TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Sandra Puusepp

Programmeerimisülesannete kogu gümnaasiumi valikkursusele „Tarkvaraarendus“

Bakalaureusetöö (9 EAP)

Juhendajad: Eno Tõnisson
Tauno Palts

Tartu 2019

Programmeerimisülesannete kogu gümnaasiumi valikkursusele „Tarkvaraarendus“

Lühikokkuvõte:

Programmeerimise õpetamisel tuleb tähelepanu pöörata probleemilahenduse oskusele. Selle omandamisel mängib ülesannete lahendamine olulist rolli. Bakalaureusetöö eesmärk on koostada programmeerimisülesannete kogu gümnaasiumi uuele valikkursusele „Tarkvaraarendus“. Osa ülesandeid piloteeriti kooli programmeerimise tundides. Töö esimeses pooles antakse ülevaade programmeerimise algõppest, Eesti koolides programmeerimisainete läbiviimisel kasutatavatest materjalidest ja programmeerimis-ülesannete koostamise teooriast. Teises pooles kirjeldatakse ülesannete koostamise ja piloteerimise metoodikat ning antakse ülevaade valminud ülesannetest ja nende modifitseerimisest.

Võtmesõnad:

Programmeerimisülesanded, õppematerjalid, programmeerimise algõpe

CERCS: P175 Informaatika, süsteemiteooria, S281 Arvuti õpiprogrammide kasutamise metoodika ja pedagoogika

A Collection of Programming Tasks for the Secondary School Elective Course "Software Development"

Abstract:

When teaching programming, developing problem solving skills must be one of the main areas of focus. Solving programming tasks is an integral part of achieving this skill. The aim of the Bachelor's thesis is to create a collection of programming tasks for the new upper-secondary school elective course "Software Development". Some of the created tasks have also been piloted in the programming classes of a secondary school. The first half of the thesis focuses on teaching programming to novices, the teaching materials used in the programming courses of Estonian schools, and the theoretical aspect of creating programming tasks. The second half of the thesis gives an overview of the method used for creating and piloting the tasks, the created tasks themselves and the modifications made.

Keywords:

Programming tasks, teaching materials, teaching programming to novices

CERCS: P175 Informatics, systems theory, S281 Computer-assisted education

Sisukord

1. Sissejuhatus 6
2. Programmeerimise õpetamine algajatele 8
2.1 Lühiülevaade arvutialaste ainete õpetamisest Eesti üldhariduskoolides 8
2.1.1 Koolides kasutatavad programmeerimise õppematerjalid 9
2.2 Programmeerimise õpetamine 10
2.3 Algajate käitumismustrid programmeerimisel 12
2.4 Python programmeerimise õpetamisel 14
3. Programmeerimisülesannete koostamine 16
3.1 Ülesande sõnastus ja huvi tekitamine 16
3.2 Ülesannete keerukus 16
3.3 Koolides kasutatavate materjalide programmeerimisülesanded 19
3.3.1 Valikkursuse „Rakenduste loomise ja programmeerimise alused" ülesanded .. 19
3.3.2 MOOCi „Programmeerimise alused" ülesanded 20
3.3.3 Kursuse „Teeme ise arvutimänge – algus" ülesanded 22
3.4 Kokkuvõte programmeerimise õpetamisest ja ülesannete koostamisest 22
4. Metoodika 24
4.1 Eeltöö 24
4.2 Ülesannete koostamise protsess 24
4.3 Ülesannete katsetamine 25
5. Tulemused ja arutelu 26
5.1 Teemade jaotus 26
5.2 Ülesannete ülesehitus 28
5.3 Tagasiside ja parandused ülesannetele 31
5.4 Võimalikud edasiarendused 34

6. Kokkuvõte 35
7. Viidatud kirjandus 36
Lisad 41
I. Ülesandekogu 41
II. Õpetaja koostatud tööleht ülesande „2.4 Hinnete tabel“ põhjal 57
III. Litsents 60

1. Sissejuhatus

Informaatika ja sealhulgas programmeerimise õpetamine üldhariduskoolides on muutumas järjest tähtsamaks [1]. Programmeerimisainete õpetamine varieerub erinevate koolide vahel õpetavate programmeerimiskeelte, kasutatavate õppematerjalide ja ainete sisu poolest [2, 3]. Programmeerimise õppimise käigus ei õpita ainult programmide kirjutamist, testimist ja silumist, vaid omandatakse ka keeruliste probleemide lahendamise oskus. Sellise mõtlemise saavutamiseks ei piisa ainult teooria lugemisest, vaid peab lahendama ka mitmeid ülesandeid [4]. Ülesanded peavad lisaks tehnilisele poolele suutma õpilases tekitada tahtmist neid lahendada. Motivatsiooni aitab luua näiteks võimalus ülesannetega suhestuda [5]. Selles bakalaureusetöös keskendutaksegi programmeerimise ülesannetele gümnaasiumis.

Hetkel pakutakse koolides erinevaid arvutialaseid aineid ja huviringe, mis tegelevad programmeerimise, robootika, kontoritarkvara, multimeedia vms õpetamisega [4]. Programmeerimist tutvustakse üldiselt mittekohustuslike ainete või huviringide raames. Gümnaasiumi riiklikus õppekavas [6] toodud programmeerimisega seotud valikkursusi on kaks: „Mehhatroonika ja robootika“, kus õpitakse programmeerima mikrokontrollereid, ja „Rakenduste loomise ja programmeerimise alused“, mille käigus omandatakse programmeerimise põhikontseptsioonid ja mõisted. Programmeerimise õppele koolides on plaanis läheneda mõnevõrra süsteemsemalt ning selle raames valmivad gümnaasiumitele uued programmeerimise valikkursused „Programmeerimine“ ja „Tarkvaraarendus“, kus kasutatakse õppematerjalidena digitaalselt kättesaadavad õpikud „Programmeerimine“ ja „Tarkvaraarendus“ [7].

Bakalaureusetöö eesmärk on luua ülesannete kogu kursusele „Tarkvaraarendus“. Selle kursuse õpik [8] on alles piloteerimisel ning sisaldab hetkel teooriat, näiteid ja enesekontrolli küsimusi loetud teemade kohta, kuid puuduvad ülesanded. Nagu eelnevalt mainitud, on need vajalikud, sest probleemilahenduse oskuse saavutamiseks tuleb lahendada ülesandeid. Töö uurimuslikus osas tutvutakse programmeerimise õpetamisega koolides ja ülesannete koostamisega ning antakse ülevaade juba olemasolevatest koolides kasutatavatest programmeerimisülesannetest. Praktilise osa raames antakse ülevaade loodud ülesannete kogust ja ülesannete loomise protsessist. Ülesanded on teema ja

raskusastme poolest vastavuses kursuse „Tarkvaraarendus“ läbitavate teemade ja nende raskusastmega.

2. Programmeerimise õpetamine algajatele

2.1 Lühiülevaade arvutialaste ainete õpetamisest Eesti üldhariduskoolides

2017. aastal viis Praxise poliitikauuringute keskus läbi uuringu [2], mis annab ülevaate digioskuste[1] (infootsing Internetist, kirjatükkide koostamine, programmeerimine jne) õpetamisest üldhariduskoolides ja lasteaedades. Läbiviidud uuringu järgi on Eesti koolides digioskuste, sh ka programmeerimise õpetamine, varieeruv ning IT-oskusi hakatakse õpetama eraldi ainena erinevates kooliastmetes. Sellest tulenevalt erinevad ka õpetatavate ainete sisu ja tasemed. Internetipõhise infootsingu ja arvutis kirjatükkide koostamise ning vormistamise õpetamine on levinuim, kuid koolide ainekavades leidub ka programmeerimisega seotud tegevusi. Kätlin Viilukase bakalaureusetöös [3] küsitleti 22 Eesti põhikooli õpetajat 23 arvutialase õppeaine kohta. Selgus, et programmeerimist õpetati peamiselt huviringides ja tutvustati teemana üksikutes arvutiõpetuse/informaatika tundides.

Koolide programmeerimiskursuste ja -ainete erinevat sisu illustreerivad Puniste bakalaureustetöö [9] tulemused, mille käigus tuli välja, et 16 Eesti gümnaasiumites õpetatava programmeerimiskursuse peale õpetati kokku 14 erinevat programmeerimis-keelt. Vajaka jääb spetsiaalse ettevalmistusega informaatikaõpetajatest [10]. Praxise uuringu järgi, mille läbiviimiseks küsitleti üle 1500 õpetajat, hindavad õpetajad oma probleemilahenduse ning rakenduste ja programmide loomise oskust kehvaks. Sellele vaatamata ei ole digioskuste osas enesetäiendusega tegelenud ja selle vastu ei tunne huvi vastavalt 63% ja 53% õpetajatest.

Uuringus „Mis saab Eesti haridusest?“ [10] käsitletakse arvutiõpetuse all programmeerimist ja programmide loomist. Olulise märkusena on ära toodud, et riiklikud tasemetööd ja eksamid avaldavad ainetele ning õppematerjalidele normeerivat ja kontrollivat mõju, kuid informaatikaainetes need puuduvad. Õppematerjalidena kasutatakse peamiselt digitaalselt kättesaadavaid või õpetajate endi poolt loodud materjale. See on läbiv joon IT-ainete õpetamisel. Praxise uuringu järgi peab tervelt kolmandik küsitletud õpetajatest õppematerjalide puudust või vähest kättesaadavust üheks suuremaks

[1] Uuringus [2] peeti digioskuste all silmas informaatika õppekavas ja õpilaste digipädevuse mudelis välja toodud oskusi ning tehnoloogiahariduse seotud oskusi.

takistusteks õpetamisel. Et õppematerjalidest ja õpetajate teadmistest jääb arvutialaste ainete õpetamisel vajaka, on kindlasti vajalik luua juurde õpivara, mis oleks koolides õpetatavate arvutialaste ainetega ja kursustega kooskõlas.

2.1.1 Koolides kasutatavad programmeerimise õppematerjalid

Programmeerimise õppematerjale leidub veebis hulganisti, kuid suurem osa neist on ingliskeelsed. Viilukase bakalaureusetöös [3] on analüüsitud 22 arvutiõpetaja vastuseid küsimusele „Millises keeles õppematerjale eelistate?“, mille käigus selgus, et suur osa eelistavad ainult eestikeelseid materjale ning ingliskeelsed pigem kohandatakse ümber emakeelde. Eestikeelsete programmeerimismaterjalide loomine ja kasutamine on tähtsal kohal eesti keele säilimisel ja arendamisel.

Puniste bakalaureusetöö [9], kus uuriti 31 Eesti koolis õpetatavaid programmeerimis-kursusi, järgi on üks populaarsemaid keskkondi programmeerimise materjalide leidmiseks ProgeTiiger. See on 2012. aastal alguse saanud programm, mis keskendub koolidele programmeerimise tutvustamisega [11]. ProgeTiigri kogumikust [12] on võimalik leida eesti- ja ingliskeelseid materjale erinevatele kooliastmetele ning valida kaheksa erineva programmeerimiskeele vahel: PHP, Javascript, Python, Java, Ruby, JQuery, HTML/CSS ja Logo/UBCLogo.

Programmeerimiskeeles Python pakutakse ProgeTiigri veebilehel seitset eestikeelset materjali. Nende hulgas on viited mitmetele ülikoolide poolt pakutavatele e-kursustele, millel on avaliku ligipääsuga õppematerjalid. Näiteks tuuakse välja Tartu Ülikooli arvutiteaduse instituudi poolt pakutav õppematerjal „Python koolis“, mis põhineb PyGame mooduli abil mängude loomisel ja on mõeldud kasutamiseks III kooliastme ja gümnaasiumitaseme tava- või ringitundides [13]. Puniste uurimuse järgi kasutati õpetamisel lisaks ProgeTiigri materjalidele ka mugandatud kujul Tartu ja Tallinna ülikoolide ning mõningaid ingliskeelseid materjale.

Veebilehelt e-koolikott.ee [14], mis on keskkond õppematerjalide otsimiseks ja avaldamiseks, on leitav viide informaatika digiõpikutele I ja II kooliastmele [15]. Mõlemas õpikus tutvustatakse teemasid digitaalmeediast, programmeerimisest ja arvuti kasutamisega seotud ohtudest, kuid seda tehakse erineval raskustasemel vastavalt kooliastmele.

2.2 Programmeerimise õpetamine

2013. aasta uuring „Eesti IKT kompetentsidega tööjõu hetkeseisu ja vajaduse kaardistamine“ [16] ennustab, et aastaks 2020 on IKT-alase kõrgharidusega lõpetajate arv 4550 ringis, mis vastab nõudlusele. Vaatamata õppurite arvu kasvule ei ole IKT spetsialistide hariduse kvaliteet paranenud. Programmeerimine on üks osa IKT-alasest haridusest. Üldhariduskoolis programmeerimise õppimine annab IT kõrghariduse omandamisel eelise teiste üliõpilaste ees [10]. Kuigi programmeerimise õppimine võtab kaua aega, on sellele vaatamata õigeid meetodeid ja materjale kasutades võimalik ka kooliõpilastel see oskus omandada [4].

Põhjusi, miks keskkooliõpilased peaksid õppima programmeerimist, on uurinud Mara Saeli jt [4]. Selle uurimuse järgi arendab programmeerimine õpilastes oskust leida algoritm ülesande lahendamiseks ning panna see lühidalt ja konkreetselt koodina kirja. Seetõttu aitab programmeerimise õppimine kooli ajal kaasa ka teistes õppeainetes ja valdkondades, nagu matemaatika, füüsika, majandus, lingvistika jms. Seega Saeli jt arvates pakub programmeerimine õpilastele võimalust võtta ainet, mis ühendab endas mitme erineva õppeaine aspekte.

Mannila jt [17] toovad välja, et programmeerimine ongi oma olemuses just probleemilahendus ja selle tulemusena programmi loomine. Mannila jt jaotavad probleemilahenduse protsessi neljaks sammuks:

1. probleemi ehk ülesande mõistmine,
2. lahenduse leidmine,
3. algoritmi loomine ja
4. algoritmi rakendamine.

Programmeerimise õppimise käigus peaksid õpilased omandama algoritmilise mõtteviisi ning suutma aru saada ja osata seletada oma koodi kommenteerides ja/või esitledes [4]. Koodi lugemine tekitab algajatel programmeerijatel tihtipeale raskusi. Kui jääb selgusetuks, miks ja kuidas programm probleemi lahendab, ei saada hakkama ka koodi kirjutamise, testimise või silumisega [18]. Näiteks rekursiooni kirjutamisel ei oska algajad tihtipeale ette kujutada koodi erinevaid vaheetappe, nagu millal muutujate väärtused suurenevad või vähenevad, kuhu täpselt rekursiivne väljakutse lisada ja millal tagastatakse väärtus. Siinkohal oleks abiks nende sammude visualiseerimine, kuna see aitaks jälgida,

milliseid ülesandeid erinevad koodiosad täidavad ning kuidas need omavahel koostööd teevad [18].

