TARTU ÜLIKOOL

Arvutiteaduse instituut

Informaatika õppekava

Patrik Pruunsild

Tartu Ülikooli kursuse “Programmeerimise alused” uus ülesannete kogu

Bakalaureusetöö (9 EAP)

Juhendaja: Heidi Meier

Tartu 2022

Tartu Ülikooli kursuse “Programmeerimise alused” uus ülesannete kogu

Lühikokkuvõte:

Tartu Ülikooli kursusel “Programmeerimise alused” kasutatavate ülesannete suurema varieeruvuse saavutamiseks koostati käesoleva bakalaureusetöö raames uus praktikumi- ja koduülesannetest koosnev ülesannete kogu. Ülesannete kogu väljatöötamisel lähtuti eesmärgist luua elulise ja praktilise sisu ning selge tekstiga ülesanded, mis on tudengite hinnangul huvitavad ja mille keerukus suureneb kursuse arenedes. Koostatud ülesandeid piloteeriti osaliselt 2022. aasta kevadsemestril ja tudengitel paluti ülesannete kohta anda tagasisidet. Tagasisidest saadud vastuseid analüüsiti, et mõista ülesannete tajutavat keerukust, huvitavust ja tekstide selgust ning võrrelda varasemaid ja käesoleva töö raames loodud ülesandeid eelmainitud tunnuste lõikes. Tudengid hindasid varasemaid ja käesoleva töö raames loodud ülesandeid huvitavuse ja tekstide selguse poolest võrdselt, kuid hindasid käesoleva töö raames koostatud ülesandeid varasematest ülesannetest keerukamaks.

Võtmesõnad:

Programmeerimisülesanded, õppematerjalid, programmeerimise algõpe

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

A Collection of Programming Tasks for the University of Tartu course "Introduction to Programming"

Abstract:

In order to achieve a greater variety of tasks used in the course "Introduction to Programming" taught at University of Tartu, a new collection of programming tasks consisting of practicum and homework tasks was developed as a result of this bachelor's thesis. The development aimed at creating tasks of vital and practical content and with clear descriptions that students find interesting. The collection of tasks was created to support the idea that the complexity of the tasks should increase as the course progresses. The developed tasks were partially piloted in the spring semester of 2022 together with older tasks from previous years, and students were asked to provide feedback on all of the tasks. The responses received from the feedback were analyzed in order to understand the perceived complexity and interestingness of the tasks and clarity of the descriptions during the 6 weeks of homework and practicum task solving, and to compare the older tasks and the tasks created in the process of this thesis according to the above-mentioned characteristics. The students evaluated the older tasks and the tasks created in the process of this bachelor's thesis equally in terms of interestingness and clarity of the descriptions, but they assessed the tasks created in the process of this thesis more complex than the older tasks.

Keywords:

Programming tasks, teaching materials, teaching programming to novices

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

Sisukord

1. Sissejuhatus 6

2. Pythoni algkursused ja Python esimese programmeerimiskeelena 8

2.1 Veebirakenduse põhised Pythoni programmeerimiskeele kursused algajatele 8

2.1.1 Helsingi Ülikooli välja töötatud Pythoni MOOCid 8

2.1.2 Õppekeskkond Python Principles 8

2.1.3 Sololearni kursus „Python Core" 9

2.2 Ülikoolide pakutavad veebipõhised Pythoni programmeerimiskeele kursused algajatele 9

2.2.1 Massachusettsi Tehnoloogiainstituudi loetav Pythoni programmeerimiskeele algkursus 9

2.2.2 Michigani ülikooli kursus „Python for Everybody" 10

2.3 Tartu Ülikoolis loetavad Pythoni programmeerimiskeele põhised kursused 10

2.3.1 "Programmeerimine" (6 EAP) 10

2.3.2 "Programmeerimise alused" (3 EAP) ja "Programmeerimise alused II" (3 EAP) 10

3. Programmeerimise õpetamine ja ülesannete koostamine algajatele 11

3.1 Programmeerimise õpetamine algajatele 11

3.2 Esmase programmeerimiskeele valik programmeerimise õpetamisel 14

3.3 Alternatiivsed meetodid programmeerimise õpetamiseks 16

3.4 Programmeerimisülesannete koostamine algajatele 17

4. Metoodika 21

4.1 Programmeerimisülesannete väljatöötamine 21

4.2 Väljatöötatud ülesannete piloteerimine ja tagasiside 23

5. Tulemused ja arutelu 25

5.1 Kolmele ja kahele tagasisidevormile vastanute tajutava ülesannete keerukuse, huvitavuse ja tekstide selguse analüüs 25

