitab peale ühte programmi käivituse korda kõiki teste koos sisenditega ka õppuritele, kasutasid mõned õppurid olukorda ära ja kirjutasid programmi vastavalt nendele sisenditele.

Viga 7 ehk automaatkontrolli nõudmistest tulnud viga märgiti andmetabelisse kokku 118 korda (5,4%). Ülesande tekstis on selgelt jutumärkide abil kirjas, millised sõnalised väärtused kasutaja peab sisestama faili olemasolu kontrollküsimuse peale ning samuti on kirjas, milline on korrektne programmi väljundi sõnastus (vt alapeatükk 3.1.1). Siit järeldub, et osalejad ei ole ülesande teksti piisavalt süvenenud või on hakanud ülesannet lahendama

ülesande teksti enne täielikult läbi lugemata. Kuid samas tuleks arvestada, et tegu on kursuse 2. nädala ülesandega ning on võimalik, et õppurid ei osanud oodata nii rangelt sõnaliste tekstide kontrolli automaatkontrollis. Tegu oli kursuse esimese sõnalist kontrolli sisaldava ülesandega. Nii mõnelgi osalejal põhjustas see viga suurt esituste arvu.

Viga 8 ehk tingimuste osaline puudumine märgiti andmetabelisse kokku 100 korda (4,6%). Viga esines tihti õppurite esimeste esituste hulgas, kus terve programm ei olnud veel valmis, kuid seni tehtud programm töötas tingimuste kohaselt.

Viga 9 ehk probleem muutuja tüüpidega märgiti andmetabelisse kokku 64 korda (2,9%). Probleemi esines kõige enam faili olemasolu kontrolli puhul, kui küsitud sisendi tüübiks oli sõne, kuid tingimuslauses kontrolliti kas kasutaja sisestas tõeväärtuse.

Viga 11, milleks oli küsitud sisendite üleväärtustamine enne *if*-plokki, märgiti andmetabelisse vaid 10 korda (0,5%). Võimalik, et lahendajatel oli vea tegemise hetkel mingil põhjusel veel ebaselge, kuidas programm peaks töötama ning mida lahenduse osas neilt täpselt oodatakse.

Kõige vähem esines viga 12 ehk probleeme muutujatega – kaks korda (0,2%). Kuid samuti leidus trüki- ja süntaksivigu ehk viga 10, mida märgiti andmetabelisse 19 korda (0,9%).. Nende vigade põhjuseks võib olla asjaolu, et kõik õppurid ei kasuta Thonnyt oma programmi silumiseks ja alustavad ülesande lahendamist *Moodle*'i keskkonnas ning seetõttu ei jõua jälile mõningatele näpuvigadele, mille kohta automaakontroll ei anna tagasisidet (vt Lisa V, Trüki- ja süntaksivead). Samamoodi võib selle põhjuseks olla Pythoni süntaksi vähene tundmine kursuse teisel nädalal.

Kokkuvõtteks võib öelda, et ülesande „Spämm" lahenduste vigade arv oli suur ja ülesanne oli osalejatele raske. Samas said paljud osalejad lõpuks ikkagi tulemuseks arvestatud, mis näitab, et nad suutsid oma vigu iseseisvalt nii materjalide, murelahendaja, foorumi või kirjade abil ära lahendada, vaatamata paljudele esituskordadele. Garneri jt läbi viidud uuringus selgus [21], et õpilastel esines mitme nädala jooksul probleeme baasmehhanismidega ning selles osas ühtivad uuringu tulemused antud ülesande analüüsi tulemustega, kuna „Spämm" ülesanne hõlmas erinevaid programmeerimise baasteadmisi, näiteks: muutujad, andmetüübid, tingimuslaused, loogilised tehted ja avaldised.

4.3 Ülesande „Lillede arv“ lahendused

Kokku lahendas ülesannet „Lillede arv” 1213 osalejat, kellest arvestatud hinde said 1202 osalejat ja mittearvestatud 11 osalejat. 1202-st osalejast sai esimese esituse korraga mittearvestava hinde 232 osalejat. Analüüsi käigus vaatas autor läbi ligi 830 lahenduse esitust. Keskmine esituste arv kõigi lahenduste esituste peale oli 1,6 korda ja esmalt mittearvestava hinde saanud osalejate esituste keskmine arv oli 3,4 korda.

Üks ülesande tingimus oli, et programm küsib kasutaja käest sisendiks klientide arvu, mis on täisarv (vt alapeatükk 3.1.2). Sobivaid lahendusi oli kolm:

- `kliendid = int(input("Sisesta ostjate arv: "))`
- ```
  print("Sisesta ostjate arv: ")
  kliendid = int(input())
  ```
- ```
  print("Sisesta ostjate arv: ")
  kliendid = input()
  kliendid = int(kliendid)
  ```

Analüüsitud programmi põhiosa lahendused sai liigitada viide erinevasse kategooriasse (vt Tabel 5). Lahenduste tüüpideks olid:

1. *while*-tsükkel tingimusega `i <= kliendid`, loenduri suurendamine 1 võrra ja *if*-tingimuse kasutus;
2. *while*-tsükkel tingimusega `i <= kliendid`, loenduri suurendamine 2 võrra, ilma *if*- tingimuseta;
3. *while*-tsükkel tingimusega `kliendid > 0`, loenduri vähendamine 1 võrra ja *if*-tingimuse kasutus;
4. *while*-tsükkel tingimusega `kliendid > 0`, loenduri vähendamine 2 võrra ilma *if*-tingimuseta;
5. lahendus, kus summa arvutamisel oli kasutusele võetud keerulisem valem kui summa leidmine liitmise teel.

Tabelis 5 välja toodud lahendustel võis olla aga mitmeid variatsioone, mis kajastusid näiteks *if*-tingimuse, loenduri (koodis muutuja `i`) ja/või summa (koodis muutuja `summa`) asukoha või summa ja/või loenduri algses väärtuse erinevuses (vt Lisa III).

Tabel 5. Ülesande „Lillede arv“ põhiosa tüüplahendused.