Robins jt [19] eristavad kahte sorti teaduslikke töid. Ühtedes keskendutakse programmi mõistmisele (ingl *program comprehension*), kus osalejad peavad näitama, et nad on aru saanud, kuidas programm töötab, ja teistes programmide loomisele (ingl *program generation*), mis tähendab, et osalejad kirjutavad ülesannetele vastavaid programme või koodijuppe. Näitena tuuakse Risti mudel, mida on kirjeldanud Gray ja Boehm-Davis [20]. See põhineb tüüpiliste programmeerimisteadmiste implementeerimisel. Risti mudeli kohaselt loovad programmeerijad koodi kirjutades programmist „plaani", mis koosneb mitmetest skeemidest või osadest, kus iga osa kohta on toodud kood, koodi ja selle osade eesmärk ning eel- ja järeltingimused. Sellise kava genereerimine põhineb varasemastel kogemustel ning mida kogenenumaks programmeerija saab, seda rohkem hakkab ta neid osasid pigem meelde tuletama, kui uuesti looma. Selle mudeli seisukohalt alustavad algajad programmide loomist „alt-üles" ning samm-sammult täiendavad „plaani" ja lisavad skeeme juurde. Samas edasijõudnutel piisab vaid skeemide meeldetuletamisest ja seetõttu genereerivad oma kava „ülevalt-alla".

Programmi loomise ja sellest arusaamine on Robins jt [19] järgi seotud protsessid, sest koodi kirjutades või silumise käigus on vaja programmi analüüsida ning seda mõista. Sellele vaatamata on tehtud uuringuid, mis näitavad vähest seost programmi lugemisoskuse ja selle loomise vahel. Ahmadzadeh jt [21] uurisid algajate käitumismustreid programmide kirjutamisel ja silumisel. Osalejatele anti ette programm, kust tuli silumise käigus leida üles kompileerimisaegsed- ja loogikavead. Katse näitas, et 66% osalejatest, kes olid oskuslikud silujad, olid ka programmide kirjutamises pädevad. Samas kõigest 39% pädevatest programmeerijatest olid silumises kompetentsed. Etteantud programmi koodi mõistmine oli sellise tulemuse peamiseks põhjustajaks. Selle oskuse arendamine parandab üleüldist programmeerimispädevust.

Merrill [22] toob välja, et mitmed uuringud näitavad järjepidevalt, kuidas oskuse omandamiseks on vaja harjutamist. Programmeerimise õppimisel aitab seda teha tegelike programmide kirjutamine [19]. Merrill [22] jagab õpetamise strateegia neljaks komponendiks: „seleta", „näita", „küsi" ja „tee" (ingl *tell, show, ask, do*). Esimesed kaks annavad edasi informatsiooni õpitava teema kohta ning toovad illustreerivaid näiteid ning viimased kaks moodustavad praktilise osa. „Küsi" koosneb kontrollküsimustest, mis

aitavad õpilasel kinnistada just läbi töötatud materjali, näiteks tuletada meelde definitsioone, protsesssiks vajalikke samme jmt. „Tee“ osas saavad õpilased rakendada oma õpitud oskusi erinevaid ülesandeid lahendades. Seega praktiline ülesannete lahendamine on märkimisväärse tähtsusega. See aitab lisaks probleemilahenduse oskuse omandamisele, saada aimu tarkvaraarenduse protsessist ning teha tutvust programmeerimiskeelte ja –keskkondadega [23]. Robins jt [19] on kirjutanud, et kuigi algajatele oleks tihtipeale vaja tugevamat teoreetilist alust teatud keeleerisuste vallas (nagu tsüklid, tingimuslaused, järjendid ja rekursioon) on siiski kõige keerulisem omandada probleemilahenduse oskus. Siinkohal ongi abiks sage harjutamine koodi kirjutamise näol.

2.3 Algajate käitumismustrid programmeerimisel

Bakalaureusetöö raames koostatakse valikkursusele programmeerimisülesanded. Et saada ülevaade võimalikest raskustest, mis õpilastel võivad tekkida nende lahendamisel, tutvutakse selles alapeatükis algajate käitumisega programmeerimisülesannete lahendamisel. Suurem osa selle kohta läbiviidud uuringutest ei keskendu mitte Pythonile, vaid objektorienteeritud programmeerimisele Javas [24]. Selline on näiteks Jadudi uuring [25], kus keskenduti algajate käitumisele koodi kompileerimisel. Selgus, et väike hulk süntaksivigu põhjustas suure osa kõigist esinenud vigadest ning koodi kompileeritakse tihti ja lühikeste intervallide tagant. Kui koodi käivitamisel veateateid ei esine, on ajavahemik kompileerimiste vahel pikem (1–2 minutit). Süntaksivigade puhul, mis esinevat kõige sagedamini, toimub peale kiiret koodiparandust taaskompileerimine ligikaudu 30 sekundiga. Muude veateadete korral kompileeritakse koodi mitmeid kordi järjest ilma viga parandamata. Jadud pakub välja, et selline käitumine tuleneb suutmatusest „uskuda”, et kompilaator veateate annab.

Ihantola jt [26] võtsid aluseks varasemalt tehtud uurimusi ning nendes kasutatud meetoditega viisid läbi oma uurimuse. Nende hulgas oli ka Jadudi kompileerimisele keskenduv uuring, kuid Ihantola jt kasutasid Java asemel Pythoni harjutusi. Veateated, millega õpilased enim kokku puutusid, olid parsimise, taande, muutujanime ja teisendamisega seotud vead. Ka siin esines kõige sagedamini süntaksivigu, mis olid ära toodud parsimisvigadena. On leitud, et kui kasutada IDE, mis märgib ära esinenud süntaksivead ja pakub neile parandusi, on suur osa õppijate esitatud programmidest vigadeta [24].

Meieri magistritöö [24] üheks eesmärgiks oli leida käitumismustreid vaba juurdepääsuga e-kursuse programmeerimiskeeles Pythoni arvestusülesande lahendamises. Üle poolte õppijatest avas enne programmeerima hakkamist varasemate ülesannete lahendused ning osad tegid seda koodi kirjutamise käigus, et oleks võimalik sarnasest koodist leida abi. Ülesande lahendamist alustati erinevatest programmiosadest ning samuti varieerus käitumine koodi testimises. Üle 40% lahendajatest alustas koodi testimisega, kui kindel funktsioon või muu programmiosa kirjutatud sai, kuid peaaegu sama suur hulk inimesi käivitas koodi esmakordselt alles peale terve programmi valmis kirjutamist. Programmis esinenud vigade parandamiseks kasutati koodijuppide väljakommenteerimist, teistes failides katsetamist ja ligikaudu neljandikel juhtumitest silumist. Ühel juhul kasutati ka ainult silumist programmi käivitamise asemel.

Algajaid programmeerijaid on nende tüüpiliste käitumismustrite põhjal võimalik klassifitseerida. Jadud [25] kirjeldab ühte sellist viisi, kus õppijad jaotatakse peatujateks (ingl *stoppers*), liikujateks (ingl *movers*) ja äärmuslikeks liikujateks (ingl *extreme movers*). Peatujaid iseloomustab pidev abiküsimine iga sammu juures, kuid liikujad üritavad iseseisvalt probleemidest jagu saada. Äärmuslik liikuja kirjutab koodi pööramata tähelepanu programmeerimiskeskkonna tagasisidele ning „raiub aina edasi“.

Meier jaotas oma magistritöös [24] õppijad lahendamise sujuvuse alusel kolme gruppi:

1. müüriladujad, kes töötavad järk-järgult ning testivad kõigepealt valmiskirjutatud programmiosa (näiteks funktsiooni) enne järgmise osa kirjutamise juurde liikumist. Meier jaotab selle nad kaheks alamgrupiks: ühed, kes testisid koodi pidevalt funktsioonide kirjutamise ajal, ja teised, kes peale funktsiooni valmis kirjutamist seda testima hakkasid;
2. kiviraiujad, kes vastupidiselt müüriladujatele kirjutavad kõigepealt terve programmi koodi valmis ning alles siis hakkavad seda viimistlema. See koosneb koodi korduvatest käivitamisest ning järjest ilmuvate veateadete parandamisest;
3. meistrid, kes sarnaselt kiviraiujatele kirjutavad enne testima hakkamist programmi valmis, kuid nende oskused on piisavalt head, et testimise käigus vigu ei esine või siis on need meistri jaoks kergesti parandatavad.

Meier leidis, et kiviraiujatel esines võrreldes müüriladujatega rohkem veateateid, samad veateated kordusid tihedamini ja programmide kirjutamiseks kulunud aeg oli suurem. Lisaks vajasid müüriladujad vähem abi varasemalt lahendatud ülesannete näol. Kuigi

meistrid kirjutasid sarnaselt kiviraidujatele enne testima hakkamist terve koodi valmis, oli nende lahendamine sujuv nagu müüriladujatel. Meier pakub, et põhjuseks võib olla see, et ülesanne osutus nende jaoks liiga kergeks, kuid täheldab, et valim on liiga väike meistrite töökäigu analüüsimiseks.

Algajad programmeerijad puutuvad koodi kirjutamisel kokku sarnaste probleemidega, nagu sagedate süntaksivigade parandamisega, kuid nende käitumismustrites leiab siiski erinevusi. Seega tuleb õpetamisel tuleb arvestada sellega, et õpilased ei lähene probleemile samamoodi ning ülesannete lahendamisel vajavad osad lahendajad rohkem abi kui teised. Ülesannete koostamisel saab tekstidesse lisada vihjeid või näitelahendusi, mis aitaksid vajadusel probleemi paremini mõista.

2.4 Python programmeerimise õpetamisel

Programmeerimiskeele valik õpetamisel on samuti asi, millega tuleks arvestada. Kuigi suurem osa probleemidest, mis õppimisel esineb, on seotud pigem programmeerimise kui keele endaga, on siiski oluline mõelda, millises keeles õpetada [4]. Python on Guido van Rossumi loodud kõrgtaseme skriptimiskeel, mis lihtsustab programmeerimise õppimist ja on seega algajatele sobilik [27]. Linda Grandelli jt [28] kohaselt on Python sobilik programmeerimise õpetamiseks, sest sel leiduvad järgnevad omadused:

1. Võrreldes teiste populaarsete keeltega nagu Java või C++, on Pythonil **lühike ja puhas süntaks**. See muudab koodi kirjutamise loomulikumaks (vt tabel 1).

Tabel 1. Lihtsa programmi süntaksite võrdlus Pythonis ja Javas.

Python	Java

`print("Tere, maailm!")`	`class Tere {`
	`  public static void main (String args[]) {`
	`  System.out.println("Tere, maailm!");`
	`  }`
	`}`

2. Python on **dünaamiline** keel. See omadus vähendab koodi kirjutamist veelgi, kuna ei pea andmetüüpe täpsustama.

3. Interpretaator annab **kohese tagasiside** potentsiaalsete vigade puhul.
4. Pythonil on **sunnitud struktureeritud disain**. Taanded peavad olema korrektsed ning seetõttu kood meenutab pseudokoodi, mis on õppimisel abiks.
5. Python on **tasuta lähtekooditarkvara** ja see on laialdaselt kasutusel. Materjale, juhendeid ja dokumentatsiooni leidub veebis hulganisti.

Samuti tõid Grandell jt välja järgnevad nõrkused:

1. Python on skriptimiseks loodud keel ning seetõttu suurte programmide puhul võib **jõudlus kannatada**.
2. Pythoni **keele dünaamilisus** on ka nõrkuseks, kuna on lubatud samale muutujale määrata erinevat tüüpi andmeid, mistõttu on vead kerged tekkima.
3. Objektorienteeritult kirjutades **ei ole Pythonis võimalik atribuute peita**.

Kuna eesmärk on algajatele õpetada programmeerimist, mitte ühe programmeerimiskeele mitmeid võimalusi, ei ole õpetamisel need Pythoni nõrkused takistuseks. Probleemid jõudluse ja atribuutide peitmisega ei tohiks programmeerimise sissejuhatavas aines esineda, kuna sealsed loodavad programmid on lühikesed ning koodi ei kirjutata objektorienteeritult.

Mannila jt uurimuses [17] võrreldi keskkooliõpilaste poolt Pythonis ja Javas kirjutatud 60 programmi. Eesmärk oli leida, millised raskused programmeerimise õppimisel tulenevad puhtalt keelest. Tuli välja, et Javas kirjutatud programmides esines märkimisväärselt rohkem süntaksivigu, nagu puuduvad loogelised sulud ja semikoolonid ning defineerimata muutujate kasutamine. Samuti oli loogikavigade hulk Java programmides suurem. Tähelepanu tuleks pöörata sellele, et üle 50% Javas kirjutatud programmides olid algoritmid vigased või sootuks puudusid, samas Pythoni puhul juhtus see alla 10% juhtudest. Uuringu kohaselt oli Java keeruline süntaks takistuseks.

Esimese programmeerimiskeelena Pythoni õpetamisele on ka vastuargumente. Peamiselt väidetakse, et tegemist on liiga lihtsa keelega ja seetõttu tekib õpilastel hiljem keerukama süntaksiga keelte õppimisel mitmeid raskusi [17]. Siinkohal tuleks teha vahet sellel, et õpetatakse kooliõpilasi, mitte IKT-erialade üliõpilasi. Nagu ka teiste ainete puhul on üldhariduskoolide eesmärk valmistada õpilasi ette edasisteks õpinguteks. Just seda teeb programmeerimise õpetamine algoritmilise mõtteviisi ja ülesannete lahendamise oskusega.

3. Programmeerimisülesannete koostamine

3.1 Ülesande sõnastus ja huvi tekitamine

Programmeerimisülesannete puhul tuleb pöörata tähelepanu selgele sõnastusele. Winslow [29] järgi ei ole õpilastel raskusi süntaktiliselt korrektsete koodiridade kirjutamisega, kui on täpselt teada, mida neilt ülesandes küsitakse. Ent programmeerimiseülesanded on suuremas osas tekstülesanded, mis on ühed keerulisemat tüüpi ülesanded. Seega valmistab nende tekstist arusaamine tihtipeale raskusi. Ülesannete kontekst saab olla mitmesugune. Uurimustes [30, 31] on välja toodud, et matemaatiliste ülesannete kasutamine mõjub osadele õpilastele demotiveerivalt, sest tekib eeldus, et matemaatiliste teadmiste rakendamine ülesande lahendamiseks on keeruline, ning huvi programmeerimise vastu võib kaduda. Selleks, et matemaatilised ülesanded oleksid kergemini hoomatavad, jagatakse need tavaliselt mitmeks etapiks. Nii võib küll õpilane edukalt leida lahenduse erinevatele sammudele, kuid ta ei pruugi näha ülesannet kui tervikut [32].