5.2 Piloteeritud ülesannete tajutava keerukuse, huvitavuse ja tekstide selguse võrdlus varasemate ülesannetega 27

5.3 Parandusettepanekud ülesannetele ja tudengite meelestatus ülesannete suhtes 29
6. Kokkuvõte 31
7. Viidatud kirjandus 33
Lisad 37
I. Ülesannete kogu 37
II. Küsitluse ankeedi näidis 69
III. Litsents 78

1. Sissejuhatus

Tartu Ülikoolis saab kevadsemestritel võtta osa kursusest “Programmeerimise alused” [1], mis kestab 7 nädalat ja mille läbimise eest annab ülikool tudengile 3 EAP-d (Euroopa ainepunktisüsteemi ainepunkt). Kursusel osalemiseks puuduvad eeldusained, mis tähendab, et kursusest saavad osa võtta tudengid, kes ei ole varem programmeerimisega kokku puutunud. Kursusel õpetatakse programmeerimise universaalseid põhitõdesid Pythoni programmeerimiskeele abil ja suure osa kursusel tehtavast individuaalsest tööst moodustab kodu- ja praktikumiülesannete lahendamine. “Programmeerimise aluste” kursusel lahendatakse kodu- ja praktikumiülesandeid kuuel õppenädalal ja kokku saab iga tudeng lahendada üldjuhul 3 praktikumi- ja 3 koduülesannet õppenädala vältel, kuid nädalati võib ülesannete arv kohati erineda.

Kursusel “Programmeerimise alused” on õppeaastate vältel kasutatud paljuski samasid ülesandeid [2, 3, 4, 5], mistõttu leidsid ainekursuse läbiviijad, et suurema varieeruvuse saavutamiseks kasutatavates ülesannetes tuleks välja töötada kursusele uus ülesannete kogu. Vähe varieeruvate ülesannete kasutamise tulemusena võivad ülesanded sisult muutuda tudengitele ebahuvitavaks ja mitteaktuaalseks ning suureneb tõenäosus, et kursuse läbinud on loonud lahendatud ülesannete kogu, mida uued õppurid kursuse lihtsamaks läbimiseks võivad ära kasutada. Bakalaureusetöö eesmärk on luua uus ülesannete kogu Tartu Ülikoolis loetava ainekursuse “Programmeerimise alused” jaoks. Ülesannete väljatöötamisel lähtutakse peamiselt eesmärgist luua elulise ja praktilise sisuga ülesanded, sest on leitud, et ülesannete elulisus ja praktilisus motiveerivad õpilasi neid lahendama [6, 7] ning soodustavad ülesannete järjepideva lahendamise läbi programmeerimise põhitõdede selgeksõppimist ja paremate tulemuste saamist [8]. Robins jt [6] hinnangul tuleks individuaalse töö keerukust ajas tõsta, mistõttu on autor lisaks ülesannete elulisusele ja praktilisusele seadnud eesmärgi koostada selge tekstiga ülesandeid, mille keerukus ajas kasvab, kuid huvitavus säilib.

Bakalaureusetöö esimene peatükk teeb ülevaate erinevatest programmeerimiskursustest, mida viiakse läbi, kasutades Pythoni programmeerimiskeelt. Peatüki eesmärk on mõista lisaks kursusel “Programmeerimise alused” paika pandud struktuurile ka alternatiivsete kursuste ülesehitust ja sisu, et nendest ülesannete kogu väljatöötamisel inspiratsiooni ammutada. Teine peatükk annab ülesannete kogu väljatöötamise protsessile teoreetilise tausta. Uuritakse

programmeerimise õpetamise kohta, peamiselt Pythoni programmeerimiskeele vaatepunktist, ja kogutakse teaduspõhist informatsiooni ülesannete kogu väljatöötamise kohta. Kolmandas peatükis kirjeldatakse ülesannete väljatöötamise protsessi ja ülesannete kogu osalist piloteerimist ning tudengitelt tagasiside küsimist 2022. aasta kevadsemestril. Neljandas peatükis analüüsitakse tudengitelt saadud tagasisidet ja võrreldakse piloteeritavate ülesannete tajutavat keerukust, huvitavust ja tekstide selgust varasemate ülesannetega.

2. Pythoni algkursused ja Python esimese programmeerimiskeelena

Selles peatükis tutvustatakse programmeerimiskursuseid, mis on peamiselt algajatele suunatud ja tuginevad Pythoni programmeerimiskeelele. Pythoni programmeerimiskeelel tuginevad programmeerimiskursused valiti seetõttu, et käesoleva bakalaureusetöö käigus valmivad ülesanded kursuse jaoks, mis programmeerimise algtõdesid õpetades sellele programmeerimiskeelele tugineb.