Lahendus 1

```
i = 0
summa = 0
while i <= kliendid:
    if i % 2 == 1:
        summa += i
    i+=1
print("Lillede koguarv on " + str(summa))
```

Lahendus 2

```
i = 1
summa = 0
while i <= kliendid:
    summa += i
    i += 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 3

```
summa = 0
while kliendid > 0:
    if kliendid % 2 == 1:
        summa += kliendid
    kliendid -= 1
print("Lillede koguarv on " + str(summa))
```

Lahendus 4

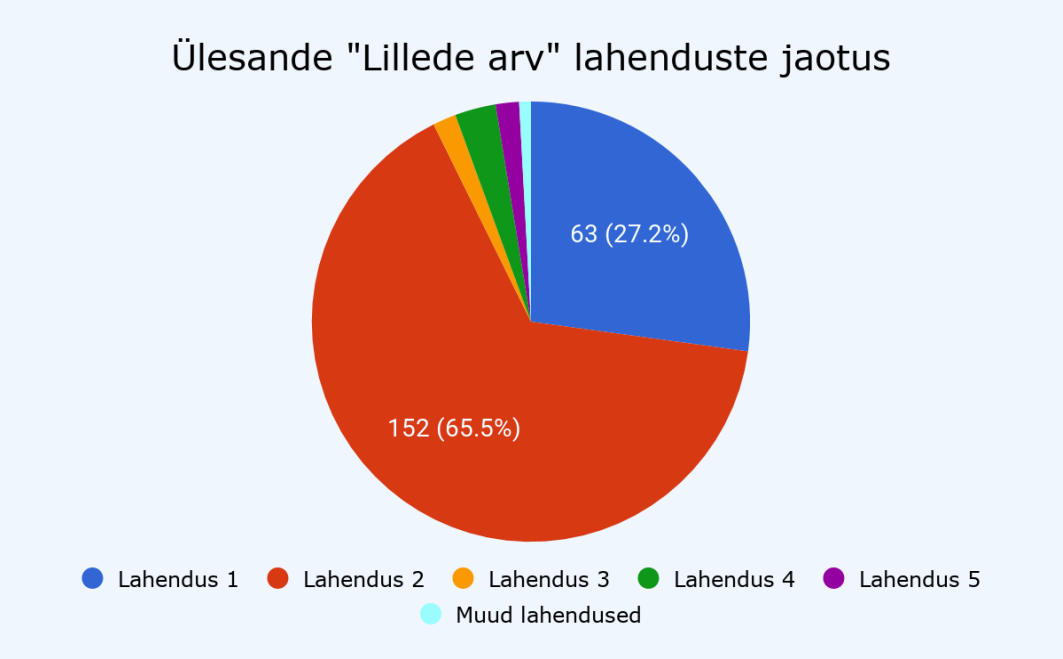
```
summa= 0
while kliendid > 0:
    summa += kliendid
    kliendid -= 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 5

```
lilled = 1
summa = 0
while lilled <= kliendid:
    summa = (kliendid // 2 + kliendid % 2) **2
    lilled = lilled + kliendid
print("Lillede koguarv on " + str(summa))
```

4.3.1 Lahenduste analüüs

Nagu ülesande „Spämm” puhul märgiti ka selle ülesande puhul lahendustüüp andmetabelisse. Järgneval joonisel (vt Joonis 2) on toodud ülesande „Lillede arv” lahenduste jaotus.



Joonis 2. Ülesande „Lillede arv“ lahenduste jaotus.

Eelnevalt esitatud lahenduste tüübid jagunesid järgnevalt: lahendus 1 esines 63 korda, lahendus 2 esines 152 korda, lahendus 3 esines 4 korda, lahendus 4 esines 7 korda ning lahendus 5 esines 4 korda. Lisaks oli kaks lahendust, mis ei liigitunud ühegi kategooria alla (Joonis 2, Muud lahendused).

Ülesanne osutus lahendajatele lihtsamaks kui ülesanne „Spämm”, kuid siinkohal tuleks ära märkida, et mitte kõik osalejate lõplikud arvestatud lahendused ei olnud lihtsasti arusaadava programmi kujul. Nimelt esines selliseid lahendusi, kus loenduri algne väärtus oli negatiivne ja seetõttu oli tsükli sees loenduri suurendamine enne summa suurendamist (vt Lisa IV, Lahendus 2b), kuid sellegipoolest õige lahendus. Peale selle oli ka lahendusi, kus summa algne väärtus oli mingi muu väärtus kui null või kus lisaks hädavajalikele muutujatele oli kasutusele võetud lisamuutuja(d) (vt Lisa IV, Lahendus 2c), mis muudavad programmi mõistmise keerulisemaks. Lahenduse 1 juurde kuulus ka selline lahendus, kus vale *if*-tingimuse puhul liideti summasse ikkagi õige arv (vt Lisa IV, Lahendus 1b).

Eelnevalt mainitud lahenduste põhjal ei saa aga öelda, kas tegemist oli osalejate teadmatusega ja proovimise teel leitud lahendustega või nad olid ikkagi täielikud teadlikud, kuidas nende programm töötab.

Teisalt tuleks ära märkida, et analüüsitud lahenduste hulgas oli ka selliseid lahendusi, kus oli näha, et õppur on lahendamisega rohkem tegelenud kui enamus osalejaid. Kuigi antud ülesannet oli keelatud lahendada *for*-tsükliga, leidus üks lahendus, kus *while*-tsükli tingimus koosnes *range*-funktsoonist, mida tavaliselt kasutatakse *for*-tsükliga (vt Lisa IV, Lahendus 1e). Erinevates lahenduste tüüpides esines ka kasutaja sisestatud klientide summa arvu üleväärtustamist, näiteks jagati sobiva arvu saamiseks see läbi kahega, mis tegelikult on ka üks üsna efektiivselt ja õigesti töötavatest lahendustest (Lisa IV, Lahendus 4c, Lahendus 2e).