Mikk jt [33] viisid läbi keskkooliõpilaste seas uurimuse teadustekstide keerukusest. Selgus, et mida keerukam on teaduslik tekst, seda raskem on sellest aru saada. Abstraktsed mõisted vähendavad lugemishuvi, kuna neid on raskem visualiseerida. Leiti, et sama mõju on ka teaduslikel valemitel, sümbolitel ja lühenditel. Huvipakkuvad teemad on need, mille kohta on inimesel kõige rohkem teadmisi. Kui uurimuses suutsid õpilased tekstiga suhestuda, oli ka huvi selle vastu märkimisväärselt kõrgem.

Programmeerimisülesannete tekstides saab kasutada elulist konteksti, et näidata õpilastele programmeerimise olulisust igapäevaelus [34]. Selline lähenemine paneb rohkem rõhku õpitu reaalsele rakendamisele ja pakub seega rohkem huvi ning annab motivatsiooni [32]. Hill [35] on kirjutanud, et teooria juurest praktika juurde liikumine on individuaalne kogemus, mistõttu peaks see protsess olema võimalikult lihtne. Õpilased peaksid nägema, et ühele probleemile on võimalik läheneda mitut moodi ning lahendusi võib olla erinevaid. Oma lahendusviisi valimine pakub võimalust olla loominguline ülesannete lahendamisel, mis jällegi aitab kaasa edukale probleemilahendusele.

3.2 Ülesannete keerukus

Teemasid, mida programmeerimise õpetamisel läbitakse, saab raskusastme järgi kategoriseerida. Butler jt [36] jagasid sissejuhataval programmeerimiskursusel läbitavad

teemad keerukuse põhjal madala-, keskmise- ja kõrgetasemelisteks. Programmeerimiskeele süntaks läheb madala raskustaseme alla, kuna süntaksi korrektsus on selgelt defineeritud ja kergesti kontrollitav ning sissejuhatavates ainetes kaetakse ära üsna väike osa terve keele võimalustest. Tingimus- ja korduslausete rakendamisel tekib rohkem probleeme, mistõttu on need keskmise raskusega. Kõige keerulisemaks osutuvad abstraktsed teemad, mille tegelik praktiline rakendamine nõuab algajatelt palju rohkemat. Nendeks on näiteks objektorienteeritud programmeerimisega seotud mõisted (klass, objekt jne), programmi disain ja järjendid. Butler jt peavad viimast erijuhuks, kuna järjendid ei ole küll nii raskesti ettekujutatav mõiste, kuid nende kasutamine programmis toob endaga kaasa tihtipeale tsüklid ja nõuab detailsemat töötlust.

Piteira jt [37] keskendusid sissejuhatavate programmeerimisainete täiustamisele, andes ülevaate programmeerimise õppimisel tekkivatest raskuskohtadest. Bakalaureuseastme üliõpilaste ja juhendajate seas viidi läbi küsitlus, mille käigus saadi teada, kui keeruliseks erinevaid teemasid peetakse. Lisaks analüüsiti ka õpilaste eksamitulemusi, et oleks võimalik võrrelda juhendajate ja üliõpilaste eeldusi teemade kohta. Sarnaselt Butler jt uurimusele [36] leidsid Piteira jt, et abstraktsed mõisted ja teemad valmistavad kõige rohkem raskusi, kuna neid ei ole kerge seostada päriseluga.

Uurimuses selgus, et juhendajad oskasid paremini ette näha, kui keeruliseks erinevad teemad osutuvad, üliõpilased olid aga hinnanguid andes liiga enesekindlad. Piteira jt järgi tuleneb see juhendajate tugevamatest teoreetilistest teadmistest ning varasemast õpetamiskogemusest. Mõlemad küsitletavad grupid pidasid problemaatilisteks teemadeks andmestruktuure ja erinditöötlust. Üliõpilaste jaoks tundus keeruline ka väliste teekide kasutamine, seevastu juhendajad pidasid seda üheks lihtsamaks teemaks. Nii üliõpilaste ja juhendajate arvamus kui ka eksamitulemused näitasid, et kursusel läbitavad teemad muutuvad järjest keerulisemaks. See võib Piteira jt põhjendusel tuleneda üldisest arvamusest, et programmeerimine ei ole lihtne.

Eksamitulemuste järgi oli kõige problemaatilisem teema andmestruktuurid, mis oli ka Butler jt järgi üks keerulisemaid. Piteira jt märgivad siiski ära, et andmestruktuuride ülesanded olid uuritud eksamil viimaste ülesannete seas ning võis juhtuda, et nendeni jõudes tekkis õpilastel piiratud aja tõttu ärevus.

Chang [38] on uurinud programmeerimise ja sellega seotud oskuste õppimisel tekkivat ärevust arvuti kasutamise ees ja selle seost ülesannete keerukusega. Chang defineerib

arvutiärevust kui rahutust, kartust ja stressi negatiivse tulemuse ees, mida tekitavad arvutiga seotud tegevused. Ta oletas, et uurimuses osalevad üliõpilased, kes üle- või alahindavad oskusi, mida on vaja arvutis ülesannete lahendamiseks, tajuvad ülesandeid vastavalt raskemate või kergematena, kui need tegelikult on. Uurimistulemused aga näitasid, et mida raskemaks lahendatavate ülesannete tase läks, seda täpsemini osati selle raskusastet tajuda. Seega nõrgemate oskustega üliõpilased, kes lahendasid madalama raskustasemega ülesandeid, üle- või alahindasid neid rohkem. See põhjustas omakorda suuremat ärevust arvutiülesannete lahendamise ees. Chang soovitab eriti algajatele suunatud arvutialastes ainetes pöörata rohkem tähelepanu sellele, kuidas vähendada sellist ekslikku raskustaseme tajumist. Selleks võib näiteks lihtsustada keelekasututust ülesannete püstituses või anda ettevalmistuseks piisavalt vajalikku taustainformatsiooni.

Programmeerimisülesannete raskusastme kergendamiseks toob Myers [39] välja tehnika „Programmeerimine näite järgi" (ingl *"Programming by Example"*). Selle korral antakse lahendajale ülesandeks vajalike (näite)andmed, mis sobib hästi algajatele, kuna kindlate näidetega töötamine on kergem, kui abstraktsete andmete ette kujutamine. „Programmeerimine koos näidetega" (ingl *"Programming with examples"*) nõuab ise programmis kasutatavate näiteandmete loomist, kuid sarnaselt eelmise tehnikaga, kasutatakse neid samu andmeid koodi kirjutamisel. Selline lähenemine on abiks, kuna tehtavate vigade arv väiksem, kui ülesannet saab lahendada näite põhjal.

Myers kirjutab lühidalt ka „siludes programmeerimisest" (ingl "*programming in debugging mode"*), mille käigus väljastatakse vaheetappidel ekraanile muutujate väärtused. See lihtsustab programmeerija tööd, kuna ta ei pea koodis olevate muutujate olekuid ise meeles pidama. Du Boulay jt [40] soovitavad samuti ekraanile kuvada programmi erinevatel sammudel toimunud muutused, sest see aitab paremini mõista iga koodirea mõju lõpptulemusele.

Graafika kasutamine ja visuaalne programmeerimine aitavad Myersi [39] järgi isegi kogenud programmeerijatel ülesannet hoomatavamaks muuta. Et inimesed suudavad palju paremini töödelda visuaalset informatsiooni, on ka ekraanil kahemõõtmelise graafika kujutamine abiks ülesannete lahendamisel.

3.3 Koolides kasutatavate materjalide programmeerimisülesanded

Selles peatükis antakse ülevaade olemasolevatest eestikeelsetest ülesannetest, mis on veebis vabalt kättesaadavad või mida on üldhariduskoolides programmeerimise ainetes ja kursustel kasutatud.

3.3.1 Valikkursuse „Rakenduste loomise ja programmeerimise alused" ülesanded

Koolides kasutatakse programmeerimise õpetamiseks materjale ProgeTiigri kogumikust [3, 9]. Programmeerimiskeele Pythoniga tutvumiseks pakutakse gümnaasiumile suunatud valikkursuse „Rakenduste loomise ja programmeerimise alused" [41] materjale, mis sisaldavad õpikut ja töölehti ülesannetega. Kursusel pannakse rõhku rakenduste loomisele, mitte programmeerimisele mingis kindlast keeles. Seetõttu tutvustatakse selle käigus rohkem kui ühte programmeerimiskeelt. Kuna bakalaureusetöö käigus luuakse ülesanded programmeerimiskeelt Python tutvustavale kursusele, keskendub töö autor siinkohal ainult Pythoni materjalidele mõeldud ülesannetele.

Ülesanded on toodud töölehtedel, kus on juures informatsiooni Pythoni funktsioonide kohta. Esimestes ülesannetes nõutakse etteantud programmi käivitamist ja selle vähesel määral muutmist (nt muutujate, avaldiste, Pythoni funktsioonide muutmine). Edasised ülesanded nõuavad komplitseeritumat lahendust, mistõttu ka püstitus on keerulisem. Tekstides on toodud nõuded rakendusele või funktsiooni(de)le, mitmetel on juures eluline kontekst, lisatud on hulganisti vihjeid ja näiteid. Viimased aitavad kindlasti lahendamisele kaasa, kuna ülesanded ise ei ole lihtsad. Mitmel juhul on juurde lisatud lahenduse pseudokood ja/või keeles Scratch loodud programm. Töö autor leiab, et see võtab õpilaselt võimaluse ise lahenduseks nõutav algoritm välja mõelda, seda eriti juhul, kui kursuse õpikus on mainitud keskendumist algoritmilise mõtlemise arendamisele. Pigem võiksid ülesanded olla lihtsamad ja seeläbi vältida pseudokoodi näitamist. Kuid kuna tegemist on kõigest töölehel väljatoodud ülesannetega, siis programmeerimise tundides neid kasutades ennetab see pikemate ülesannete lahendamise juurde seisma jäämist.

Eluline kontekst on lisatud juurde mitmetele ülesannetele, mis on õpilastele näide sellest, kuidas kirjutatud programme saab reaalses elus rakendada. Antud valikkursuse ülesannetes on nõuded programmile pikad ning tekstides on need tihti esitatud matemaatilisel kujul. Nagu on kirjutanud Mikk jt [33], raskendavad teaduslikud valemid ja sümbolid teksti mõistmist. Lisaks nagu tunduvad matemaatilised ülesanded mõningatele

õpilastele keeruliste ja rasketena [30, 31]. Matemaatiline on ülesande „Funktsioonid" püstitus (vt joonis 1).

Koostada alltoodud valemite järgi kaks funktsiooni:

- rasvaprotsendi leidmine etteantud soo (mees/naine), vanuse t (aastates), pikkuse l (cm) ja kaalu m (kg) alusel

- hinnangu andmine pikkuse l (cm) ja kaalu m (kg) alusel

$$m_{id} = \begin{cases} (3l - 450 + t) \cdot 0{,}225 + 40{,}5 \ \ naine \\ (3l - 450 + t) \cdot 0{,}250 + 45{,}0 \ \ mees \end{cases}$$

$$r = \begin{cases} \dfrac{m - m_{id}}{m} \cdot 100 + 22 \ \ naine \\ \dfrac{m - m_{id}}{m} \cdot 100 + 15 \ \ mees \end{cases}$$

$$k_{ind} = \frac{m}{(l/100)^2}$$

$$hinnang = \begin{cases} k\tilde{o}hn \ \ k_{ind} \leq 18 \\ normaal \ \ 18 \leq k_{ind} \leq 25 \\ \ddot{u}lekaal \ \ 25 < k_{ind} \leq 30 \\ ei \ v\tilde{o}i \ olla! \ \ k_{ind} > 30 \end{cases}$$

Joonis 1. Ülesanne „Funktsioonid" [41].

3.3.2 MOOCi „Programmeerimise alused" ülesanded

„Programmeerimise alused" [42] on Tartu Ülikooli arvutiteaduse instituudi poolt pakutav algajatele suunatud MOOC (ingl *massive open online course*). Kursusel on iga nädala teemade juures kontrollülesanded. Nende ülesehitus on läbivalt sarnane, st ülesanne sisaldab mingit konteksti, nõudeid programmile, lisatud on näidisväljund ja mõnel juhul ka lahendusvihje(d). Pikemate ülesannete lihtsamaks mõistmiseks on nõuded toodud ka iga funktsiooni kohta. Näited programmi või funktsiooni tööst mitte ainult ei illustreeri nõudeid, vaid on ka abiks testimisel. Ülesannetes, kus nõutakse kasutajapoolset sisendit, saab näidisväljundis kasutatavaid andmeid kasutada testandmetena. Ülesannete osad on selgelt eristatavad, mis teeb teksti jälgimise lihtsaks. Näide nende osade olemasolust on kontrollülesanne „3.2 Lillede arv" (vt joonis 2). Konteksti osas antakse ülesandele eluline sisu. See on ära toodud esimeses lõigus. Püstituse teises osas on nimekirjana loetletud kõik nõuded. Juurde on lisatud vihje koos näitega, mis aitavad eelneva teksti arusaamisel juhul, kui see jäi mõistmatuks.

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* $1 + 3 + 5 + 7 = 16$.

Joonis 2. Ülesanne „3.2 Lillede arv" [42].

Kivisoo koostas magistritöö [11] käigus materjalid Pythoni algkursuse läbiviimiseks, mida katsetati 9. ja 10. klassi õpilaste peal. Aluseks võeti Tartu Ülikooli arvutiteaduse instituudi MOOCi „Programmeerimise alused" õppematerjalid [42]. Kivisoo koostatud materjalid sisaldasid kontrollülesandeid iga nädala teemade kohta. Enamikel nädalatel pidi lahendamiseks valima kahe kuni kolme ülesande seast ühe. Kivisoo töös on tabelina iga ülesande kohta ära toodud, mitu õpilast neid kokku lahendas ning valikuliste ülesannete juures on näha, millised osutusid õpilaste seas ebapopulaarseimateks. Üks suurimaid erinevusi tuleb välja tingimuslausetele keskenduval nädalal, kus kolme võimaliku ülesande seast valis lahendamiseks teise ainult kolm õpilast, teisi aga valiti võrdselt. Tsüklite teema juures osutus samuti teine ülesanne kõige ebapopulaarsemaks. Selle valis viis õpilast, esimest ja teist aga vastavalt 11 ja 7 õpilast. Põhjuseks võib pidada seda, et mõlemal juhul tuli ebapopulaarsetes ülesannetes leida tõenäosust, mis võis tunduda kui keerulise matemaatilise alamülesandena.

Nädalal, mil õpiti järjendeid, osutus esimene valikuline ülesanne ülejäänud kahest märkimisväärselt kaalukamalt valituks. Seda valiti 13 korda ja teisi 2 ning 5 korda. Selline menukus võis tuleneda kontekstist. Nimelt tuleb esimeses ülesandes luua programm, mis jäljendab plaadimasinat (ingl *jukeboxi*) ja kasutab andmetena faile tuntud Eesti ja välismaa muusikapaladega. Teine ülesanne keskendub loomuliku iibe leidmisele ning nõuab samuti

andmete failidest lugemist ja kolmanda sisuks on õpilaste tahvli juurde kutsumine, kus tuleb lahenduse käigus leida ka tänane kuupäev. Kuna kõigis ülesannetes leidub keerulisemaid aspekte, nagu failist lugemine või Pythoni moodulite kasutamine, sõltus ülesannete populaarses tõenäoliselt suuresti kontekstist. Seega plaadimängija ja muusikapaladega oli õpilastel ilmselt kergem suhestuda ning selline sisu pakkus neile rohkem huvi.