2.1 Veebirakenduse põhised Pythoni programmeerimiskeele kursused algajatele

2.1.1 Helsingi Ülikooli välja töötatud Pythoni MOOCid

Helsingi Ülikooli veebipõhised MOOCid [9] (ingl *massive open online course)* on loodud ülikooli arvutiteaduse valdkonnas loetavate ainekursuste „Introduction to Programming" ja „Advanced Course in Programming" õpetamiseks. Kursus on Helsinki Ülikooli tudengitele suunatud, kuid kasutaja loomisel avaneb võimalus kõigil soovijatel materjalidega tutvuda ja probleemülesandeid lahendada. Kursusel õpetatakse Pythoni põhitõdesid erinevates peatükkides sisukate, kuid mitte liigdetailsete kirjelduste abil. Kasutaja loomisel avaneb sellest kursusest osavõtjal võimalus sooritada veebilehel toetatud redaktoris harjutusülesandeid, mis on paigutatud teemade lühidalt kokku võtmiseks peatükkide vahele. Lehel on sektsioon eksami sooritamiseks, mis on mõeldud eelmainitud Helsingi Ülikooli ainekursustel osalejatele, kuid millest saab soovi korral osa võtta ka ainekursuste väliselt.

2.1.2 Õppekeskkond Python Principles

Python Principles [10] (varasemalt oli õppekeskkonna nimi Learnival) on ülesehituselt sarnane Helsingi Ülikooli MOOCidega, kuid pakub autori hinnangul rohkem interaktiivsust. Õppimiseks loodud veebirakendus on kujundatud järgmiselt: lehe keskpiirist vasakul pool asetsevad õppematerjalid ja ülesannete tekstid ning lehe keskpiirist paremal pool asetseb koodi kirjutamiseks mõeldud redaktor. Kujunduslik lahendus toetab uue õppija võimalust katsetada õpitava käigus pidevalt ülesandeväliseid lühiprogramme, mille ideed võivad olla materjalist ja ülesannetest inspireeritud. Järgnevatele materjali ja ülesannetega lehtedele kasutajat ei lubata, kui ei ole kirja pandud korrektset lahendust eesoleva lehe probleemülesandele.

2.1.3 Sololearni kursus „Python Core”

“Python Core” [11] erineb Helsinki ülikooli MOOCist ja õppekeskkonnast “Python Principles” tekstipõhise materjali lakoonilisuse tõttu, tuginedes Pythoni programmeerimiskeele teooria õpetamisele läbi lünkade täitmist nõudvate ülesannete. “Python Core” annab olulisemate peatükkide lõpus osavõtjale võimaluse kontrollida enda teadmisi lühiprogrammi kirjutamise abil. Kursuse algmaterjalid on tasuta, kuid teatud peatükkide läbimise järel ei saa kasutaja maksumüüri tõttu kursust tasuta jätkata. “Python Core” kursust on läbinud tuhanded inimesed ning kursusel toodud ülesannete alla on võimalik jätta kommentaare, mille abil on osavõtjal vajadusel võimalik õpitava osas täpsustusi saada. Eeskätt algajatele mõeldud kursus käsitleb hilisemas faasis algajatele komplekssemaid teemasid: funktsionaalne programmeerimine, objektorienteeritud programmeerimine, regulaaravaldised.

2.2 Ülikoolide pakutavad veebipõhised Pythoni programmeerimiskeele kursused algajatele

2.2.1 Massachusettsi Tehnoloogiainstituudi loetav Pythoni programmeerimiskeele algkursus

Uuenduslike veebirakenduse põhiste kursuste kõrval pakutakse veebiloengu vormis kursuseid. Veebilehe edX [12] vahendusel on võimalik osa võtta tehnikateaduste valdkonnas ühe tunnustatuima ülikooli Massachusettsi Tehnoloogiainstituudi algajatele suunatud programmeerimiskursust Pythoni programmeerimiskeeles. Kursuse eesmärk on 9 nädala vältel õpetada osalejat lahendama päriselulisi analüütilise sisuga probleeme Pythoni programmeerimiskeele abil. Kursus on osavõtjale soovi korral tasuta, kuid tasu eest on võimalik taotleda kursuse läbimise tunnistus. Tegemist on ühe populaarseima programmeerimiskursusega, mille ühele toimumiskorrale registreerib end kuni kaks miljonit osavõtjat.

2.2.2 Michigani ülikooli kursus „Python for Everybody”

Michigani ülikool viib läbi avatud kursust „Python for Everybody” [13], mis kestab 32 nädalat. Selle kursuse käigus on osalejal võimalus õppida Pythoni programmeerimiskeelele tuginedes programmeerimise alustõdesid, andmestruktuure ja nende kasutust, andmeanalüüsi ja operatsioone andmebaasidega. Kursus koosneb viiest osast, alustades programmeerimise põhitõdedega ning lõpetades õpetatu praktiseerimisega andmebaasidega opereerides.