4.4 Vigade liigitus ülesandes „Lillede arv“

Autori ja juhendaja ühise arutluse järel sõnastati ülesande „Lillede arv” lahenduste läbivaatlusel 9 erinevat veatüüpi. Veatüübid antud ülesandes olid järgnevad:

1. **Vale summa liidetav** – summa suurendamine vale väärtuse võrra või liidetav puudu. Näiteks:

```
while i <= kliendid:
    summa += 1
    i += 2
```

2. **Vale summa algne väärtus või summa muutuja puudu** – summa muutujale oli väärtustatud ebasobiv algne väärtus. Näiteks:

```
summa = 3
```

3. **Vale väljundi asukoht** – *print*-lause asus tsükli sees. Näiteks:

```
i = 1
summa = 0
while i <= kliendid:
    summa += i
    i += 2
    print("Lillede koguarv on : " + str(summa))
```

4. **Vale loenduri algväärtus või loenduri muutuja puudu** – loenduri muutujale oli väärtustatud ebasobiv algne väärtus. Näiteks:

```
i = 0
summa = 0
while i <= kliendid:
    summa += i
    i += 2
```

5. **Vale *while*-tsükli tingimuse võrdlusoperaator** – *while*-tsüklis kasutatud vale võrdlusoperaator. Näiteks:

```
while i < kliendid:
    summa += i
    i += 2
```

6. **Vale väärtusega võrdlus *while*-tsükli tingimuses** – *while*-tsükli tingimuses kasutatud vale väärtus. Näiteks:

```
while kliendid > 3:
```

7. **Loenduri ja lillede summa suurendamine vales järjekorras** – loenduri ja lillede arvu suurendamine olid vales järjekorras. Näiteks:

```
i = 1
while i <= kliendid:
    i += 2
    summa += i
```

8. **Vigane väljundlause või väljundlause puudu** – väljastati vale muutuja väärtus, väljastati väärtuseid mitu korda või väljundlause oli puudu. Näiteks:

```
i = 1
summa = 0
while i <= kliendid:
    summa += i
    i += 2
print("Lillede koguarv on : " + str(i))
```

9. **Vale loenduri liidetav** – loenduri suurendamine vale väärtuse võrra kui *if*-tingimus puudub või loenduri liidetav puudu. Näiteks:

```
i = 1
summa = 0
while i <= kliendid:
    summa += i
    i += 1
```

4.4.1 Vigade analüüs

Ülesande „Lillede arv" arvestatud lahenduste andmetabelis loeti kokku 475 erinevat viga kõigi läbi vaadatud lahenduste peale kokku. Järgnevalt on toodud ülesande „Lillede arv" vigade jaotus (vt Tabel 6).

Tabel 6. Ülesande „Lillede arv" vead.

Veatüüp	**Kokku**	**Protsent**
Viga 1 – vale summa liidetav või liidetav puudu	68	14,3%

Viga 2 – vale summa algne väärtus	66	13,9%
Viga 3 – vale väljundi asukoht	61	12,8%
Viga 4 – vale loenduri algväärtus	46	9,7%
Viga 5 – vale võrdlusoperaator	46	9,7%
Viga 6 – vale *while*-tsükli tingimus	45	9,5%
Viga 7 – loenduri ja lillede summa suurendamine vales järjekorras	44	9,3%
Viga 8 – vigane väljundlause või väljundlause puudu	40	8,4%
Viga 9 – vale loenduri liidetav või loenduri liidetav puudu	31	6,5%

Kõige sagedasem oli viga 1, vale lillede summa liidetava väärtuse määramine, mida märgiti andmetabelisse 68 korda (14,3%). Seejuures vale loenduri liidetav ehk viga 9 esines vaid 31 korda (6,5%). Siit järeldub, et suurem osa lahendajatest on saanud tsükli küll tööle, kuid pole osanud välja mõelda moodust, kuidas seda tsüklit kasutades saaks ka lillede summa õigesti kokku arvutatud.

Sageduselt teine oli viga 2 ehk vale lillede summa algse väärtuse omistamine enne tsüklit, mis esines 66 korda (13,9%). Võrreldes veaga 4, milleks oli vale loenduri algse väärtuse omistamine, esines seda viga siiski vähem kordi, 46 korda (9,7%). Selle põhjal saab öelda, et jällegi tekitas lillede summa osalejatele rohkem probleemi kui loendur. Samas on teada, et muutujate algsed väärtused on seotud vastavate muutujate suurendamisega ehk eelnevalt mainitud vigadega 1 ja 9. Kui summa algne väärtus on vale, siis ei saa ka lõplik summa tulla õige, isegi kui summa suurendamise liidetav on õige ja samamoodi tsükli läbikäimise puhul. Näiteks sellise lahendustüübi puhul, kus puudub *if*-lause, mis kontrollib summasse liidetava arvu väärtust (vt alapeatükk 4.3, Tabel 5: Lahendus 2, Lahendus 4).

Sageli esines ka olukordi, kus lillede summa suurendamine töötas vastavalt tsükli tüübile valesti ja see põhjustas seisundi, kus hakati otsima lahendust algse summa muutuja väärtuse muutmisest, mille tõttu õppurid sai lõpuks mõlemad vead andmetabelisse kirja. Loenduriga esines sarnaseid situatsioone. Kokkuvõttes, kui esines üks nendest neljast (viga 1, 2, 4, 9) väljatoodud vigadest, oli ka lõpptulemus vale. Sellest tulenevalt annab autor ülesande teksti täiendamiseks soovituse. Kuna ülesanne koosneb kahest erinevast muutuja suurendamisest, siis võimalik, et õppuritele võib olla abiks soovitus esmalt tsükkel tööle panna ja seejärel alles mõelda, kuidas summa suurendamine selles tsüklis toimuma peaks vastavalt ülesande tingimustele. Samuti oleks abi viitest õppematerjalidele, mis tuletaks meelde, et abimaterjali

saab ka ülesande lahendamise ajal kasutada ning kus muuhulgas tutvustatakse näiteks kuidas tsükli sees *if*-tingimusi kasutada.

Esines ka *print*-lause väljastamist vales kohas ehk viga 3, millele tehti 61 märget (12,8%) andmetabelisse. Enamasti tähendas see viga seda, et väljundlause oli paigutatud tsükli sisse, mitte tsüklist väljapoole, mis põhjustas summa väljastamise igal tsükli läbimise korral ning mida automaatkontroll õigeks ei lugenud, kuna ülesanne nõudis vaid lõpliku lillede summa väärtuse väljastamist.