3.3.3 Kursuse „Teeme ise arvutimänge – algus" ülesanded

Tartu Ülikooli veebikursus „Teeme ise arvutimänge – algus" [43] on gümnasistidele suunatud sissejuhatav programmeerimiskursus, mille ülesanded keskenduvad algeliste mängude loomisele. Muhhin [44] lõi bakalaureusetöö raames antud kursusele programmeerimisülesanded. Programmeerimiskeelena kasutati Pythonit. Kuna kursus oli orienteeritud arvutimängude loomisele, on osa Muhhini loodud ülesandeid lihtsamad mängud. Näiteks tsükleid ja tingimuslauseid tuli rakendada luues mäng, mille eesmärgiks oli arvata ära genereeritud juhuslik number, ning kahekordsete tsüklite osas pidid õpilased looma programmi laevade pommitamise mängust.

Muhhin toob välja iga ülesande juures, milliseid läbitud teemasid see katab. Selle asemel, et koostada ülesandeid, mis ainult kontrollivad läbitud teemade rakendamist, on ülesannetele antud juurde kontekst, mis aitab muuta ülesanded elulisemaks ja huvitavamaks [34].

Õpilaste seas viidi läbi küsitlus, mille käigus sai tagasisidet loodud ülesannete kohta. Esimeste nädalate teemade ülesanded, mis olid ka lihtsamad, pakkusid õpilastele vähem huvi kui hilisematel nädalatel, kuna kursusel osales õpilasi, kes oli varasemalt programmeerimisega kokku puutunud ning nende taseme jaoks olid need ülesanded lihtsavõitu. Täheldati, et ülesande püstituse mõistmisega tekkis probleeme, kui programmi kirjutamiseks vajalikud matemaatilised teadmised jäid puudulikuks (näiteks Pythagorase teoreemi rakendamine).

3.4 Kokkuvõte programmeerimise õpetamisest ja ülesannete koostamisest

Eesti üldharidustasemel on erinevate koolide vahel programmeerimise õpetamine, sh ainete sisu ja kasutatavad õppematerjalid, varieeruv. Kuigi programmeerimise õppimine sellel tasemel ei ole lihtne, on ometi võimalik programmeerimisoskus omandada ka kooliõpilastel. Selle protsessi üheks oluliseks osaks on ülesannete lahendamine. Tuleb

arvestada sellega, et õpilaste käitumine ülesannete lahendamisel (teksti mõistmine, probleemile lähenemine, programmi loomise etapid) ei ole samasugune.

Ülesannete koostamisel tuleb tähelepanu pöörata sellele, kui raskesti mõistetav on tekst. See oleneb näiteks teaduslike sümbolite, lühendite ja matemaatiliste valemite kasutusest. Mida mõistetavam on ülesande püstitus, seda huvitavam see tundub. Huvi saab tekitada ülesandele elulist konteksti andes, kuna see annab võimaluse tegeleda õpilastele juba tuttavate probleemidega ning näitab, kuidas on võimalik programme kasutada igapäevaste ülesannete lahendamiseks.

Hetkel Eesti koolidele pakutavates materjalide seas leidub ka ülesandeid programmeerimise algõppe tarbeks. Üldiselt on ülesannetel sarnane ülesehitus. Ära on toodud nõuded programmile, tavaliselt on lisatud juurde mingisugune eluline kontekst. Juures võivad olla ka lahendusvihjed ja lõpptulemust illustreerivad näited. Sarnasele ülesehitusele vaatamata varieeruvad erinevatest allikatest pärit ülesanded siiski mingil määral vormistuse ja raskustaseme poolest.

4. Metoodika

4.1 Eeltöö

Bakalaureusetöö raames koostati ülesanded viiele digiõpiku „Tarkvaraarendus“ [8] peatükile. Töö eesmärgi saavutamiseks töötati läbi ja pandi iga peatüki puhul kirja, milliseid programmeerimise elemente need sisaldavad. Kuna „Tarkvaraarendus“ on järg gümnaasiumi õpikule „Programmeerimine“ ja kasutajalt eeldatakse sealsete teemade varasemat omandamist, töötati läbi ka selle õpiku teemad, et kaardistada ka varasemalt nõutavad oskused.

Enne ülesannete koostamist tutvuti ka juba samateemaliste olemasolevate ülesannetega teistes õppematerjalides. Kasutati MOOCi „Programmeerimise alused II“ materjale [45], Aivar Annamaa digiõpikut „Programmeerimine“ [46] ja Tartu Ülikooli õppejõudude koostatud programmeerimise eksamiülesannete kogu [47]. Need võeti aluseks raskustaseme määramisel, millele ülesanded peaksid vastama, ja aitasid mõista, millistest osadest ülesanded koosnema peaksid ning kuidas on lahendajas huvi tekitamiseks ülesande püstituses kasutatud elulist konteksti.

4.2 Ülesannete koostamise protsess

Ülesannete koostamisel peeti eelkõige silmas, et programmi loomise käigus korratakse ja kinnistatakse õpikus „Programmeerimine“ ja õpikus „Tarkvaraarendus“ omandatud teadmisi ja oskusi. Pärast peatükkidega tutvumist, pandi vastavate teemade juures märkmetena kirja erinevad ideed elulistest probleemidest, millele teemaga seonduvad ülesanded keskenduda võiksid. Näiteks kahemõõtmelise järjendiga saaks esitada kortermaja, kus järjendi esimene tase kujutab maja korruseid ja teine korterite numbreid.

Toetudes varasemate ülesannete ülesehitusele, püüti vältida dubleerimist varasemate ülesannete püstitustega, nagu täringuviskel juhusliku arvu genereerimine ja summa funktsiooni kirjutamine järjendi elementide kokkuliitmiseks. Kuna taolisi tüüpilisi ülesandeid esineb tihti programmeerimismaterjalides ja need on ka kergesti leitavad veebist ning õpetajad on ilmselt selliste ülesannetega juba kokku puutunud, oleksid originaalse sisuga ülesanded väärtuslikumad.

Ülesannete loomisel läbiti järgmised sammud:

1. tehti selgeks, milliseid programmeerimise elemente on vaja käesolevas peatükis katta,
2. tutvuti olemasolevate ülesannetega, mis keskendusid nende teadmiste kontrollimisele,
3. mõeldi ülesandele kontekst,
4. kirjutati valmis programm, kus rakendatakse peatükis kaetud elemente,
5. kirjutati valmis ülesande püstitus,
6. lisati näiteväljund ja vajadusel vihje(d).

Iga koostatud ülesande puhul oli üheks sammuks lahenduskoodi valmis kirjutamine. Nii sai autor ise näha, kas raskusaste on paras, ja sai teha kindlaks, kas nõutud lahendus on sellisel kujul võimalik ja optimaalne. Nagu juba varasemalt mainitud on elulise sisuga ülesanded motiveerivad, mistõttu peeti konteksti lisamist oluliseks osaks.

4.3 Ülesannete katsetamine

Töö raames loodud programmeerimisülesandeid piloteeriti valikkursusel „Tarkvaraarendus“ Tartu Tamme Gümnaasiumis. Seda selleks, et saaks otsest tagasisidet gümnaasiumiõpilastelt ja –õpetajalt. Õpetajalt küsiti kasutatud ülesannete kohta tagasisidet ja selle põhjal viidi sisse vajalikud täiendused. Loodud 25 ülesande hulgast katsetati järgmisi:

1. „2.1 Lauatennis“ („Kahemõõtmeline järjend“),
2. „2.4 Hinnete tabel“ („Kahemõõtmeline järjend“),
3. „3.4 Kalorsus“ („Kahekordne tsükkel“),
4. „3.6 Minesweeper“ („Kahekordne tsükkel“),
5. „5.2 Hamlet“ („Andmestruktuurid“),
6. „6.1 Kiirloto“ („Rekursioon“) ja
7. „6.2 Püramiid“ („Rekursioon“).

Ülesandeid saab näha töö I lisas ja nende piloteerimisel esinenud probleemid ning vastavalt tagasisidele tehtud parandused on leitavad bakalaureusetöö peatükist „Tagasiside ja parandused ülesannetele“.

5. Tulemused ja arutelu

5.1 Teemade jaotus

Bakalaureusetöö raames valmis 25 programmeerimisülesannet, mis on leitavad töö I lisas. Programmeerimise ülesanded koostati õpiku viiele peatükile („Kahemõõtmeline järjend“, „Kahekordne tsükkel“, „Andmevahetus“, „Andmestruktuurid“, „Rekursioon“) ja nende alamteemadele, millest annab ülevaate tabel 2. Iga peatüki kohta loodi minimaalselt viis ülesannet. Erandiks on „Andmevahetus“, kus kaetakse failis olevate andmete lugemist kahemõõtmelisse järjendisse. Küll aga failist lugemist korratakse ka järgnevate teemade, nagu sõnastikud ja rekursioon, ülesannetes. Ülesannetes „4.1 Tehnika“ ja „5.5 Albumid“ kasutatavate failide koostamiseks kasutati Statistikaameti andmebaasist [48] ja Wikipedia artiklist [49] leitud andmeid.

Õpikus on mitmetel peatükkidel lisalugemise alampeatükk, kuid nagu tabelist 2 on näha, koostati algselt ülesanded ainult peatüki „Rekursioon“ lisalugemisele. Põhjuseks oli see, et rekursioonist ei olnud varasemalt ka „Programmeerimise“ valikkursusel räägitud ning lisalugemise osas kaeti palju olulisi teadmisi. Üldiselt on ülesanded loodud nii, et need kinnistaksid järk-järgult uut esitatavat informatsiooni. Peatüki lõpus on üks või kaks suuremat ülesannet, mille lahendamiseks tuleb meelde tuletada terves peatükis õpitut. Tabelis 2 esineb ülesandeid, kuhu on juurde märgitud, et neid on muudetud. Tegemist on nende ülesannetega, mille puhul vastavalt piloteerimisel saadud soovitustele kohendati mitte ainult teksti, vaid ka nõudeid programmile. Alles jäeti ka originaalsed versioonid juhuks, kui õpetajad soovivad õppetöös algse tekstiga ülesandeid kasutada.

Tabel 2. Kursusele „Tarkvaraarendus“ loodud ülesannete nimekiri vastavalt peatükkidele.

Peatükk	**Alamteema**	**Ülesanne (rasvases kirjas on märgitud piloteeritud ülesanded)**
2. Kahemõõtmeline järjend	Sissejuhatus	**2.1 Lauatennis**
		2.2 Vale sõna välja
		2.3 Retseptid

	Järjendite järjend	**2.4 Hinnete tabel**
	Kokkuvõttev ülesanne	2.5 Kohustuslik kirjandus
3. Kahekordne tsükkel	Järjend ja tsükkel	3.1 Muster
		3.2 Kaustade sorteerimine
		3.3 Koristaja
	Järjend ja funktsioon	**3.4 Kalorsus**
		3.5 Nummerda
	Kokkuvõttev ülesanne	**3.6 Minesweeper**
4. Andmevahetus	Failist järjendisse lugemine	4.1 Tehnika
	Lisalugemine: CSV	4.2 Hinnete tabel (muudatustega)
5. Andmestruktuurid	Ennik	5.1 Kombinatsioonid
	Sõnastik	**5.2 Hamlet**
		5.3 Kangid
	Kokkuvõtvad ülesanded	5.4 Arva salasõna
		5.5 Albumid
6. Rekursioon	Rekursiooni sissejuhatus	**6.1 Kiirloto**
		6.2 Kiirloto (muudatustega)
		6.2 Püramiid
		6.3 Romb
		6.4 Mäng

	Lisalugemine: rekursioonist põhjalikumalt	6.5 Liida
		6.6 Eralda
	Kokkuvõtvad ülesanded	6.7 Kohver
		6.8 Shuffle

5.2 Ülesannete ülesehitus

Loodud ülesanded sarnanevad üksteisele ülesehituse poolest. Saab eristada nelja elementi: kontekst, nõuded programmile, näiteväljund ja vihjed. Selleks, et vältida olukorda, kus õpilase jaoks tundub mingi ülesande teema kauge või keeruline, on teemad ülesannete vahel varieeruvad. Joonisel 3 on toodud ülesande „2.1 Lauatennis“ tekst. Ülesandes on kontekstiks lauatennise võistlus, millest on antud taustainformatsiooni ülesande püstituse esimeses ja teises lõigus.

Lauatennise võistlusel mängitakse viiest parem süsteemis. See tähendab, et mängu võitmiseks on vaja võita 3 setti. Seti võitmiseks on vaja saada 11 punkti nii, et mängijate punktide vahe oleks vähemalt 2 punkti. Seega skoor 11–8 on võit, kuid skoori 11–10 puhul tuleb mängida edasi, kuni punktivahe on vähemalt 2.

Võistlusel mängivad Eesti ja Soome mängija. Iga seti punktid on salvestatud kahemõõtmelisse järjendisse, kus esimesel kohal on Eesti mängija punktid ning teisel Soome mängija omad.

Kirjutada programm, mis seti tulemuste põhjal väljastab võitja ning lõpliku mänguseisu.

Näide programmi tööst:

Tulemuste järjend:

```
tulemused = [[11, 8], [11, 5], [9, 11], [14, 12]]
```

Programmi töö:

```
>>> %Run lauatennis.py
Eesti võitis 3-1
```

Joonis 3. Ülesanne „2.1 Lauatennis“.

Ülesanne kontrollib kahemõõtmelisest järjendist indeksi abil elementide kättesaamist. Püstituse kolmandas lõigus on välja toodud nõuded programmile. Juurde on lisatud ka

näide programmi tööst. See on abiks ülesande tekstist arusaamisel ning pakub näiteandmed ja nendele vastava väljundi, millega saab programmi tööd testida.

Leidub ka väike hulk ülesandeid, millel eluline kontekst puudub. Nende püstitus on lühem ja konkreetne. Õpikus on rekursiooni peatüki lisalugemise osas kirjutatud sellest põhjalikumalt. Selleks, et harjutada funktsiooni korduvat väljakutset, ei sisalda esimesed kaks sellele alamteemale loodud ülesannet konteksti, vaid keskenduvad puhtalt rekursiooni rakendamisele. Küll aga on peatüki järgnevates ülesannetes lisatud juurde ka kontekst, et probleem elulisemaks muuta. Tegemist on teemaga, mis valmistab algajatele tihtipeale raskusi. Pikem ülesande tekst, kus on lisaks programme nõuetele kirjas ka taust, võib muuta tekstist arusaamist keerulisemaks. Selline on näiteks ülesanne „6.4 Liida", mis sisaldab nõutud funktsiooni kirjeldust ja näiteväljundit (vt joonis 4).