2.3 Tartu Ülikoolis loetavad Pythoni programmeerimiskeele põhised kursused

2.3.1 “Programmeerimine” (6 EAP)

Tartu Ülikooli kursusel „Programmeerimine” [14] õpetatakse programmeerimise põhikonstruktsioone ja esmaseid oskusi algoritmide ning programmide koostamiseks. Kursusel õpetatakse andmetüüpe ja andmestruktuure, programmide töö analüüsimist ja probleemülesannete lahendamiseks algoritmide koostamist. Kursuse käigus saab osaleja koos teiste kursusest osavõtjatega teostada projekti. Kursusel osalemine ei eelda varasemaid programmeerimisalaseid teadmisi.

2.3.2 “Programmeerimise alused” (3 EAP) ja “Programmeerimise alused II” (3 EAP)

Tervikuna moodustavad kursused „Programmeerimise alused” ja „Programmeerimise alused II” kursuse „Programmeerimine”. Kursus „Programmeerimise alused” käsitleb programmeerimise baaskonstruktsioone: hargnemist, tingimuslauseid; tsüklit ja alamprogramme, õpetades neid esitama plokkskeemide ja programmilõikudena [15]. Lisaks tutvustatakse järjendit, funktsioonide deklareerimist, andmevahetust, failist kirjutamist ja graafilise kasutajaliidese rakendamist koodile [1]. Selle kursuse jätkukursus „Programmeerimise alused II” käsitleb kahemõõtmelist järjendit, andmestruktuure, kahekordset tsüklit, viitamist, muteerimist ja rekursiooni [16].

3. Programmeerimise õpetamine ja ülesannete koostamine algajatele

Selles peatükis tutvustatakse programmeerimise õpetamisega seotud lähenemisvõtteid ja uuringuid ning algajatele programmeerimisülesannete koostamist. Peatükis tehakse ülevaade õpetatava esmase programmeerimiskeele valiku olulisusest ja tutvustatakse tekstipõhise programmeerimise väliseid meetodeid programmeerimise õpetamiseks. Käsitletakse ka Pythoni programmeerimiskeelt esimese õpitava programmeerimisekeelena.

3.1 Programmeerimise õpetamine algajatele

Programmeerimise õpetamisel, nagu õpetatavate oskuste puhul üldiselt on oluline kasutada meetodeid, mis on efektiivsed programmeerimisoskuse õppimise suhtes. Õpetajal tasub lähtuda printsiipidest, mis soodustavad õpilase selget arusaama ja huvi õpetatava osas. Üldisteks efektiivseteks õpetamismeetoditeks peetakse selgesõnalist ja kaasahaaravat õpetamist; empiiriliselt tõestatud auditooriumi haldamise praktikaid, näiteks laudade ja toolide asetust klassiruumis; usalduse ja austuse tekitamist õpetaja ja õpilase vahel [17]. Kaasahaarav õpetamine tugineb paljuski materjali elulisusele ja praktilisusele, mistõttu tuleb õpetajal nendele aspektidele tähelepanu pöörata. Kui mõne käegakatsutava ja intuitiivselt mõistetava oskuse puhul on kaasahaaravus ehk iseeneslik, siis programmeerimise õpetamise puhul, mis üldiselt tähendab õpilase jaoks uute ja senitundmatute kontseptsioonide tundmaõppimist, tuleb kaasahaaravus seada prioriteediks. Materjali elulisus ja praktilisus on esmane prioriteet ka käesoleva bakalaureusetöö raames välja töötatava ülesannete kogu puhul, mistõttu on eraldi programmeerimise õpetamisele suunatud meetoditega tutvumine olulisel kohal.

Programmeerimise õpetamisel on oluline kaardistada keerulisemad teemad, milleks peetakse näiteks rekursiooni ja samuti “Programmeerimise aluste” kursusel käsitletavaid tsüklite ja tingimuslausete teemasid [7], et olla nende õpetamise osas tähelepanelikum selgesõnalise materjali väljatöötamisel ja kaasahaaravate probleemülesannete koostamisel. Rekursioon ei ole ühelgi Tartu ülikoolis pakutaval programmeerimiskursusel esimeste õpetatavate teemade seas [6, 7], kuid samas on oluline, et ka varasemad lihtsamad teemad oleksid kaasahaaravalt edasi antud, et õpilased keerulisemate teemadeni jõuaks. Kui teema on kaasahaarav, on õpilasel suurem motivatsioon sellega tegeleda. Jenkins [18] kõrvutab programmeerimise õpetamise perspektiivis õpetaja ja motivaatori rolli, väites, et õpetaja ei saa olla kõigest kursuse jaoks

olulise teabe edasiandja, kuid ka sisemise motivatsiooni tekitaja. Kuna motivatsioon on paljuski kaasahaaravuse produkt, on õpetajal hea võimalus õpilase motivatsiooni programmeerimist õppida just eluliste ja praktiliste näidete põhjal tõsta.