Veel tasub välja tuua vead seoses *while*-tsükli tingimusega: viga 5, vale võrdlusoperaatori kasutamine tsükli tingimuses, millele tehti 46 märget (9,7%), ja viga 6, vale väärtusega võrdlemine tsükli tingimuses, millele tehti 45 märget (9,5%).

Lisaks esines viga 7, summa ja loenduri suurendamist vales järjekorras, mida loeti kokku 44 korda (9,3%). Sellegipoolest, ei saa kindlalt öelda, kumb lahendusviis on õige, kas summa suurendamine enne loenduri suurendamist või vastupidi, sest mõlemat viisi kasutades on võimalik ülesanne õigesti ära lahendada. Esines rohkem lahendusi, kus loenduri suurendamine toimus alles pärast summa suurendamist. Kasutades aga eelnevale vastupidist lahendust, tuleb õige tulemuse saamiseks loenduri algväärtuseks omistada -1 või summa algseks väärtuseks omistada 1.

Leidus ka väljundlauseid, kus väljastati vale muutuja väärtus või väljundlause oli puudu ehk viga 8, neid oli kokku 40 (8,4%). Viga esines enamasti siis, kui programm ei töötanud veel õigesti.

Lisaks esines vigu, mida tehti vähem kui 10 korda (alla 2%) (vt Lisa VI): summa või loenduri ülevääртustamine vales kohas, trüki- või süntaksivead, vigane *if*-tingimus, lahendus *for*- tsükliga ja probleem tüüpidega.

Ülesande lahenduste vigade analüüs näitas, et suurema esinemise arvuga vigu oli tehtud peaaegu võrdselt ja ükski viga ei paistnud teistest selgelt rohkem silma. Ühtlasi vastanduvad selle ülesande lahenduste analüüsi tulemused Garneri jt koostatud Java sissejuhatava programmeerimise kursuse analüüsi tulemustele, kus selgus, et tsüklite teema seminar oli üks raskematest algajate programmeerijate jaoks [21]. *While*-tsüklit kasutava „Lillede arv" analüüsi ja vigade arvu põhjal võib öelda, et see ülesanne oli osalejatele lihtsam kui ülesanne „Spämm". Kuigi teemade poolest peaks tingimuslausete teema olema siiski lihtsam, võis praeguseid analüüsi tulemusi mõjutada ülesannete keerukus.

Kokkuvõte

MOOCid on viimaste aastate jooksul kogunud üha enam tuntust nii Eestis kui ka välismaal. Tartu Ülikool pakub kolme erinevat programmeerimise MOOCi: „Programmeerimisest maalähedaselt", „Programmeerimise alused" ja „Programmeerimise alused II". Vaatamata sellele, et varem on püütud Tartu Ülikooli programmeerimise MOOCil osalejatel ülesannete lahendamisel tekkinud küsimusi ja muresid erinevatel viisidel lahendada ja ülesannete raskusastet ühtlustada, näiteks murelahendaja abil, on mõned programmeerimise ülesanded siiski osalejatele raskemad kui teised.

Antud töös analüüsiti kursuse „Programmeerimise alused" kahe kohustusliku ülesande lahendusi, kus esimene ülesanne oli seotud tingimuslausetega ja teine *while*-tsükliga. Töö eesmärgiks oli välja selgitada analüüsitud ülesannete võimalikud lahendustüübid ja esinenud vead ning selle põhjal anda soovitusi kursuse materjalide täiendamiseks. Eesmärgist lähtuvalt esitas autor kolm uurimisküsimust.

Esimene uurimusküsimus oli, milliseid lahendusi esines analüüsitud ülesannete lahenduste hulgas. „Spämm" ülesande puhul liigitati lahendused *if*-harude hulga ja paigutuse järgi. Ülesande „Lillede arv" puhul liigitati lahendused *while*-tsükli tingimuse, *if*-tingimuse olemasolu ja klientide summeerimise järgi. Tingimuslausete teema ülesande „Spämm" lahenduste hulgas oli neli erinevat lahenduse tüüpi ning *while*-tsükli ülesande „Lillede arv" lahenduste hulgas oli viis erinevat lahenduse tüüpi. Lisaks esines mõlema ülesande puhul lahendusi, mida ei saanud liigitada ühegi lahendustüübi hulka.

Teine uurimisküsimus oli, milliseid vigu tehti analüüsitud ülesannete lahendamisel. Ülesande „Spämm" puhul sõnastati kokku 12 erinevat ja ülesande „Lillede arv" puhul 9 erinevat veatüüpi. Ülesande „Spämm" puhul tehti vigu seoses loogiliste operaatoritega, võrdlusoperaatoritega, *if*-tingimuses kasutatud väärtustega, tingimuste kombineerimisega, tingimuste kombineerimisega *if*-harudega ja sulgudega, muutuja tüüpidega ja väljundiga. Ülesande „Lillede arv" puhul tehti vigu seoses *while*-tsükli tingimuse ja selle võrdlusoperaatoriga, loenduri ja summa algse väärtuse omistamisega, loenduri ja summa väärtuse suurendamisega ning väljundlause sisu ja asukohaga. Mõlema ülesande lahendustes esines ka trüki- või süntaksivigu.

Kolmas uurimisküsimus oli, milliseid vigu esines analüüsitud ülesannete lahendustes kõige rohkem. Ülesandes „Spämm" lahendustes esines kõige rohkem viga seoses valede loogiliste operaatorite kasutamisega, mida tehti 486 korda ja mis moodustas 22,3% kõikidest selles

ülesandes tehtud vigadest. Ülesandes „Lillede arv” lahendustes esines kõige rohkem viga, milleks oli vale summa liidetava kasutamine või summa liidetava puudumine, mida tehti 68 korda ja mis moodustas 14,3% kõikidest selles ülesande lahendustes tehtud vigadest.