Kirjutada rekursiivne funktsioon, mis liidab pikemale järjendi elementidele lühema järjendi vastavad elemendid. Näiteks järjendite `[1, 2, 3, 4]` ja `[5, 6]` korral on tagastatav tulemus `[6, 8, 3, 4]`.

Joonis 4. Ülesanne „6.4 Liida".

Nagu näha, sisaldab tekst siiski näidet juhuks, kui probleemi püstitus segaseks jääb. Selle ülesande lahenduses tuleb varasematest teadmistest kasutada tingimuslauseid ja järjendist indeksi abil elementide kättesaamist. Funktsiooni kirjutamisel tuleb lahendamisel silmas pidada seda, kuidas kasutada rekursiooni kui tsüklit, kus kohas koodis asub rekursiivne väljakutse, millised on rekursiooni baas ja samm ning mitu funktsiooni argumenti kasutusele võtta. Sellistest rekursiivse programmi osadest on kirjutatud ja toodud näiteid õpiku vastavas peatükis. Joonisel 5 on näha ühte näitelahendust.

```
def liida(j1, j2, i):                    # valitud on kolm funktsiooni argumenti
   if i == len(j1) or i == len(j2):      # rekursioon kui tsükkel
      if len(j1) > len(j2):              # rekursiooni baas
         return j1
      else:
         return j2
   if len(j1) > len(j2):
      j1[i] = j1[i] + j2[i]
   else:
      j2[i] = j1[i] + j2[i]
   return liida(j1, j2, i+1)             # rekursiooni samm
```

Joonis 5. Ülesande „6.4 Liida" näitelahenduse kood koos kommentaaridega.

Ülesande püstituses on ära toodud tingimised programmile. Kui lahenduseks ei ole nõutud funktsiooni, on üldiselt tegemist programmiga, mis väljastab midagi ekraanile. Sellisel juhul on kirjas, mida ja kuidas peaks kuvama, ning on lisatud väljundi näide. Üks selline ülesanne on „3.1 Muster", kus tuleb etteantud mustriosast, mis on järjendi kujul, peegeldamise ja ümberpööramise abil luua tervik (vt joonis 6). Nõuded programmile on ära toodud teksti esimeses lõigus. Tekstist parema visuaalse ettekujutuse saamiseks on juures ka näide ekraanile kuvamisest koos joonisega.

Kirjutada programm, mis teeb mustri ühest veerandist terve sümmeetrilise mustri. Programm teeb kahemõõtmelisest järjendist suurusega nm kahemõõtmelise järjendi suurusega $(n + (n - 1))(m + (m - 1))$. Antud esimest veerandit tuleb peegeldada mööda sümmeetriatelgi (viimane rida ja viimane veerg).

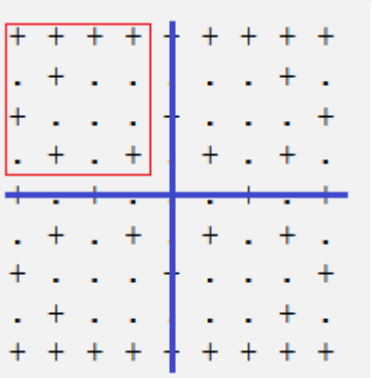
Näide programmi tööst:

```
# Esimene veerand:

+ + + + +
. + . . .
+ . . . +
. + . + .
+ . + . .

# Terve muster:


+ + + + + + + + +
. + . . . . . + .
+ . . . + . . . +
. + . + . + . + .
+ . + . . . + . +
. + . + . + . + .
+ . . . + . . . +
. + . . . . . + .
+ + + + + + + + +
```



Joonis 6. Ülesanne „3.1 Muster".

Kui lahenduseks nõutakse ühte või mitut funktsiooni, on ülesande püstituses kirjas, millised on nõutud argumendid ja mida tagastatakse. Erandiks on siinkohal rekursioonile keskenduvad ülesanded, kus üheks oluliseks osaks on kontrollida õpilaste oskust valida sobivaid funktsiooni argumente, mistõttu ei ole neis alati eraldi välja toodud informatsiooni nõutud argumentide kohta. Selline on näiteks juba eelnevalt väljatoodud ülesanne „6.4 Liida".

Mõningatele ülesannetele on lisaks näidetele juures ka vihjed. Peamiselt annavad need informatsiooni erinevatest moodulitest imporditavate funktsioonide kohta, kuid keerulisemate ülesannete puhul on lisatud abistavad vihjed. Need aitavad ülesannet väiksemateks osadeks jagada ja on suunavad probleemi lahendamisel. Ülesanne „2.5 Kohustuslik kirjandus“ võtab kokku peatükis „Kahemõõtmeline järjend“ läbitud teemad.

Lahendusena tuleb luua programm, mis väljastaks ekraanile raamatute pealkirjad lugemiskiiruse põhjal. Sellele ülesandele on pakutud kolm vihjet, millest esimene viitab sellele, et programmi võiks jagada sammudeks, teine suunab esimese sammu lahendust, tuletades meelde füüsikast ja matemaatikast tuttavat valemit, ning viimane tutvustab imporditava mooduli funktsiooni (vt joonis 7).

1. Lugemiskiirus on vaja teisendada kujule lk/h.
2. Kiiruse valem $v = \frac{s}{t}$, kus v on kiirus, s on teepikkus ning t on aeg.
3. Üles ümardamiseks on funktsioon `ceil`, mis tuleb importida moodulist `math`. Näide:

```
>>> from math import ceil
>>> ceil(2.1)
3
>>> ceil(3.8)
4
```

Joonis 7. Ülesande „2.5 Kohustuslik kirjandus“ vihjed.

5.3 Tagasiside ja parandused ülesannetele

Siin peatükis tuuakse välja tagasiside katsetatud ülesannete kohta ja vastavalt nendele märkustele sisse viidud parandused. Ülesannete „1.1 Lauatennis“ ja „6.3 Püramiid“ kohta soovitusi muutmiseks ei tehtud. Kahel juhul viidi tekstides sisse täiendused mitte ülesande püstituse osas, vaid õpilast abistavas osas. Esimene oli ülesanne „3.4 Kalorsus“, kus muudatus tehti programmi tööd illustreerivas näites, kuhu lisati funktsioonile argumendina antavasse järjendisse kolme toidukorra asemel neli, et ei tekiks võimalust omavahel segamini ajada kolme makrotoitainet kolme söögikorraga. Teine ainult abistava osa muudatus, tehti ülesandes „5.2 Hamlet“, kuhu lisati vihje failist lugemisel reavahetuste eemaldamise ja lahenduskäigu sammude järjekorra kohta. Juhul, kui ülesandel puudus näide programmi või funktsiooni tööst, lisati need juurde, et õpetajatel oleks võimalus valida, kas näidata näiteväljundit või mitte.

Nõudeid vähendati kahes ülesandes: „3.6 *Minesweeper*“ ja „6.1 Kiirloto“. Esimeses ülesande oli eesmärgiks mängu Minesweeper mänguvälja genereerimine. Algselt pidi

lahendusprogramm koosnema kahest funktsioonist, millest esimene lõi mänguväljale vastava järjendi ja teine kuvas selle ekraanile, kuid lihtsuse ja tunnis lahendades ajakulu kokkuhoiu mõttes soovitati need kombineerida üheks funktsiooniks. Eemaldati ka miinide arvu kontroll, millega vähenes mänguvälja loomisfunktsioonilt nõutav argumentide arv kolme pealt kahele. Teksti lisati ka juurde täpsustus kasutada lahenduses kahekordset tsüklit.

Ülesandes „6.1 Kiirloto“ vähendati samuti nõudeid programmile, kuid muudeti ka tekst arusaadavamaks. Erinevad võidusummad- ja võimalused asendati ühega, mis eemaldas vajaduse võidutõenäosuste programmeerimiseks (vt joonis 8). Lisaks kadus eelnevalt olemasolev võimalus erinevate võitude põhjal otsustada, kas osta uus pilet või mitte, kuna võita oli võimalik ainult ühte summat. Seetõttu kadus ka vajadus kahe erineva rekursiivse väljakutse järele.

Varasem ülesande tekst:

Kiirlotot mängides on võimalik võita 2, 10 või 50 eurot ning nende summade võiduvõimalused on vastavalt 20%, 15% ja 5%.

Mängija ostab pileti, mis maksab 2 eurot. Ta ostab uusi pileteid senikaua, kuni ta võidab rohkem kui 2 eurot või raha saab otsa. Kui mängija võidusummaks on 2 eurot, ostab ta selle raha eest uue pileti.

Kirjutada rekursiivne funktsioon `kiirloto`, mis leiab pileti võidusumma, mängib mängija reeglite järgi kiirlotot ning lõpuks tagastab enniku, mille esimeseks elemendiks on võidetud summa ja teiseks mängimise käigus ostetud piletite arv. Funktsioonil on kaks argumenti: mängija raha jääk ning ostetud piletite arvu loendur. Pileti võidusumma genereerimine on soovitatav teha abifunktsioonina, mida rekursiivses funktsioonis välja kutsutakse.

Muudetud ülesande tekst:

Kiirlotot mängides on võimalik võita 4 eurot, mille võiduvõimalus on 30%.Mängija ostab pileti, mis maksab 2 eurot. Ta ostab uusi pileteid senikaua, kuni uue ostuks enam raha ei jätku.

Kirjutada programm, mis mängib mängija reeglite järgi kiirlotot ja väljastab ekraanile,mitu eurot mängimisest tagasi saadi. Programmi sisaldab rekursiivset funktsiooni kiirloto, millel on kaks argumenti: mängija raha jääk ja võidetud rahasumma. Töö lõpuks tagastab funktsioon enniku kujul (raha_jääk, võidusumma).

Joonis 8. Ülesande „6.1 Kiirloto“ algne ja muudetud tekst.

Need kaks suuremat õpetajapoolset modifitseerimist ei tähenda, et esialgne ülesanne oleks olnud lahendamatu, vaid pigem seda, et õpetajal on vabadus ülesandeid täiendada vastavalt konkreetsele olukorrale kursusel.

Võib ka juhtuda, et üks ülesanne sobib mitme teema juurde. Näiteks ülesanne „2.4 Hinnete tabel“ (vt joonis 9), mis algselt oli mõeldud peatüki „Kahemõõtmeline järjend“ tarbeks, muudeti ümber nii, et seda saaks kasutada hilisemas peatükis „Andmevahetus“ alamteemas „Lisalugemine: CSV“.

Varasem ülesande tekst:

Hinnete tabel on kahemõõtmelise järjendi kujul. Iga aine on eraldi järjendis, mis sisaldab esimesel kohal aine nimetust ning ülejäänud kohtadel hindeid. Hindeid võib olla 1 või rohkem.

Kirjutada programm, mis väljastab ekraanile iga aine keskmise hinde eraldi real.

Hinnete tabel järjendina:

```
hinnete_tabel = [["eesti keel", 5, 4, 3, 4, 4, 3, 4, 4],
                 ["matemaatika", 4, 4, 5, 5, 4, 5, 5, 4, 5],
                 ["keemia", 4, 3, 3, 2, 3, 4],
                 ["kirjandus", 5, 5, 5, 4, 5, 5, 4],
                 ["bioloogia", 4, 5, 4, 4, 3, 5, 5, 5]]
```

Muudetud ülesande tekst:

Hinnete tabel on CSV-failis hinded.csv, kus on eraldi ridadel erinevad õppeained. Iga aine kohta on toodud selle nimetus ja vastavad hinded. Kirjutada programm, mis väljastab ekraanile iga aine keskmise hinde eraldi real.

Joonis 9. Ülesande „2.4 Hinnete tabel“ tekst ilma näiteta.

Samuti on õpetajatel võimalik iseseisvalt valida, kuidas teemat käsitleda. Näiteks antud teema kohta koostas õpetaja ülesande „2.4 Hinnete tabel“ põhjal töölehe, mis on toodud töö II lisas. Algselt olid programmis kasutatavad andmed kahemõõtmelise järjendi kujul, kuid peale muudatusi loetakse need sisse CSV-failist, mille õpilased koostavad ise. Lisaks töödeltakse sisseloetud andmeid erinevatel viisidel, näiteks nõutakse ühe hinde esinemissageduse ja positiivsete hinnete arvu leidmist. Orginaalülesandes nõuti ainult iga hinde kohta aritmeetilise keskmise arvutamist.

5.4 Võimalikud edasiarendused

Osa ülesannetest on olnud kasutuses ja nende kohta on saadud tagasisidet ka ühe valikkursuse „Tarkvaraarendus“ läbiviinud õpetaja käest. Töö autori arvates oleks hea, kui siiski kõiki loodud ülesandeid saaks mingil hetkel katsetada ka reaalses koolisituatsioonis. Kursust läbivate gümnasistide seas võiks läbi viia ka küsitluse, kus nende käest küsitakse tagasisidet ülesannete raskusastme, huvitavuse, tekstide arusaadavuse ning lahenduseks kulunud ajakulu kohta. See aitaks teha vastavaid parandusi ning tõsta ülesannete kvaliteeti.

Ülesannetele sobiks juurde luua automaatkontroll sisend-väljund- ja funktsioonipõhiste testidega, kuna õigeaegse tagasiside saamine on õppimisel ülimalt oluline [21]. Charmani ja Elmesi [50] järgi aitab see siduda töö tegemist ning tulemuse nägemist, vastupidiselt päevi või nädalaid hiljem tagasiside nägemisele, mis ajaks keskendutakse hinde nägemisele, mitte vigadele, mida tulevikus vältida. Lisaks on see ka motiveeriv, kuna õppijad soovivad kiiresti oma tulemusi teada. See säästaks ka õpetajate aega, kuna nad saaksid juba enne esitatud koodi nägemist aimu, milliseid probleeme lahenduses leidub. Automaattestide kirjutamine ei ole lihtne ning nõuab korraga palju ajaressurssi, kuid pikemas perspektiivis on see siiski ajasäästlik ning seda eriti juhtudel, mil on tegemist suure õppijate hulgaga [21].

6. Kokkuvõte

Bakalaureusetöö eesmärk on luua programmeerimisülesannete kogu gümnaasiumi valikkursusele „Tarkvaraarendus“. Töö käigus valmis 25 ülesannet. Need koostati viiele valikkursusel kasutatava digiõpiku peatükile. Eelkõige jälgiti, et nõutav lahendus kontrolliks neis peatükkides õpitud teadmisi ja kordaks varasemaid. Seitset ülesannet katsetati gümnaasiumiõpilastega valikkursuse piloteerimisel ja vastavalt õpetajalt saadud soovitustele viidi sisse mõngingad muudatused.