Programmeerimise õpetamisel tuleb huvitavate ja eluliste probleemülesannete kõrval tähele panna ka muid sujuvaks õppimiseks olulisi aspekte. Robins jt [6] läbi viidud teoreetilises uuringus kaardistati programmeerimise õpetamise ja õppimise problemaatilisi külgi ning pakuti seejuures välja sobivaid õpetamisvõtteid, mis jaotati neljaks:

1. Programmeerimiskursuse materjali keerukus peaks ajas kasvama, alustades lihtsatest teemadest ja liikudes süstemaatiliselt keerukamate teemade juurde, kus lihtsate teemade tundmine toestab keerulisemate teemade selgeksõppimist.
2. Algajaid programmeerijaid tuleks õpetada lähenema probleemidele astmeliselt, luues kõigepealt arusaama probleemi sisust, seejärel mõttelise lahenduse ning viimaks reaalse programmi, mis probleemi lahendab. Lisaks tuleks arendada programmi töö silumise (ingl *debugging*) oskusi, et programm korrektselt töötaks.
3. Algajatel programmeerijatel on programmeerimisalased teadmised tüüpiliselt puudulikud, eeskätt tsüklite, tingimuslausete, järjendite ja rekursiivsete programmide puhul. Õppematerjalides tuleks nende teemade puhul suuremat rõhku panna nii teoreetilises sisus kui ka praktilise sisuga ülesannete väljatöötamisel.
4. Algajate programmeerijate õpetajatel on oluline mõista, mis tüüpi programmeerijaga on õpilase näol tegemist. Perkinsi jt [19] järgi on kahte tüüpi programmeerimise õppijaid. On peatujad (ingl *stoppers*), kellel tekivad tõrgetega kokkupuutumisel tugevad negatiivsed emotsionaalsed reaktsioonid ja kes seejärel annavad lihtsasti alla ning on liikujad (ingl *movers*), kes ei lõpeta uute lahenduste katsetamist, eksperimenteerimist ja teevad tõrgetest ratsionaalseid järeldusi, mis teeb neist efektiivse programmeerija, kes jõuab lahenduseni. Kolmandaks programmeerija tüübiks on nokitsejad (ingl *tinkerers*), kes ei mõista kuigi hästi programmi töö kulgu ja teevad juhuslikke muudatusi, lootes, et programm hakkab tööle ning seetõttu võivad peatujatele sarnaselt mitte edasi areneda. Teades õpilase programmeerija tüüpi, on lihtsam anda isikustatud abi ja tagasisidet, mis õpilast aidata võib, jättes eemale üldistatud auditooriumile suunatud tagasiside.

Tartu Ülikooli pakutavad programmeerimisalased algkursused järgivad käesoleva töö autori hinnangul esimeses kolmes alamjaotuses väljatoodud võtet [6, 7, 9]. Neljas, programmeerija tüüpi käsitlev õpetamismeetod ei ole regulatiivselt sätestatud. Seda võtet on ka teistest võtetest ülikooli perspektiivis tunduvalt keerulisem rakendada, kuna nõuab individuaalse lähenemise tõttu rohkem ressursse. Madalamatel haridusastmetel, kus on tüüpiliselt väiksem osalejaskond ja individuaalne abi tõenäoliselt lihtsamini kättesaadav, tasub kaaluda lähenemisvõtte praktiseerimist.

Vihavainen jt [20] on empiirilise uuringu tarvis sarnase skemaatilise lähenemise välja töötanud, mida katsetati programmeerimise algkursuse tudengite peal. Töö käigus töötati välja eraldi "äärmuslik praktikameetod" (ingl *extreme apprenticeship*), mida autorid nimetavad Collins jt [8] väljatöötatud kognitiivse praktikameetodi (ingl *cognitive apprenticeship*) laienduseks. Meetod tugineb järgmistele printsiipidele: õppida tuleb tööd tehes, tagasiside peab olema pidev, oskusi tuleb siluda nii kaua, kuni need on selged ning "õpipoisist saab meister". Neid printsiipe toetatakse üleliigsel hulgal teoreetiliste teadmiste edasiandmise vältimisega, et oleks suurem rõhk õppimisel läbi töö tegemise; kursuse algusest peale programmeerimisega alustamisega, et vältida teooria õpetamist ilma praktikata; kindlate juhiste ja pideva instruktorite poolse abiga; väikese ulatusega eesmärkide sätestamisega, et vältida liigse informatsiooni andmist lühikese aja sees ning julgustamisega otsida iseseisvalt vajaminevat informatsiooni. Väljatöötatud praktikameetodi tulemusena oli nii algajatele kui ka edasijõudnutele suunatud programmeerimiskursuse edukalt läbinute protsent ainekursuste ajaloo kõrgeim, mil eelneval aastal oli algajate puhul edukalt läbinute osakaal 64% ja uuringu tegemise aastal 70% ning edasijõudnute puhul vastavalt eelneval aastal 60% ja uuringu tegemise aastal 84%.