Leitud tulemuste põhjal pandi antud töö raames kirja mõned soovitused ülesannete tekstide kohta. Ülesande „Spämm” puhul soovitati ära nummerdada ülesande põhitingimused. Ülesande „Lillede arv” juures soovitati anda lahendajatele soovitus proovida ülesannet lahendada kahes osas: kõigepealt kirjutada kood töötava tsükli jaoks ja alles seejärel summeerida lilled selle tsükli abil. Mõlema ülesande juures soovitati lisada ülesande teksti ka vastavate teemade materjalide viited.

Antud tööd saab kasutada programmeerimise MOOCide materjalide ja ülesannete täiendamisel või muutmisel kui ka uute MOOCide loomisel, keskendudes rohkem töös analüüsitud ülesannete teemade sagedasemate vigade ennetamisele.

Viidatud kirjandus

[1] Hone, K. S., El Said, G. R. Exploring the factors affecting MOOC retention: A survey study. *Computers & Education*, 2016, vol. 98, pp. 157-168.

[2] Tõnisson, E. Programmeerimisest maalähedaselt. Juured. *Tartu Ülikooli e-õppe ajakiri*, 2015.

http://etu.ut.ee/kevad-2015/programmeerimisest-maalahedaselt/ (03.03.2018)

[3] Tartu Ülikooli programmeerimise MOOCide võrdlus.

https://courses.cs.ut.ee/2018/progmaa/spring/Main/Maalahealusedvordlus (03.03.2018)

[4] Tartu Ülikooli ÕIS.

www.ois.ut.ee (23.04.2018)

[5] Deng, J. Research on Higher Vocational Students' Acceptance and Use of MOOC in Web Software Development Course. *Boletín Técnico,* 2017, vol. 55, no. 7, pp. 689-695.

[6] Shah, D. By The Numbers: MOOCS in 2016. Class Central, 2016.

https://www.class-central.com/report/mooc-stats-2016/ (28.02.2018)

[7] Programmeerimisest maalähedaselt 2017. aasta kevadel.

https://courses.cs.ut.ee/2017/progmaa/spring (25.02.2018)

[8] Programmeerimise alused 2017. aasta sügisel.

https://courses.cs.ut.ee/2017/eprogalused (25.02.2018)

[9] Programmeerimise alused II 2017. aasta kevadel.

https://courses.cs.ut.ee/2017/eprogalused2/spring (20.02.2018)

[10] Virtual Programming Lab (VPL).

http://vpl.dis.ulpgc.es/index.php/about/what-is-vpl (01.03.2018)

[11] Hollo, H. Programmeerimise e-kursusel osalejate küsimuste analüüs ja selle põhjal murelahendajate koostamine. TÜ arvutiteaduse instituudi bakalaureusetöö. 2016.

[12] Tartu Ülikooli Moodle keskkond.

https://moodle.ut.ee (10.05.2018)

[13] Lepp, M., Luik, P., Palts, T., Papli, K., Suviste, R., Säde, M., Tõnisson, E. MOOC in Programming: A Success Story. *Proceedings of the International Conference on e-Learning (ICEL)*, 2017, pp. 138-147.

[14] Liyanagunawardena, T. R., Parslow, P. and Williams, S. Dropout: MOOC participants'perspective. EMOOCs 2014, the Second MOOC European Stakeholders Summit, 2014, pp. 95-100.

[15] Alturki, R. A. Measuring and Improving Student Performance in an Introductory Programming. *Informatics in Education*, 2016, vol. 15, no. 2, pp. 183-204.

[16] Wilson, B.C., Castillo, J. C., Favela, J., Prieto, M. E. A study of factors promoting success in computer science including gender differences. *Computer Science Education*, 2002, vol. 12, pp. 141-164.

[17] Courney, M., Teague D., Thomas, R. N. Engaging Students in Programming. *Proceedings 12th Australasian Computing Education Conference*, 2010.

http://crpit.com/confpapers/CRPITV103Corney.pdf (23.03.2018)

[18] Murphy, E., Crick, T. and Davenport, J.H. An Analysis of Introductory Programming Courses at UK Universities, *The Art, Science, and Engineering of Programming*, 2017, vol. 1, no. 2, Article 18.

[19] Ala-Mutka, K., Järvinen, H.-M., Lahtinen, E. A Study of the difficulties of novice programmers. *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2005, ACM SIGCSE Bulletin, vol. 37, no. 3, pp. 14-18.

[20] Vizcaíno, A., Contreras, J., Favela, J., Prieto, M. An Adaptive, Collaborative Environment to Develop Good Habits in Programming. *Intelligent Tutoring Systems (ITS), 5th International Conference*, 2000, LNCS, vol. 1839, pp. 262-271.

[21] Garner, S., Haden, P., Robins, A. My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems. *Proceedings of the 7th Australasian conference on Computing education*, 2005, vol. 42, pp. 173-180.

[22] Tartu Ülikooli ATI Courses keskkond.

https://courses.cs.ut.ee/ (25.03.2018)

[23] Mitmeosaline tingimus. Loogilised tehted ja avaldised.

https://courses.cs.ut.ee/2017/eprogalused/Main/Tingimuslause3 (12.04.2018)

Lisad

I. Ülesande „Spämm“ andmetabel

Vea esinemise puhul on märgitud iga lahenduse puhul vastavasse vea lahtrisse 1. Samuti on iga lahenduse puhul andmetabelisse märgitud vastav lahenduse tüüp tulbas „Lahendus“. Osalejate nimed on asendatud tähega „X“.

Järgnevalt lingilt leiab arvestatud lõpptulemustega lahenduste vigade andmetabeli:
https://docs.google.com/spreadsheets/d/1YpBEwoXH5GjL47fs_mjkgplaEj3jWM2QU96X0OM4p3o/edit#gid=616511857

Järgnevalt lingilt leiab mittearvestatud lõpptulemustega lahenduste vigade andmetabeli:

https://docs.google.com/spreadsheets/d/1YpBEwoXH5GjL47fs_mjkgplaEj3jWM2QU96X0OM4p3o/edit#gid=910824741

II. Ülesande „Lillede arv“ andmetabel

Vea esinemise puhul on märgitud iga lahenduse puhul vastavasse vea lahtrisse 1. Samuti on iga lahenduse puhul andmetabelisse märgitud vastav lahenduse tüüp tulbas „Lahendus“. Osalejate nimed on asendatud tähega „X“.