Enne ülesannete loomist töötati läbi valikkursuse „Tarkvaraarendus“ ja sellele eelneva kursuse „Programmeerimine“ õpikud ning tutvuti olemasolevates programmeerimise materjalides pakutud ülesannetega. Seejärel mõeldi ülesandele juurde eluline kontekst, kirjutati valmis lahenduskood ning lõpuks pandi selle põhjal kirja ülesande püstitus ning lisati õpilasi abistavaid elemente, nagu vihjed ja näitelahendused. Tekstid püüti kirja panna võimalikult huvitavalt ja üheselt mõistetavalt. Huvi tekitamist peeti oluliseks, kuna see motiveerib lahendama ja programmeerimise õppimisega tegelema. Eluline kontekst lisati selleks, et muuta ülesannete püstitus põnevamaks ja õpilastele lähedasemaks. Tekstist üheselt arusaamiseks toodi eraldi välja, mida lahenduseks nõutavalt programmilt oodatakse. Ülesannetele lisati juurde illustreerivad näited, et veelgi rohkem vältida olukorda, kus tekstist arusaamine jääb takistuseks probleemi lahendamisel. Mõnel juhul, kui lahenduses kasutati Pythoni moodulite funktsioone või lahenduskäik koosnes mitmest sammust, sisaldas ülesanne ka abistavat vihjet.

Ülesannetele saadud tagasisidest tehti ettepanekuid nii püstituse, nõuete kui ka teksti osas. Seitsmest ülesandest jäid kaks samaks, kahte muudeti vastavalt tundidele lihtsamaks ja üks kirjutati ümber, et see sobiks kasutamiseks teise teema juures.

Võimaliku edasise uurimusena võiks kõik loodud ülesanded koolides läbi katsetada, küsida õpilaste käest tagasisidet teksti arusaadavuse, ülesannete raskuastme ja huvitavuse kohta ning teha vajalikud parandused. Lisaks saaks võimaliku edasiarendusena luua neile ülesannetele ka automaatkontroll, mis võimaldaks õpilastel saada kohest tagasisidet ning hoida kokku õpetajate aega lahenduste hindamisel.

7. Viidatud kirjandus

[1] Juurak R. Millal saab informaatikast kohustuslik õppeaine? *Õpetajate Leht*, 2018. https://opleht.ee/2018/03/millal-saab-informaatikast-kohustuslik-oppeaine/ (07.05.2019)

[2] Leppik C., Haaristo H.-S., Mägi E. IKT-haridus: digioskuste õpetamine, hoiakud ja võimalused üldhariduskoolis ja lasteaias. 2017. http://www.praxis.ee/?download=&kccpid=&kcccount=http://www.praxis.ee/wp-content/uploads/2016/08/IKT-hariduse-uuring_aruanne_mai2017.pdf (23.03.2019)

[3] Viilukas K. Arvutialaste ainete õpe Eesti põhikoolides kahe maakonna näitel. TÜ arvutiteaduse instituudi bakalaureusetöö. 2016. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=53601&year=2016

[4] Saeli M., Perrenet J., Jochems W. M. G., Zwaneveld B. Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective. *Informatics in Education*. Vilnius University, 2011, 73–88.

[5] Programmeerimise õpetamine e-kursuse materjalide põhjal. https://courses.cs.ut.ee/2019/poemp/Main/HomePage (07.03.2019)

[6] Gümnaasiumi riiklik õppekava (17.02.2018). Riigi Teataja I. https://www.riigiteataja.ee/akt/129082014021?leiaKehtiv (08.03.2019)

[7] Uus hange: gümnaasiumi informaatika õppekava valikkursuste loomine, *HITSA*, 2018. https://www.hitsa.ee/uudised-1/uus-hange-gumnaasiumi-informaatika-oppekava-valikkursuste-loomine (07.05.2019)

[8] Õpik „Tarkvaraarendus". https://web.htk.tlu.ee/digitaru/tarkvara/ (07.05.2019)

[9] Puniste S. Eesti gümnaasiumides õpetatavad programmeerimiskursused. TÜ arvutiteaduse instituudi bakalaureusetöö. 2015. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=46301&year=2015

[10] Mis saab Eesti IT haridusest? Kes tuleb õppima? Kes kuidas õpib? Kes langeb välja? Mida saab keegi teha? Uuringuprojekti raport. Tartu, 2015. https://sisu.ut.ee/sites/default/files/ikt/files/iktraport_31.08.2015.pdf

[11] Kivisoo K. Pythoni programmeerimise algõppe e-kursuse „Programmeerimise alused" läbiviimine kombineeritud õppena koolis. TÜ arvutiteaduse instituudi magistritöö. 2018. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=62195&year=2018

[12] ProgeTiigri kogumik. http://www.progetiiger.ee/?q= (31.03.2019)

[13] Õppematerjal „Python koolis“. https://courses.cs.ut.ee/t/pythonkoolis (31.03.2019)

[14] E-koolikoti keskkond. https://e-koolikott.ee/ (07.05.2019)

[15] Informaatika digiõpik. https://courses.cs.ut.ee/t/digiopik/ (07.05.2019)

[16] Jürgenson A., Mägi E., Pihor K., Batueva V., Rozeik H., Arukaevu R. Eesti IKT kompetentsidega tööjõu hetkeseisu ja vajaduse kaardistamine. 2013. http://www.praxis.ee/fileadmin/tarmo/Projektid/Innovatsiooni_poliitika/Uuringu_lopparuanne.pdf (07.05.2019)

[17] Mannila L., Peltomäki M., Salakoski T. What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, 2006, volume 16:3, 211–227.

[18] Cooper S., Dann W., Pausch R. Alice: A 3-D Tool For Introductory Programming Concepts. *Journal of Computing Sciences in Colleges*. USA: Consortium for Computing Sciences in Colleges, 2000, 107–116.

[19] Robins A., Rountree J., Rountree N. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 2003, volume 13:2, 137–172.

[20] Gray W. D., Boehm-Davis D. A. Empirical Studies of Programmers: Sixth Workshop. United States of America: Ablex Publishing Corporation. 1996.

[21] Ahmadzadeh M., Elliman D., Higgins C. An Analysis of Patterns of Debugging Among Novice Computer Science Students. *ITiCSE '05 Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education.* Caparica, Portugal, 2005, 84–88.

[22] Merrill M. D. Components of Instruction Toward a Theoretical Tool for Instructional Design. *Instructional Science 29*. Netherlands: Kluwer Academic Publishers, 2001, 291–310.

[23] Staubitz T., Klement H., Renz J., Teusner R., Meinel C. Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses. *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE),* Zhuhai, China, 2015, 23–30.

[24] Meier H. Õppijate käitumismustrid programmeerimisülesande lahendamisel: logifailide analüüs. TÜ arvutiteaduse instituudi magistritöö. 2018. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=62175&year=2018

[25] Jadud M. C. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education.* 2005, volume 15:1, 25–40.

[26] Ihantola P., Vihavainen A., Ahadi A., Butler M., Börstler J., Edwards S. H., Isohanni E., Korhonen A., Petersen A., Rivers K., Rubio M. A., Sheard J., Skupas B., Spacco J., Szabo C., Toll D. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. *ITICSE-WGR '15 Proceedings of the 2015 ITiCSE on Working Group Reports.* Vilnius, Lithuania, 2015, 41–63.

[27] Rossum, G. Computer Programming for Everybody. Corporation for National Research Initiatives, 1999.

[28] Grandell L., Peltomäki M., Back R.-J., Salakoski T. Why Complicate Things? Introducing Programming in High School Using Python. *ACE '06 Proceedings of the 8th Australasian Conference on Computing Education*. Darlinghurst, Australia: Australian Computer Society, Inc, 2006, volume 52, 71–80.

[29] Winslow L. E. Programming Pedagogy – A Psychological Overview. *ACM SIGCSE Bulletin.* New York, USA, volume 28:3, 17–22.

[30] Matsuyama C., Nakashima T., Ishii N., Itoh H. Development of Teaching Material on the Web for Programming Course and its Evaluation. *SERA '05 Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications.* Washington, USA: IEEE Computer Society, 2005, 212–219.

[31] Nakashima T., Matsuyama C., Ishii N. Development of Teaching Materials for Students to Tackle Programming with Interest and its Effectiveness. *Java in Academia and Research*, 37–60.

[32] Gravemeijer K., Doorman M. Context Problems in Realistic Mathematics Education: A Calculus Course as an Example. *Educational studies in mathematics.* Netherlands: Kluwer Academic Publishers, 1999, volume 39, 111–29.

[33] Mikk J., Kukemelk H. The Relationship of Text Features to the Level of Interest in Science Texts. *TRAMES: A Journal of the Humanities & Social Sciences*. Estonia: Eesti Teaduste Akadeemia, 2010, 54–70.

[34] Fincher S., Robins A., Baker B., Box I., Cutts Q., de Raadt M., Haden P., Hamer J., Hamilton M., Lister R., Petre M. Predictors of Success in a First Programming Course. *Proceedings of the 8th Australasian Conference on Computing Education*. Australia: Australian Computer Society, Inc. 2006, volume 52, 189–196.

[35] Hill A. M. Problem solving in real-life contexts: An alternative for design in technology education. *International journal of technology and design education*. Netherlands: Kluwer Academic Publishers, 1998, volume 8, 203–220.

[36] Butler M., Morgan M. Learning challenges faced by novice programming students studying high level and low feedback concepts. *Proceedings ascilite Singapore 2007*. 2007, 99–107.

[37] Piteira M., Costa C. Learning computer programming: study of difficulties in learning programming. *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*. Lisbon, Portugal: ACM, 2013, 75–80.

[38] Chang S. E. Computer anxiety and perception of task complexity in learning programming-related skills. *Computers in Human Behavior*. Elsevier Ltd, 2005, volume 21:5, 713–728.

[39] Myers B. A. Visual programming, programming by example, and program visualization: a taxonomy. *CHI '86 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* Boston, USA: ACM, 1986, volume 17: 4, 59–66.

[40] Du Boulay B., O'Shea T., Monk J. The Black Box Inside The Glass Box: Presenting Computing Concepts to Novices. *International Journal of Man-Machine Studies*. 1981, volume 14:3, 237–249.

[41] Valikkursus „Rakenduste loomise ja programmeerimise alused". https://koolielu.ee/waramu/view/1-5171dab3-4720-4363-a956-b7602de68489 (07.05.2019)

[42] E-kursus „Programmeerimise alused". https://www.ut.ee/et/mooc/programmeerimise-alused (07.05.2019)

[43] E-kursus „Teeme ise arvutimänge – algus". https://courses.cs.ut.ee/2017/TIAM/fall (07.05.2019)

[44] Muhhin M. Ülesannete komplekt ainele "Teeme ise arvutimänge - algus". TÜ arvutiteaduse instituudi bakalaureusetöö. 2018. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=63661&year=2018

[45] E-kursus „Programmeerimise alused II". https://www.ut.ee/et/mooc/programmeerimise-alused-ii (07.05.2019)

[46] Annamaa A. Programmeerimise õpik. https://progeopik.cs.ut.ee/ (07.05. 2019)

[47] Hein H., Kiho J., Palm R., Tõnisson E. Programmeerimise eksamiülesannete kogu. Tartu: Tartu Ülikooli Kirjastus. 2007.

[48] Statistikaamet: Audiovisuaaltehnika, arvuti ja interneti-ühendusega leibkonnad. http://andmebaas.stat.ee/Index.aspx?lang=et&DataSetCode=KUT01 (07.05.2019)

[49] Wikipedia: List of best-selling albums. https://en.wikipedia.org/wiki/List_of_best-selling_albums (07.05. 2019)

[50] Charman D., Elmes A. Computer Based Assessment (Volume1): A guide to good practice. University of Plymouth: SEED Publications. 1998.

Lisad

I. Ülesandekogu

2 Kahemõõtmeline järjend

2.1 Lauatennis

Lauatennise võistlusel mängitakse viiest parem süsteemis. See tähendab, et mängu võitmiseks on vaja võita 3 setti. Seti võitmiseks on vaja saada 11 punkti nii, et mängijate punktide vahe oleks vähemalt 2 punkti. Seega skoor 11–8 on võit, kuid skoori 11–10 puhul tuleb mängida edasi, kuni punktivahe on vähemalt 2.

Võistlusel mängivad Eesti ja Soome mängija. Iga seti punktid on salvestatud kahemõõtmelisse järjendisse, kus esimesel kohal on Eesti mängija punktid ning teisel Soome mängija omad.

Kirjutada programm, mis seti tulemuste põhjal väljastab võitja ning lõpliku mänguseisu. Näide programmi tööst:

Tulemuste järjend:

```
tulemused = [[11, 8], [11, 5], [9, 11], [14, 12]]
```

Programmi töö:

```
>>> %Run lauatennis.py
Eesti võitis 3-1
```

2.2 Vale sõna välja

Kahemõõtmelises järjendis on kolm järjendit nelja sõnaga, millest üks ei sobi teiste sekka. Kirjutada programm, mis küsib kasutajalt iga järjendi kohta, millise indeksiga sõna ei sobi teiste hulka, ja kui kõigil kordadel on õigesti pakutud, väljastatakse ekraanile “Õige!”, vastasel juhul antakse veast teada. Näide programmi tööst:

```
>>> %Run vale_sona_valja.py
['kass', 'labidas', 'koer', 'karu']
Millise indeksiga sõna ei sobi hulka? 2
['Tartu', 'Võru', 'Haapsalu', 'New York']
Millise indeksiga sõna ei sobi hulka? 3
['lusikas', 'nuga', 'kapsas', 'kahvel']
Millise indeksiga sõna ei sobi hulka? 0
Mõni pakkumine oli vale!
```

2.3 Retseptid

Miina tegi suures koguses maasikamoosi, kuid ikka on tal üle nii maasikaid kui ka suhkrut. Ta otsib välja magustoitude retseptid ning lisab nende koostisosad kahemõõtmelisse järjendisse.

```
retseptid = [["muna", "suhkur", "jahu", "kohupiim", "vahukoor",
"maasikad"],["küpsised", "või", "toorjuust", "hapukoor", "jahu", "suhkur",
"muna"], ["banaanid", "maasikad", "apelsinimahl", "suhkur", "maitsestamata
jogurt"],["või", "suhkur", "tume šokolaad", "muna", "jahu"]]
```

Kirjutada programm, mis väljastab nende retseptide koostisosad, mis kasutavad nii suhkrut kui ka maasikaid.

Näide programmi tööst:

```
>>> %Run retseptid.py
['muna', 'suhkur', 'jahu', 'kohupiim', 'vahukoor', 'maasikad']
['banaanid', 'maasikad', 'apelsinimahl', 'suhkur', 'maitsestamata jogurt']
```

2.4 Hinnete tabel

Hinnete tabel on kahemõõtmelise järjendi kujul. Iga aine on eraldi järjendis, mis sisaldab esimesel kohal aine nimetust ning ülejäänud kohtadel hindeid. Hindeid võib olla 1 või rohkem.

Kirjutada programm, mis väljastab ekraanile iga aine keskmise hinde eraldi real.