Mõlemat kirjeldatud skemaatilist lähenemist silmas pidades võib täheldada teatud määral vasturääkivust. Robins jt [6] arvates tuleks teatud teemade puhul rohkem teema teoreetilise tausta õppimisele keskenduda, mil Vihavainen jt [20] leiavad, et teoreetilise teabe edasiandmist tuleks teatud määral piirata, et õpilane saaks õppida läbi programmi kirjutamise. Käesoleva töö autori arvamusel mõistavad usutavasti ka praktikameetodi välja töötanud autorid, et teatud teemade puhul tuleb olla printsiipides paindlikum, mille puhul soodustab tugev teoreetiline taust efektiivset praktiseerimist.

3.2 Esmase programmeerimiskeele valik programmeerimise õpetamisel

Esmase programmeerimiskeele valimine ei ole asjatundmatule lihtne protsess. Esmast programmeerimiskeelt valides võib valija kaaluda keele eesmärgipärasust, süntaktilist lihtsust ja selle asjakohasust aktuaalsel tööturul. Oluliseks võib ka pidada, kui hästi valitud keele abil programmeerimise universaalsed põhitõed selgeks õpitakse. Arvutiteaduse suunal õppima asuvate tudengite puhul võidakse esmase keele valik langetada vastava eriala õppekava poolt, kus võetakse arvesse erinevaid tegureid, näiteks programmeerimise põhitõdede selgeks õppimise kiirust.

Algajatele suunatud programmeerimiskursusi viiakse läbi paljudes programmeerimiskeeltes ja ei eksisteeri üht kokkulepitult standardiks seatud programmeerimiskeelt, mis algajatele programmeerijatele kõige sobilikum on. Üldteadaoleva algajatele suunatud programmeerimiskeele puudumist toetab kolme ülikooli esmakursuslaste ja valdavalt esimest korda programmeerimisega kokku puutuvate arvutiteaduste üliõpilaste seas läbi viidud uuring [21], kus leiti, et sõltumata programmeerimiskeele süntaktilistest erinevustest ja pealtnäha erinevast keerukusest, võrreldes C-keelt ja Pascalit, ei ole erinevustel programmeerimise põhitõdede selgeks õppimisega seoseid. Mõlema grupi puhul puudusid statistiliselt olulised erinevused süntaktika tundmaõppimisel ja probleemülesannete lahendamisel algoritmide abil. Seevastu on leitud erinevusi, kui võrrelda kahe tekstipõhise programmeerimiskeele asemel tekstipõhist ja graafilist programmeerimiskeelt. 118 erineva programmeerimise algkursusi pakkuva instituudi õpilaste [22] põhjal tehtud uuringus selgub, et kuue- kuni kümneaastaselt graafilise sisuga programmeerimiskeeli õppinutel sujuvad vastavasisulised õpingud hilisemas elus paremini, kui samas eas tekstilise sisuga programmeerimiskeeli õppinutel. See erinevus oli märgatav vaid konkreetses vanusevahemikus graafilise programmeerimiskeelega kokkupuutunud õpilaste puhul. Mõlema uuringu tulemustest kajastub kokkuvõtlikult, et võttes arvesse programmeerimisega esmakordselt kokkupuutuvaid üliõpilasi, ei mõjuta programmeerimise selgeks õppimist valitud programmeerimiskeel või -vahend. Edukust programmeerimise sujuval õppimisel võib määrata see, millises vanuses millise vahendi abil programmeerimise põhitõdesid õppida.