Järgnevalt lingilt leiab arvestatud lõpptulemustega lahenduste andmetabeli:

https://docs.google.com/spreadsheets/d/1YpBEwoXH5GjL47fs_mjkgplaEj3jWM2QU96X0OM4p3o/edit#gid=487089223

Järgnevalt lingilt leiab mittearvestatud lõpptulemustega lahenduste andmetabeli:

https://docs.google.com/spreadsheets/d/1YpBEwoXH5GjL47fs_mjkgplaEj3jWM2QU96X0OM4p3o/edit#gid=819997408

III. Ülesande „Spämm” lahendused

Järgnevas tabelis saab tutvuda lahendustüüpide alla kuuluvate kõikide erinevate variatsioonidega.

Lahendus 1

Lahendus 1a

```
if pealkiri == "" or fail == "jah" and suurus > 1:
    print("Kiri on spämm.")
else:
    print("Kiri ei ole spämm.")
```

Lahendus 1b

```
if pealkiri != "" and (fail != "jah" or suurus <= 1):
    print("Kiri ei ole spämm.")
else:
    print("Kiri on spämm.")
```

Lahendus 1c

```
if (fail == ("jah") and suurus > 1.0) or pealkiri == str():
    print("Kiri on spämm")
elif fail == ("ei") or (fail == ("jah") and suurus <= 1.0):
    print("Kiri ei ole spämm")
```

Lahendus 2

Lahendus 2a

```
if pealkiri == "":
    print("Kiri on spämm.")
elif fail == "jah" and suurus > 1:
    print("Kiri on spämm.")
else:
    print("Kiri ei ole spämm.")
```

Lahendus 2b

```
if not pealkiri:
    print("Kiri on spämm")
elif fail == "ei":
    print("Kiri ei ole spämm")
elif fail == "jah" and suurus < 1.1:
    print("Kiri ei ole spämm")
```

```
else:
    print("Kiri on spämm")
```

Lahendus 3

Lahendus 3a

```
if len(pealkiri) == 0:
    print("Kiri on Spämm")
elif fail == "jah":
    if suurus > 1:
        print("Kiri on spämm")
    else:
        print("Kiri ei ole spämm")
else:
    print("Kiri ei ole spämm")
```

Lahendus 3b

```
if suurus > 1 and fail == "jah":
    print("Kiri on spämm")
    if suurus >= 1 and fail == "ei":
        print("Kiri ei ole spämm")
else:
    if pealkiri == "":
        print("Kiri on spämm")
    else:
        print("Kiri ei ole spämm")
```

Lahendus 4

Lahendus 4a

```
if suurus > 1 and fail == "jah" and pealkiri == "":
    print("Kiri on spämm")
elif suurus > 1 and fail == "ei" and pealkiri == "":
    print("Kiri on spämm")
elif suurus > 1 and fail == "jah" and pealkiri == pealkiri:
    print("Kiri on spämm")
elif suurus > 1 and fail == "ei" and pealkiri == pealkiri:
    print("Kiri ei ole spämm")
elif suurus <= 1 and pealkiri and fail == "jah":
    print("Kiri ei ole spämm")
elif suurus <= 1 and pealkiri and fail == "ei":
    print("Kiri ei ole spämm")
```

```
elif suurus <= 1 and pealkiri == "" and fail == "ei":
    print("Kiri on spämm")
elif suurus <= 1 and pealkiri == "" and fail == "jah":
    print("Kiri on spämm")
```

Lahendus 4b

```
if suurus <= 1.0 and pealkiri == str("Ülesanne 2.2") and
                                                    fail == "jah":
    print("Kiri ei ole spämm")
elif suurus<= 1.0 and pealkiri == str("Programmeerimine") and
                                                    fail == "jah":
    print("Kiri ei ole spämm")
elif suurus <= 1.0 and pealkiri == str("Kudumine") and fail ==
                                                        "jah":
    print("Kiri ei ole spämm")
elif suurus<= 1.0 and pealkiri == str("Kudumine") and fail =="ei":
    print("Kiri ei ole spämm")
elif suurus<= 1.1 and pealkiri == str("Programmeerimine") and
                                                      fail== "ei":
    print("Kiri ei ole spämm")
else:
    print("Kiri on spämm")
```

Lahendus 4c

```
if (pealkiri == "" or suurus > 1 and fail == "jah") or (pealkiri
                          == "" and suurus < 1 and fail == "jah"):
    print ("Kiri on spämm")
else:
    print ("Kiri ei ole spämm")
```

Muud lahendused

Lahendus a

```
print("Sisestage kirja suurus: ")
suurus = input()
suurus= float(suurus)

print("Sisestage kirja teema pealkiri: ")
pealkiri = input()

print("Kas kirjaga on kaasas fail?: ")
fail = input()

tyhi = str()
jah = str("jah")
```

```
a = 0
b = 0
c = 0
d = 0
e = 0
f = 0
if pealkiri == tyhi:
    a = 1
else:
    b = 0

if suurus > 1 and jah == fail:
    d = 1
else:
    e = 0

s = a + d + f

if s >= 1:
    print ("Kiri on spämm")
else:
    print("Kiri ei ole spämm")
```

Lahendus b

```
if (fail == "jah" and suurus <= 1) or fail == "ei":
    ManusKasOnKorras = True
else:
    ManusKasOnKorras = False
if pealkiri == "":
    PealkiriKasOnKorras = False
else:
    PealkiriKasOnKorras = True
if PealkiriKasOnKorras == True and ManusKasOnKorras == True:
   print("Kiri ei ole spämm")
else:
    print("Kiri on spämm")
```

IV. Ülesande „Lillede arv“ lahendused

Järgnevas tabelis saab tutvuda lahendustüüpide alla kuuluvate kõikide erinevate variatsioonidega.