Näide programmi tööst:

Hinnete tabel järjendina:

```
hinnete_tabel = [["eesti keel", 5, 4, 3, 4, 4, 3, 4, 4],
        ["matemaatika", 4, 4, 5, 5, 4, 5, 5, 4, 5],
        ["keemia", 4, 3, 3, 2, 3, 4],
        ["kirjandus", 5, 5, 5, 4, 5, 5, 4],
        ["bioloogia", 4, 5, 4, 4, 3, 5, 5, 5]]
```

Programmi töö:

```
>>> %Run hinnete_tabel.py
eesti keel: 3.9
matemaatika: 4.6
keemia: 3.2
kirjandus: 4.7
bioloogia: 4.4
```

2.5 Kohustuslik kirjandus

Kalev sai kirjanduse tunnis kohustusliku kirjanduse nimekirja. Kuna eelmine kord jäi tal mõni raamat lugemata, otsustab Kalev seekord oma aega tõhusalt kasutada ning teada saada, kui palju aega tal lugemisele kulub.

Esimese asjana uurib ta välja, mitu lehekülge on igas raamatus ning loob järjendi, mille elementideks on järjendid, mis sisaldavad raamatu pealkirja ja lehekülgede arvu.

```
raamatud = [['"Tõde ja õigus"', 495], ['"Protsess"', 160], ['"Dorian Gray
portree"', 240],['"Meister ja Margarita"', 468], ['"Keisri hull"', 318]]
```

Inimene loeb ilukirjandust keskmiselt 250 sõna minutis. See teeb lugemiskiiruseks ligikaudu üks lehekülg kahe minuti jooksul. Kalevil on iga päev aega lugeda 2 tundi.

Kirjutada programm, mis väljastab ekraanile

- nende raamatute pealkirjad ja iga raamatu lugemiseks kulunud päevade arvu (täisarvudes ümardatuna üles), mille lugemine võtab Kalevil aega rohkem kui 4 päeva,
- kõigi raamatute lugemiseks kuluv koguaeg (tundides ümardatuna esimese komakohani).

Näiteväljund:

```
>>> %Run kohustuslik_kirjandus.py
“Meie aja kangelane” - 4 päeva
“Toomas Nipernaadi” - 8 päeva
Kõigi raamatute lugemiseks kuluv aeg - 40.4 h
```

Vihjed:

1. Lugemiskiirus on vaja teisendada kujule lk/h.
2. Kiiruse valem $v = \frac{s}{t}$, kus v on kiirus, s on teepikkus ning t on aeg.
3. Üles ümardamiseks on funktsioon `ceil`, mis tuleb importida moodulist `math`.

   Näide:

```
>>> from math import ceil
>>> ceil(2.1)
3
>>> ceil(3.8)
4
```

3 Kahekordne tsükkel

3.1 Muster

Kirjutada programm, mis teeb mustri ühest veerandist terve sümmeetrilise mustri. Programm teeb kahemõõtmelisest järjendist suurusega nm kahemõõtmelise järjendi suurusega $(n + (n - 1))(m + (m - 1))$. Antud esimest veerandit tuleb peegeldada mööda sümmeetriatelgi (viimane rida ja viimane veerg).

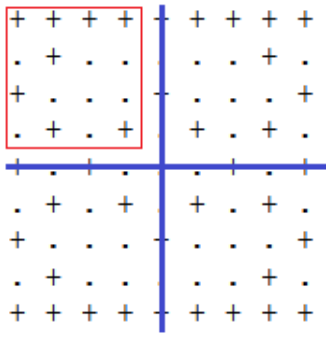
Näide programmi tööst:

Esimene veerand:

```
+ + + + +
. + . . .
+ . . . +
. + . + .
+ . + . .
```

Programmi töö:

```
>>> %Run muster.py
+ + + + + + + + +
. + . . . . . + .
+ . . . + . . . +
. + . + . + . + .
+ . + . . . + . +
. + . + . + . + .
+ . . . + . . . +
. + . . . . . + .
+ + + + + + + + +
```



3.2 Kaustade sorteerimine

Ühes suures kaustas on kaks kausta, milles leidub erinevat tüüpi faile.

```
suur_kaust = [["laud.jpg", "essee.docx"], ["ema_portree.jpg",
"minu_raamat.docx", "salajane_dokument.docx", "surmatants.jpg"]]
```

Kirjutada programm, mis sorteerib suure kausta ära nii, et selles leiduks kaks sama tüüpi faile sisaldavat kausta: dokumendid ja pildid. Näide programmi tööst:

```
>>> %Run sorteeri.py
suur_kaust
  pildid
    laud.jpg
    ema_portree.jpg
    surmatants.jpg
  dokumendid
    essee.docx
    minu_raamat.docx
    salajane_dokument.docx
```

3.3 Koristaja

Hotellis on toad järjest nummerdatud. Koristaja peab koristama igal paarisarvulisel korrusel ja igas paarisarvulise numbriga toas. Kirjutada programm, mis väljastab ekraanile kõik toad, kus koristaja käima peab.

Näide programmi tööst:

Järjend hotellitubadega:

```
toad = [[1, 2, 3],
   [4, 5, 6],
   [7, 8, 9],
   [10, 11, 12]]
```

Programmi töö:

```
>>> %Run koristaja.py
3
7
9
```

3.4 Kalorsus

Kalorisisaldus toidus ütleb ära, kui palju energiat keha selles toidus leiduvatest süsivesikutest, valkudest ja rasvadest saab. Kirjutada funktsioon, mis võtab argumendiks kahemõõtmelise järjendi, mille elementideks on toidukorrad järjenditena. Igal toidukorra kohta on antud süsivesikute, valkude ja rasvade arv grammides:

```
[süsivesik, valk, rasv]
```

Funktsioon tagastab, mitu kalorit on kõigi toidukordade peale kokku. On teada, et

- 1 g süsivesikus on 4 cal,
- 1 g valgus on 4 cal ja
- 1 g rasvas on 9 cal.

Näide funktsiooni tööst:

```
>>> print(kaloreid([[40, 53, 14], [62, 47, 26], [49, 43, 20], [10, 35, 22]]))
2094
```

3.5 Nummerda

Kinnisvarafirma ehitab mitut uut kortermaja, kusjuures iga majal on erinev korruste ja korterite arv korrusel. Korterid peavad olema igas majas nummerdatud samamoodi. Iga korteri number kahest osast: korruse number ja mitmes korter korrusel on. Näiteks korter number 23 asub teisel korrusel ning on kolmas korter sel korrusel.

Kirjutada funktsioon, mis võtab argumendiks korruste arvu ja korterite arvu ühel korrusel ning tagastab kahemõõtmelise järjendi, mis sisaldab iga korruse kohta nummerdatud kortereid.

Näide funktsiooni tööst:

```
>>> nummerda(3, 4)
[['11', '12', '13', '14'], ['21', '22', '23', '24'], ['31', '32', '33', '34']]
```

3.6 Minesweeper

Minesweeper on mäng, mille eesmärgiks on leida üles kõik suvaliselt paiknevad miinid ilma, et need plahvataksid.

Kirjutada programm, mis loob Minesweeperi miinivälja ja sisaldab funktsiooni `loo_manguvali`. Funktsioon võtab argumendiks mänguvälja pikkuse ja laiuse ning tagastab miinidega täidetud mänguvälja kahemõõtmelise järjendina.

Programm peab ka täidetud mänguvälja ekraanile kuvama.

Näide programmi tööst:

```
>>> loo_manguvali(8, 12):
. . . x x . . . x x . .
. . . . x . x . x . . x
. . x . . . . . . x . .
. x . . . . x . . . . x
. . . x . x x . x . x .
x . . . x . . x x . . .
. . . x . . x . . . x x
. . . . . x . x x . . .
```

Vihjed:

1. Juhusliku arvu genereerimiseks on funktsioon `randint()`, mis tuleb importida moodulist `random`.
2. Miini asukohast võib mõelda kui koordinaatidest (x, y), kus x oleks kahemõõtmelise järjendi esimese mõõtme element ehk üks järjenditest ja y teise mõõtme element ehk üks järjendi elementidest.

4 Andmevahetus

4.1 Tehnika

Tekstifailis tehnika.txt on andmed aastatest 2000–2006, mis kujutavad erineva tehnika, arvuti ja Internetiühendusega Eesti leibkondi. Iga aasta kohta on märgitud leibkondade osakaal protsentides kõigi leibkondade kohta.

Kirjutada programm, mis väljastab ekraanile nende seadmete puhul, mille osakaal on tõusnud rohkem kui 10 protsendipunkti võrra,

- seadme nimetuse ja
- kui suur on osakaalu muutus aastate 2000 ja 2006 vahel.

Andmed: Statistikaamet

Näide programmi tööst:

```
>>> %Run tehnika.py
Videomakk - 17
Muusikakeskus - 24
Personaalarvuti - 33
Interneti-ühendus - 32
```

Faili tehnika.txt sisu:

```
Värviteler 91 92 94 93 94    95     95
Videomakk  28 31 34 35 37    39     45
Raadio  89 90 89 86 88 89    89
Muusikakeskus27 28 34 37     40     45     51
Personaalarvuti 12 17 22     29     33     42     45
Interneti-ühendus  7  10     14     17     24     34     39
```

4.2 Hinnete tabel (muudatustega)

Hinnete tabel on CSV-failis hinded.csv, kus on eraldi ridadel erinevad ained. Iga aine kohta on toodud selle nimetus ja vastavad hinded. Kirjutada programm, mis väljastab ekraanile iga aine keskmise hinde eraldi real. Näide programmi tööst:

```
>>> %Run hinnete_tabel.py
eesti keel: 4.5
matemaatika: 3.0
keemia: 3.5
kirjandus: 4.8
bioloogia: 4.5
```

Faili hinded.csv sisu:

```
eesti keel,4,5,5,4,4,5
matemaatika,3,4,4,2,3,2
keemia,3,5,4,3,4,2
kirjandus,5,5,5,4,5,5
bioloogia,5,5,4,4,4,5
```

5 Andmestruktuurid

5.1 Kombinatsioonid

Kirjutada funktsioon, mis võtab argumendiks järjendi ja tagastab kõigi selle järjendi elementide paaride (kombinatsioonidega) järjendi. Elemendid paarides peavad olema erinevad, s.t (1, 1) ei ole lubatud.

Näide funktsiooni tööst:

```
>>> kombinatsioonid([1, 2, 3])
[(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]
```

5.2 Hamlet

Kirjutada programm, mis koostab sõnastiku iga failis hamlet.txt esineva sõna sagedusega ja väljastab ekraanile iga sõna ja tema sageduse eraldi reale. Sõned tuleks enne sõnastikku lisamist töödelda, s.t eemaldada kõik kirjavahemärgid, v.a mõttekriips.

Vihjed:

1. Failis olevate ridade lõpust saab reavahetussümboli `'\n'` eemaldamiseks kasutada funktsiooni `strip()`.
2. Sõnede järjendisse salvestamiseks on mõistlik kõigepealt read jaotada sõnedeks ning seejärel iga sõne juurest eemaldada ebavajalikud kirjavahemärgid.

Näide funktsiooni esimesest viiest väljundireast:

```
>>> %Run sagedused.py
to - 15
be - 4
or - 2
not - 2
that - 7
...
```

Faili hamlet.txt sisu:

```
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles
And by opposing end them. To die — to sleep,
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to: 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep, perchance to dream — ay, there's the rub:
For in that sleep of death what dreams may come,
When we have shuffled off this mortal coil,
Must give us pause — there's the respect
That makes calamity of so long life.
For who would bear the whips and scorns of time,
Th'oppressor's wrong, the proud man's contumely,
The pangs of dispriz'd love, the law's delay,
The insolence of office, and the spurns
That patient merit of th'unworthy takes,
When he himself might his quietus make
With a bare bodkin? Who would fardels bear,
To grunt and sweat under a weary life,
But that the dread of something after death,
The undiscovere'd country, from whose bourn
No traveller returns, puzzles the will,
And makes us rather bear those ills we have
Than fly to others that we know not of?
Thus conscience does make cowards of us all,
And thus the native hue of resolution
Is sicklied o'er with the pale cast of thought,
And enterprises of great pitch and moment
With this regard their currents turn awry
And lose the name of action. — Soft you now!
The fair Ophelia! Nymph, in thy orisons
Be all my sins remember’d.
```

5.3 Kangid

Jõusaal tahab klienditele pakkuda võimalust kasutada tõstmiskange, mis on fikseeritud raskusega vahemikus 10 – 25 kg. Selleks tuleb kangidele eelnevalt plaadid peale laadida, mis tulevad valikus 5 kg, 2,5 kg ja 1,25 kg ning lisaks kaalub kang ise 7,5 kg.

Kirjuta programm, mis vastavalt sõnastikus kangid antud elementide kogukaaludele (võtmetele) komplekteerib raskused ja lisab need võtmete väärtusteks.

```
kangid = {25: [], 22.5: [], 20: [], 17.5: [], 15: [], 12.5: [], 10: []}.
```

Näide programmi tööst:

```
>>> %Run kangid.py
kangid = {25: [7.5, 5, 5, 2.5, 2.5, 1.25, 1.25],
          22.5: [7.5, 5, 5, 2.5, 2.5],
          20: [7.5, 5, 5, 1.25, 1.25],
          17.5: [7.5, 5, 5],
          15: [7.5, 2.5, 2.5, 1.25, 1.25],
          12.5: [7.5, 2.5, 2.5],
          10: [7.5, 1.25, 1.25]}
```

5.4 Arva salasõna

Firma igal töötajal on tööarvutis isiklik kasutaja, millel on salasõna. Kuigi soovitatakse panna keeruline salasõna, on osad töötajad parooliks pannud 3-kohalise numbrikombinatsiooni (näiteks 106).

Kirjutada funktsioon `arva_salasona`, mis saab argumendiks sõnastiku, kus võtmeks on 3-numbriline salasõna enniku kujul ning väärtuseks sõne. Funktsioon genereerib kõikvõimalikud 3-kohalised kombinatsioonid numbritest 0–9 ja katsetab neid sõnastiku peal. Funktsioon tagastab järjendi, kus on kõik need sõnastiku väärtused, mille võtme programm õigesti pakkus.

Näide funktsiooni tööst:

```
>>> arva_salasona({(1, 2, 3): "Mati dokumendid", (8, 0, 7): "Kati dokumendid",
('p', 6, 'r', 4, 'n', 'd'): "Kristiina dokumendid"})
['Mati dokumendid', 'Kati dokumendid']
```

Vihje: koodi võib jagada osadeks. Näiteks kirjutada eraldi funktsioon kõigi kombinatsioonide leidmiseks ja kasutada seda `arva_salasona` funktsioonis.

5.5 Albumid

Failis albumid.txt on antud tähestikulises järjekorras nimekiri muusikaalbumitest, mis on maailmas müünud üle 30 miljoni koopia. Iga albumi kohta on toodud ära esitaja, albumi nimi, väljalaske aasta ja müüdud koopiate arv.