Vastukaaluks uuringutele, mis peavad programmeerimiskeele valikut esmasel kokkupuutumisel programmeerimisega vähetähtsaks, leiavad Wiedenbeck ja Ramalingam [23], et notatsioonilised erinevused programmeerimiskeelte vahel mõjutavad õpilaste arusaama programmi tööst, ehk seeläbi programmeerimise põhitõdede õppimist. Protseduuriliselt ja objektorienteeritult kirjutatud programmide töö tõlgendamisel ilmneb, et programmi andmevoo (ingl *data flow*) mõistmiseks on objektorienteeritud keeled algaja jaoks kasulikumad. Tasub mainida, et teatud programmeerimiskeelte puhul, näiteks Java ja Python, mis võimaldavad nii protseduurilise kui ka objektorienteeritud lähenemisega programme kirjutada, on üleliigne väljatoodud moel notatsioonilistest erinevustest tingitud õppimisefektiivsust võrrelda. Deklaratiivse programmeerimiskeele Prolog õppijate peal läbiviidud uuringus [24], kus analüüsiti kolme tüübi: protseduurilise, andmevooga seonduva ja funktsionaalse sisuga programmeerimisalaste küsimuste vastuseid ja vastava programmilõigu jaoks kirjutatud lühikokkuvõtteid, leiti, et vastuste täpsus ja lühikokkuvõtete pädevus erines COBOL ja Fortran keelega väljatoodud tüüpide lõikes. Sealjuures Prologi õppijate pädevus protseduuriliste ülesannete puhul oli kõrgem kui teist tüüpi ülesannete puhul. Protseduurilise programmeerimise ülekaalu (ingl *procedural predominance*) on täheldatud ka varasemates uuringutes [25, 26].

Käesolev bakalaureusetöö tugineb ainekursusele, kus programmeerimist õpetatakse Pythoni programmeerimiskeeles, mistõttu on oluline uurida Pythonit esmase programmeerimiskeele vaatepunktist. Pythoni programmeerimiskeel leiab laialdast kasutust paljudel algajatele suunatud programmeerimiskursustel mitmetel põhjustel. Seda peetakse algajasõbralikuks süntaktilise lihtsuse tõttu, mis võimaldab uute kontseptsioonidega tutvuval õppuril keskenduda rohkem kontseptsiooni sisule kui selle programmaatilisele realiseerimisele [27, 28]. Python on lisaks süntaktilisele lihtsusele TIOBE indeksi, tuntud programmeerimiskeelte ülemaailmse populaarsuse indikaatori alusel 2021. aasta oktoobrikuus kõige populaarsem programmeerimiskeel [29], mis printsiibis tähendab seda, et leidub palju õpetusi ja probleemipüstitusi, mida on õppuril internetist lihtne leida ja oma ülesannetes vastavalt rakendada. Tööhõivelisest vaatepunktist on andmeteaduse ja masinõppe oskustele rõhuvate töökohtade arv kiires kasvutrendis [30] ja peamiselt kasutusel olevad programmeerimiskeeled selles valdkonnas on Python ja R [31]. Põhitõdede selgeksõppimist soodustavat süntaktilist lihtsust, populaarsusest tulenevat suurt kasutajate ja õpetuste baasi ja nõudlust tööturul arvesse võttes on Pythoni programmeerimiskeel esmaseks õppimiseks ratsionaalne valik.

Pythoni programmeerimiskeele rakendust algajatele suunatud programmeerimiskursusel on uuritud võrdluses C programmeerimiskeelega. Kuigi nimetatud uuringus ei erinenud eksamitele mitteilmunud tudengite osakaal programmeerimiskeelte vahelises võrdluses, on erinevused üldise läbikukkunute osakaalude, kodutööde kordusesitamiste arvude ja keskmiste lõpphinnete puhul osutavad Pythoni esmaseks programmeerimiskeeleks valimise kasuks [32]. Watsoni ja Li [33] läbiviidud algajatele suunatud 161 programmeerimiskursuse õppetulemuste analüüsis ilmnes samuti, et Pythonit õppinute seas on keskmine läbikukkunute osakaal väiksem, kui Fortrani, Java, C++ ja C programmeerimiskeelte puhul. Pythoni abil programmeerima õpetamisel leiduvad ka kitsaskohad. Oldham [34] leiab, et õpilased, kes puutuvad programmeerimisega esmakordselt kokku läbi Pythoni õppimise, ei oma vastava kursuse lõppedes piisavalt head ülevaadet muutujatüüpidest, mis võib esmasel õppimisel tuua kaasa hooletusvead. Kuna Pythonis ei ole vaja muutujat deklareerides ette anda selle muutuja tüüpi, siis võib muutuja tüüp programmi töö käigus muutuda ja soodustada ebaoptimaalsete ning vigaste programmide kirjutamist. Oldhami väitel ei anna Pythonis õpetamine õpilasele õiget arusaama programmi kompileerimise osas. Käesoleva töö autori hinnangul on programmi kompileerimise mõistmine ja muutujatüübi selge deklareerimine sissejuhataval programmeerimiskursusel vähemtähtsad teiste programmeerimise fundamentaalsete teadmiste kõrval ning neid võiks õpetada edasijõudnutele mõeldud kursustel, kui õppijas on esialgsed arusaamad ja huvi tekitatud.