Lahendus 1

Lahendus 1a

```
i = 0
summa = 0
while i <= kliendid:
    if i % 2 == 1:
        summa +=i
    i+=1
print("Lillede koguarv on" + str(summa))
```

Lahendus 1b

```
i = 1
summa = 1
while i < kliendid:
    if i % 2 == 0:
        summa = summa +1 + i
    loendur += 1
print("Lillede koguarv on " + str(summa))
```

Lahendus 1c

```
summa = 0
i = 1
while i <= kliendid:
    if kliendid % 2==0:
        kliendid -= 1
    summa += i
    i = i+2
print("Lillede koguarv on " + str(summa))
```

Lahendus 1d

```
summa = 0
i = 0
while i < klient:
    i += 1
    if i % 2 != 0:
        summa = summa + i
print("Lillede koguarv on " + str(summa))
```

Lahendus 1e

```
i = 0
a = 1
summa = 0
while i in range(kliendid):
    if i % 2 == 0:
        summa = summa + a
    else:
        a = a + 2
        i+=1
print("Lillede koguarv on " + str(summa))
```

Lahendus 2

Lahendus 2a

```
i = 1
summa = 0
while i <= kliendid:
    summa += i
    i += 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 2b

```
i = 1
lilled = -1
summa = 0
while i <= kliendid:
    i = i + 2
    lilled += 2
    summa += lilled
print("Lillede koguarv on " + str(summa))
```

Lahendus 2c

```
i = 1
summa = 1
while (i + 2) <= kliendid:
    i = i + 2
    summa = summa + i
print("Lillede koguarv on " + str(summa))
```

Lahendus 2d

```
i = 1
summa = 0
while kliendid >= i:
    i += 2
    summa += i - 2
```

```
print("Lillede koguarv on " + str(summa))
```

Lahendus 2e

```
kliendid = (kliendid+1) // 2
summa = 0
i = 0
lilli_ostjale = 1
while i < kliendid:
    summa = summa + lilli_ostjale
    lilli_ostjale = lilli_ostjale + 2
    i = i + 1
print("Lillede koguarv on " + str(summa))
```

Lahendus 3

Lahendus 3a

```
summa = 0
while kliendid > 0:
    if kliendid % 2 == 1:
        summa += kliendid
    kliendid -= 1
print("Lillede koguarv on " + str(summa))
```

Lahendus 3b

```
i = 1
summa = 0
while kliendid > 0:
    kliendid = kliendid -1
    if i % 2 == 1:
        summa += i
    i += i
print("Lillede koguarv on " + str(summa))
```

Lahendus 4

Lahendus 4a

```
summa= 0
while kliendid > 0:
    summa += kliendid
    kliendid -= 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 4b

```
i = 1
summa = 0
while kliendid > 0:
```

```
    kliendid = kliendid - 2
    summa += i
    i += 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 4c

```
tsüklite_arv = klientide_arv / 2
lillede_arv = 1
summa = 0

while tsüklite_arv > 0:
    tsüklite_arv -= 1
    summa += lillede_arv
    lillede_arv += 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 4d

```
if kliendid > 0:
    summa = 1
else:
    summa = 0
if kliendid == 1:
    print("Lillede koguarv on 1.")
i = 1
while kliendid > 2:
    i = i + 2
    summa = summa + i
    kliendid = kliendid - 2
print("Lillede koguarv on " + str(summa))
```

Lahendus 5

Lahendus 5a

```
lilled = 1
summa = 0
while lilled <= kliendid:
    summa = (kliendid // 2 + ostjad % 2) **2
    lilled = lilled + kliendid
print("Lillede koguarv on " + str(summa))
```

Lahendus 5b

```
arv = 1
summa = 0
while arv <= kliendid:
    if kliendid & 1:
        summa = ((kliendid+1)/2)**2
```

```
        print("Kogu lillede arv: " + str(int(summa)))
        break
    elif kliendid % 2 == 0:
        kliendid -= 1
        summa = ((kliendid+1)/2)**2
        print("Kogu lillede arv: " + str(int(summa)))
        break
```

Lahendus 5c

```
n = kliente
lilli = 0
while n > 0:
    n = n-1
    a = n//2+1

    if n%2 == 0:
        lilli = lilli+(1+(a-1)*2)
print("Lillede koguarv on ", lilli)
```

Muud lahendused

Lahendus a

```
i = 1
summa = 0
while i <= kliendid:
    i = i + 2
    summa = summa + 1
    while loendur <= kliendid:
        i = i + 2
        summa = summa + i - 2
print ("Lillede koguarv on " + str(summa))
```

Lahendus b

```
kliendid = int(kliendid)
mul = 0
total = 0
count = 0
if kliendid > 0:
    count = round(kliendid / 2)
    while count > -1:
        if mul == 0:
            mul = mul + 1
            total = 1
            count = count - 2
        else:
            mul = mul + 2
```

```
            total = total + mul
            count = count - 1
    print(total)
else:
    print("Arv ei või olla negatiivne")
```

V. Ülesande „Spämm“ vead

Järgnevalt on välja toodud kõik ülesandes „Spämm” esinenud vead koos rohkemate näidetega. Veatüübid olid järgnevad:

1. **Valede loogiliste operaatorite kasutus** – teise tingimuse (faili olemasolu ja kirja suurus) osad olid ühendatud vale operaatori abi või ülesande esimene ja teine tingimus olid omavahel kombineeritud vale operaatori abil või operaator puudus. Näiteks:
   - `if suurus > 1 or fail == "jah" and pealkiri == ""`
   - `if suurus > 1 or fail == "jah" or pealkiri == ""`
   - `if suurus > 1 and fail == "jah" and pealkiri == ""`
2. **Valede väärtustega võrdlemine** – selle vea alla kuulusid: pealkirja võrdlus vale väärtusega, kirja olemasolu võrdlus vale väärtusega, kirja suuruse võrdlus vale väärtusega. Näiteks:
   - `pealkiri == " "`
   - `pealkiri == "True"`
   - `pealkiri == True või pealkiri == False`
   - `pealkiri == input()`
   - `pealkiri == "ENTER"`
   - `pealkiri == "Ülesanne 2.2"`
   - `pealkiri == "späm"`
   - `pealkiri(False)`
   - `pealkiri == "0"`
   - `pealkiri is None`
   - `pealkiri == null`
   - `pealkiri = str`
   - `pealkiri != str()`
   - `fail == ""`
   - `fail != input("ei")`
   - `fail == True`
   - `fail == "ei" or "jah"`
   - `fail == "ei"`
   - `fail[-3] == "jah"`
   - `suurus < 0.9`
   - `suurus > 1.1`
   - `suurus > "spämm"`
   - `suurus > 1 == "spämm"`
3. **Vale võrdlusoperaatori kasutus** – kirja suuruse tingimuses vale võrdlusoperaator kasutamine. Näiteks:
   - `suurus < 1`