Kirjutada programm, mis koosneb kahest funktsioonist:

- `sorteeri`, mis jagab failis olevad albumid aastakümnete kaupa gruppidesse ning tagastab sõnastiku, kus võtmeteks on aastakümme ning väärtusteks kahemõõtmeline järjend albumitega
- `kuva`, mis võtab argumendiks eelmise funktsiooni poolt sorteeritud sõnastiku ning väljastab ekraanile iga aastakümne kohta albumite arvu ning kõige enim müünud albumi pealkirja ja artisti nime.

Näide programmi tööst:

```
>>> kuva(sorteeri("albumid.txt"))
1960: 2 albumit (The Beatles - Sgt. Pepper's Lonely Hearts Club Band)
1970: 9 albumit (Eagles - Their Greatest Hits (1971–1975))
1980: 7 albumit (Michael Jackson - Thriller)
1990: 12 albumit (Whitney Houston - The Bodyguard)
2000: 2 albumit (Adele - 21)
```

Andmed: Wikipedia

Faili albumid.txt sisu:

```
ABBA;Gold: Greatest Hits;1992;30
AC/DC;Back in Black;1980;50
Adele;21;2011;31
Alanis Morissette;Jagged Little Pill;1995;33
Bee Gees;Saturday Night Fever;1977;40
Bruce Springsteen;Born in the U.S.A.;1984;30
Celine Dion;Falling into You;1996;32
Celine Dion;Let's Talk About Love;1997;31
Dire Straits;Brothers in Arms;1985;30
Eagles;Hotel California;1976;42
Eagles;Their Greatest Hits (1971–1975);1976;51
Fleetwood Mac;Rumours;1977;40
Guns N' Roses;Appetite for Destruction;1987;30
James Horner;Titanic: Music from the Motion Picture;1997;30
Led Zeppelin;Led Zeppelin IV;1971;37
Madonna;The Immaculate Collection;1990;30
Meat Loaf;Bat Out of Hell;1977;43
Metallica;Metallica;1991;31
Michael Jackson;Bad;1987;32
Michael Jackson;Dangerous;1991;32
Nirvana;Nevermind;1991;30
Pink Floyd;The Dark Side of the Moon;1973;45
Pink Floyd;The Wall;1979;30
Santana;Supernatural;1999;30
```

```
Shania Twain;Come On Over;1997;40
The Beatles;1;2000;31
The Beatles;Abbey Road;1969;30
The Beatles;Sgt. Pepper's Lonely Hearts Club Band;1967;32
Various artists;Dirty Dancing;1987;32
Various artists;Grease: The Original Soundtrack from the Motion Picture;1978;38
Whitney Houston;The Bodyguard;1992;45
Michael Jackson;Thriller;1982;66
```

6 Rekursioon

6.1 Kiirloto

Kiirlotot mängides on võimalik võita 2, 10 või 50 eurot ning nende summade võiduvõimalused on vastavalt 20%, 15% ja 5%.

Mängija ostab pileti, mis maksab 2 eurot. Ta ostab uusi pileteid senikaua, kuni ta võidab rohkem kui 2 eurot või raha saab otsa. Kui mängija võidusummaks on 2 eurot, ostab ta selle raha eest uue pileti.

Kirjutada rekursiivne funktsioon `kiirloto`, mis leiab pileti võidusumma, mängib mängija reeglite järgi kiirlotot ning lõpuks tagastab enniku, mille esimeseks elemendiks on võidetud summa ja teiseks mängimise käigus ostetud piletite arv. Funktsioonil on kaks argumenti: mängija raha jääk ning ostetud piletite arvu loendur. Pileti võidusumma genereerimine on soovitatav teha abifunktsioonina, mida rekursiivses funktsioonis välja kutsutakse.

Näide funktsiooni tööst:

```
>>> print(kiirloto(5, 1))
(10, 3)
>>> print(kiirloto(5, 1))
(0, 3)
>>> print(kiirloto(5, 1))
(50, 1)
```

6.2 Kiirloto (muudatustega)

Kiirlotot mängides on võimalik võita 4 eurot, mille võiduvõimalus on 30%. Mängija ostab pileti, mis maksab 2 eurot. Ta ostab uusi pileteid senikaua, kuni uue ostuks enam raha ei jätku.

Kirjutada programm, mis mängib mängija reeglite järgi kiirlotot ja väljastab ekraanile, mitu eurot mängimisest tagasi saadi. Programmi sisaldab rekursiivset funktsiooni `kiirloto`, millel on kaks argumenti: mängija raha jääk ja võidetud rahasumma. Töö lõpuks tagastab funktsioon enniku kujul `(raha_jääk, võidusumma)`.

Näide funktsiooni `kiirloto` tööst:

```
>>> print(kiirloto(7, 0))
(1, 4)
```

Näide programmi tööst:

Funktsiooni väljakutse:

```
raha = 7
mang = kiirloto3(raha, 0)
```

Programmi töö:

```
>>> %Run loto.py
Enne piletite ostmist oli raha 7 eurot
Peale piletite ostmist on alles -1 eurot
```

6.2 Püramiid

Kirjutada rekursiivne funktsioon, mis väljastab ekraanile püramiidi kuju. Püramiidi iga järgmine tase erineb eelmisest 4 võrra. Funktsioonil on kaks argumenti, millest esimene on püramiidi põhja laius ning teine jooksva taseme laius.

Näide funktsiooni tööst:

```
>>> pyramiid(12, 1)
    .
  .....
.........
>>> pyramiid(15, 1)
      .
    .....
  .........
.............
```

6.3 Romb

Kirjutada rekursiivne funktsioon, mis nüüd väljastab ekraanile kahest kolmnurgast koosneva rombi. Peab arvestama sellega et kolmnurga põhjasid kaks korda järjest ei ole vaja väljastada. Funktsiooni argumentideks on ühe kolmnurga põhja laius, jooksva taseme laius ning eelmise taseme laius.

Näide funktsiooni tööst:

```
>>> romb(24, 1, 0)
           .
         .....
       .........
     .............
   ...............
.....................
   ...............
     .............
       .........
         .....
           .
>>> romb(11, 1, 0)
     .
   .....
.........
   .....
     .
```

6.4 Mäng

Võitlusmängus on tegelsteks mängija ja tema vastane, kes löövad kordamööda. Esimesena käib alati mängija. Kahju, mida mõlemad tegelase löök teha võib, jääb vahemikku 0 kuni 5. Mäng lõpeb, kui ühe tegelase elud otsa saavad.

Kirjutada funktsioon `voitlusmang`, millele saab argumentidena ette anda käikude ja mõlema tegelase elude arvu. Iga löögi järel tuleb ekraanile kuvada mängu seis, kus näidatakse mängija ja vastaste elusid. Töö lõpuks tagastab funktsioon, mitu käiku võidu saavutamiseks kulus.

Näide funktsiooni tööst:

```
>>> print("Käikude arv:", mäng(0, 10, 10))
Elusid: 10 - Vastasel elusid: 10
Elusid: 8 - Vastasel elusid: 10

Elusid: 8 - Vastasel elusid: 5
Elusid: 4 - Vastasel elusid: 5

Elusid: 4 - Vastasel elusid: 3
Elusid: 3 - Vastasel elusid: 3

Elusid: 3 - Vastasel elusid: 1
Elusid: 0 - Vastasel elusid: 1
Oled surnud
Käikude arv: 3
```

6.5 Liida (lisalugemine)

Kirjutada rekursiivne funktsioon `liida`, mis liidab pikemale järjendi elementidele lühema järjendi vastavad elemendid. Näiteks järjendite [1, 2, 3, 4] ja [5, 6] korral on tagastatav tulemus [6, 8, 3, 4].

Näide funktsiooni tööst:

```
>>> liida([1, 2, 3, 4], [5, 6], 0)
[6, 8, 3, 4]
```

6.6 Eralda (lisalugemine)

Kirjutada rekursiivne funktsioon `eralda`, mis teeb paaride järjendist järjendite paari, s.t eraldab järjendis oleva iga paari esimese ja teise elemendi, paneb need kahte eraldi järjendisse ning lõpuks tagastab paari, mille elementideks on eraldatud elemente sisaldavad järjendid. Näiteks järjendi [("abc", 123), ("def", 456), ("ghi", 789)] korral tagastatakse (['abc', 'def', 'ghi'], [123, 456, 789]).

Näide funktsiooni tööst:

```
>>> eralda([(1,'x'), (4,'1'), (2,'p')], [], [], 0)
([1, 4, 2], ['x', '1', 'p'])
```

6.7 Kohver (lisalugemine)

Lennukisse võetaval käsipagasil on kaalupiirang 8 kg. Kirjutada rekursiivne funktsioon, mis tagastab pakitavate asjade sellise kombinatsiooni, et kohver oleks võimalikult täis ehk kaaluks üles ümardatuna 8 kg. Pakitavad asjad on järjendi kujul, mille elemendid on ennikud eseme nimetusest ja selle kaalust kilogrammides.

Näide funktsiooni tööst:

```
>>> asjad = [("kampsun", 0.4), ("supipott", 2), ("püksid", 0.2),
("vihmajope", 1), ("kiirnuudlid", 3), ("tossud", 0.9),
("joogivesi", 5), ("lauaarvuti", 10),
("fotoaparaat", 0.7), ("dokumendid", 0.1)]

>>> print(kohver(asjad, [], 0, 0))
[('kampsun', 0.4), ('supipott', 2), ('vihmajope', 1), ('kiirnuudlid', 3),
('tossud', 0.9), ('fotoaparaat', 0.7)]
```

6.8 Shuffle (lisalugemine)

Kirjutada rekursiivne funktsioon, mis võtab võtab ette järjendi kujul laulude esitusloendi ning hakkab laule segama (ingl *shuffle*). Laulud, mida on juba esitatud, ei tohi korduda, seega juba mängitud laulud on vaja kuidagi salvestada (näiteks teise järjendisse). Kasutajale antakse teada, millist laulu parajasti mängitakse. Selle peale saab kasutaja teada anda, et soovib järgmist laulu. Funktsioon lõpetab oma töö, kui kõik laulud esitusloendist on mängitud või kui kasutaja kirjutab "stopp". Peale töö lõpetamist tagastab funktsioon, mitu lugu kokku mängiti.

Näide funktsiooni tööst:

```
>>> print("Mängitud lugude arv: " + str(shuffle(["Oma laulu ei leia ma üles",
"Hommik", "Ma olen Tartu", "Lilleke rohus"], [])))
Hetkel mängib "Lilleke rohus"
edasi
Hetkel mängib "Oma laulu ei leia ma üles"
edasi
Hetkel mängib "Hommik"
stopp
Mängitud lugude arv: 3
```

II. Õpetaja koostatud tööleht ülesande „2.4 Hinnete tabel“ põhjal

Andmevahetus. CSV-failist sisselugemine. Paaristöö

I osa. CSV-faili loomine

1) Avage Excel ning tehke tabel vähemalt 5 õppeainega ja vastavalt 6 hinnet. Näide:

	A	B	C	D	E	F	G
1	eesti keel	4	5	3	5	2	5
2	matemaatika	4	4	4	5	2	5
3	keemia	5	3	4	2	4	4
4	kirjandus	2	5	5	3	3	5
5	bioloogia	3	3	4	4	2	3

PS! Ka Excelis saab kasutada random funktsiooni trükkides lahtrisse: =RANDBETWEEN(2;5)

2) Salvestage table CSV-formaadis: Fail → Save as → Browse → kirjutage failinimi ning valige “Save as type” valikust **CSV (Comma Delimited)**.

3) Avage Thonny või Notepad ning avage loodud CSV-fail selle programmiga. Kirjutage tulemus siia:

II osa. CSV-faili sisselugemine

1) Kasutades allolevat programmi, lugege failist andmed kahekordsesse järjendisse `hinded`.

```
fail = open("Book1.csv", encoding="UTF-8")


hinded = []


for rida in fail:  # loeme ridahaaval faili sisse
    ühe_aine_hinded = []
    osad = rida.split(";")  # teeme rea järjendiks
    for osa in osad:  # rida osade kaupa alamjärjendiks
        ühe_aine_hinded.append(osa.strip('\n')) # ühe aine hinded
    hinded.append(ühe_aine_hinded) # õpilase hinnetele alamjärjend
juurde


fail.close()


print(hinded) # väljastatakse ekraanile kõik hinded
```

PS! Katsetage programmi muutes hindeid CSV-failis. Kas töötab?

III osa. Tegeleme järjenditega

1) Kirjutage kasti sisse programmiread, mis väljastavad ekraanile teie failist kõikide ainete hinded eraldi real järjendina.

```

```

Näide tulemusest:

```
>>> %Run csv_failist.py
  ['eesti keel', 4, 5, 3, 5, 2, 5]
  ['matemaatika', 4, 4, 4, 5, 2, 5]
  ['keemia', 5, 3, 4, 2, 4, 4]
  ['kirjandus', 2, 5, 5, 3, 3, 5]
  ['bioloogia', 3, 3, 4, 4, 2, 3]
```

2) Kirjutage kasti programmiread, mis väljastavad ekraanile ühe aine hinded järjendina:

```

```

Näide tulemusest:

```
>>> %Run csv_failist.py
  ['keemia', 5, 3, 4, 2, 4, 4]
```

2) Looge uus järjend (näiteks keemia_hinded) iga aine hinnete jaoks. Kirjutage üks näidis programmirida uue järjendi tegemiseks kasti:

```

```

Näide tulemusest kui kirjutada `print(keemia_hinded)`:

```
>>> %Run csv_failist.py
  ['keemia', 5, 3, 4, 2, 4, 4]
```

3) Looge iga aine hinnete jaoks uued järjendid (näiteks keemia_hinded_arvud) aine hinnete hoidmiseks arvudena. Proovige selleks allolevat programmi:

```
keemia_hinded_arvud = []
for i in range(1, len(keemia_hinded)):
    keemia_hinded_arvud.append(int(keemia_hinded[i]))
print(keemia_hinded_arvud)
```

Lisaülesanne

Seejärel valige allolevatest nimekirjast **vähemalt 3** punkti täitmiseks:

- Leidke kõigi ainete kohta parim ja halvim hinne ning väljastage need ekraanile.
- Leidke kõigi ainete puhul hinnete keskmised ning väljastage need ekraanile.
- Leidke kõigi ainete puhul positiivsete ("3", "4" või "5") hinnete arv.
- Leidke üle kõigi ainete hinnete keskmine.
- Leidke kõigi ainete puhul, mitu korda esineb hinne "5".
- Mõelge ise välja üks programm hinnetega tegelemiseks.

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Sandra Puusepp**,

(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Programmeerimisülesannete kogu gümnaasiumi valikkursusele „Tarkvaraarendus“**,

   (*lõputöö pealkiri*)

   mille juhendajad on Eno Tõnisson ja Tauno Palts,

   (*juhendajate nimed*)

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Sandra Puusepp

10.05.2019