- `suurus == 1`
- `suurus <= 1`
- `suurus >= 1`

4. **Tingimuste vale kombineerimine** – ülesande „Spämm" püstituses kirjeldatud pealkirja tingimus oli valesti kombineeritud teise ülesandes esitatud kirja suuruse ja faili tingimusega. Näiteks:

```
if suurus > 1 and pealkiri == "" or fail == "jah":
```

5. **Vastupidine väljund vastavalt tingimusele** – *if*- või *else*-tingimusele oli vastavusse pandud vastupidine väljund. Näiteks:

```
if pealkiri == "" or fail == "jah" and suurus > 1:
    print("Kiri ei ole spämm.")
else:
    print("Kiri on spämm.")
```

6. **Tingimuste vale grupeerimine sulgude või *if*-harude abil** – valesti tingimuste grupeerimine, seejuures ka tihti tingimuste kordamine, *if*- ja *elif*-tingimuste või sulgude abil. Näiteks:

- `if (pealkiri == "" and suurus > 1) and fail == "jah"`

- ```
  if suurus > 1 and pealkiri == "":
      print("Kiri on spämm")
  elif fail == "jah":
      print("Kiri on spämm")
  ```

- ```
  if suurus > 1:
      if pealkiri == "":
          print("Kiri on spämm")
      else:
          print("Kiri ei ole spämm")
  ```

7. **Automaatkontrolli nõudmistest tulnud vead** – lahendus ei läinud automaatkontrollist läbi automaatkontrolli nõudmiste tõttu. Näiteks:

- ```
  if pealkiri == "" or fail == "Jah" and suurus > 1:
      print("Kiri on spämm")
  else:
      print("Kiri ei ole spämm")
  ```

- ```
  if pealkiri == "" or fail == "jah" and suurus > 1:
      print("See on spämm")
  else:
      print("See ei ole spämm")
  ```

8. **Tingimuste osaline puudumine** – ülesande püstituses esitatud tingimustest oli mõni puudu. Näiteks:

- ```
  if pealkiri == "" or suurus > 1:
      print("Kiri on spämm")
  else:
      print("Kiri ei ole spämm")
  ```
- ```
  if pealkiri == "":
      print("Kiri on spämm")
  else:
      print("Kiri ei ole spämm")
  ```

9. **Probleem muutujate tüüpidega** – tingimuse päises püüti võrrelda erinevat tüüpi muutujaid. Näiteks:

   - ```
     suurus = int(float(input("Sisesta faili suurus: ")))
     pealkiri = input("Sisesta pealkiri: ")
     fail = input("Kas fail on olemas?")
     if pealkiri == "" or fail == False and suurus > "1":
             print("Kiri on spämm")
     ```

10. **Trüki- või süntaksivead.** Näiteks:

    - ```
      suurus = float(input("Sisesta faili suurus: ")
      ```
    - ```
      suurus = float(input("Sisesta faili suurus: "))
      pealkiri = input("Sisesta pealkiri: ")
      fail = input("Kas fail on olemas?")
      if pealkiri == "" or file =="jah" and suurus > 1:
          print("Kiri on spämm")
      else:
          print("Kiri ei ole spämm")
      ```

11. **Küsitud sisendite üleväärtustamine enne *if*-plokki** – enne programmi *if*-tingimustesse sisenemist toimus kasutaja käest küsitud sisendite üleväärtustamine. Näiteks:

    ```
    suurus = float(input("Sisesta faili suurus: "))
    pealkiri = input("Sisesta faili pealkiri: ")
    fail = input("Kas fail on olemas?")
    fail = "jah"
    pealkiri == "Programmeerimine"
    suurus = 0.5
    ```

12. **Probleemid muutujatega.** Näiteks:

    ```
    pealkiri = " "
    suurus = float()
    fail = "jah"
    print(input("Sisestage kirja teema pealkiri: "))
    print(input("Sisestage kirja suurus: "))
    print(input("Kas kirjaga on kaasas fail?"))
    if pealkiri == str(" ") or (suurus > 1 and fail == "jah"):
        print("Kiri on spämm")
    ```
    ```

```
else:
    print("Kiri ei ole spämm")
```

VI. Ülesande „Lillede arv“ vead

Lisaks töö põhiosas välja toodud vigadele esines ülesandes “Lillede arv” veel järgmisi vigu:

1. **Summa või loenduri väärtuse üleväärtustamine vales kohas.** Näiteks:

```
i = 0
summa = 0
while i < kliendid:
    i = i + 1
    if i % 2 == 0:
        summa = 0
    else:
        summa = i + summa
```

2. **Trüki- või süntaksivead.** Näiteks:

- ```
  while i <= kliendid
      if i % 2 == 1:
          summa += i
      i += 1
  ```
- ```
  while ostja<= kliendid:
      if ostja % 2! = 0:
  ```
- ```
  print("Lilledekoguarv on: " + str(lillede_arv)
  ```

3. **Vigane *if*-tingimus.** Näiteks:

```
while i <= kliendid:
    if i // 2 == 1:
        summa += i
    i += 1
```

4. **Probleem tüüpidega.** Näiteks:

```
print("Lillede koguarv on: " + summa + ".")
```

5. **Lahendus *for*-tsükliga.** Näiteks:

```
summa = 0
for i in range(1, kliendid + 1):
    if i % 2 > 0:
       summa = summa + i
```
```

VII. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Helen Hendrikson**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
MOOCi „Programmeerimise alused“ ülesannete lahenduste analüüs,
(*lõputöö pealkiri*)

mille juhendaja on Marina Lepp,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **14.05.2018**