3.3 Alternatiivsed meetodid programmeerimise õpetamiseks

Programmeerimist ja selle põhitõdesid saab õpetada ka alternatiivsete meetodite abil. Tihti sisaldavad sellised alternatiivsed meetodid teatud määral mängulisuse elemente, tekitades õppijal tunde, et ta tegeleb peamiselt meelelahutusega, kuid tegelikkuses õpib uusi teadmisi. Pex4Fun on loodud eesmärgiga õpetada programmeerimist läbi mängimise ja seda on kasutatud mitmete ülikoolide pakutavate algajatele suunatud programmeerimiskursuste raames. Kuigi Pex4Fun kasutamine eeldab klassikalist tekstipõhist programmeerimist, on veebiliidesesse loodud graafika ja elulise sisuga ülesanded, mille lahendamine ei hõlma ainuüksi tavapärast tekstiredaktoris töötamist, vaid loovad keskkonna, kus õppimine toimub läbi mängu [35]. Reduct on loodud sarnaselt Pex4Fun platvormile põhimõttega õpetada programmeerimist läbi mängimise, kus erinevalt Pex4Fun platvormist tekstiliselt programmi koodi sisestama ei pea, vaid peab tõstma valmiskirjutatud koodiplokid ja muud graafilised

elemendid õigetesse kohtadesse [36]. Reductis tõstetavad koodiplokid imiteerivad JavaScripti programmeerimiskeelt ja platvormi osas tehtud uuringu alusel on leitud, et algajate programmeerijate puhul eksisteerib mängu käigus õpitu ülekanne klassikalisesse tekstipõhisesse JavaScripti programmeerimisse, kuid ainult Reducti abil õppinutel võib esineda suuremal hulgal süntaktilisi vigu [37], mis on käesoleva töö autori hinnangul loogiline, sest süntaksi õppimine tuleb suuresti läbi koodi kirjutamise, mida Reducti puhul ei esine. Veelgi mängulisem ja konkreetsest programmeerimiskeelest sõltumatu veebitarkvara Program Your Robot võimaldab õpetada tudengeid läbi mängu, kus tuleb abistada programmeeritav robot ühest punktist teise. Kasutada tuleb tsükliplokke ja tingimuslausete plokke [38]. Käesoleva töö autori hinnangul on mäng programmeerimise põhitõdede õpetamise poolest teatud määral primitiivne, sest keerulisemaid kontseptsioone (nt järjend, rekursioon) nimetatud mäng ei käsitle. Mäng on hea vahend paarisprogrammeerimiseks, mida sellisel kujul ka käesoleva töö autor esimesel programmeerimiskursusel rakendada sai. Lisaks mängimisele on programmeerimist võimalik õpetada ka läbi programmi töö või andmestruktuuride visualiseerimise, mida kasutatakse eeskätt keerulisemate kontseptsioonide lahtiseletamisel [39, 40].

3.4 Programmeerimisülesannete koostamine algajatele

Programmeerimist õppides toimub suurem osa efektiivsest õppimisest üldjuhul individuaalsel tasandil, mil õpilane tutvub materjalis leiduvate kontseptsioonidega ja rakendab neid probleemülesannete lahendamisel. Kirschneri [41] arvates peaksid individuaalset tööd nõudvate õppeprogrammide aluseks olema kaks peamist eeldust. Esiteks peaksid individuaalsed õppeprogrammid panema õpilasi lahendama „autentseid" probleeme või omandama keerukaid teadmisi teaberikkas keskkonnas, lähtudes eeldusest, et kui õppijad ise lahendusi koostavad, saavutatakse kõige efektiivsem õpikogemus. Teiseks eeldatakse, et teadmisi saab kõige paremini omandada läbi distsipiliinirohkete kogemuste. Kirschner vaidleb individuaalsele ja minimaalse juhendamisega õppimisele suunatud õppeprogrammide kasulikkuse vastu, mis ei tohiks ka käesoleva töö autori hinnangul olla ainus rakendatud õpetamisviis, kuid igas õppeprogrammis peab olema individuaalset õppimist, mistõttu võib toodud eeldusi positiivses kontekstis arvesse võtta.

Märkimisväärne osa programmeerimise individuaalsest õppimisest koosneb ülesannete lahendamisest, mille tulemusena õpitu kinnistub. Seetõttu peavad lisaks teoreetilisele õppematerjalile olema kaasahaaravad ka ülesanded, mis on selgesõnalise kirjeldusega õpilastele lahendamiseks püstitatud. Stevenson & Wagner [42] on erinevatele uuringutele tuginedes välja töötanud heade programmeerimisülesannete koostamise kriteeriumid. Autorid leiavad, et ülesanded peaksid tuginema mõnel elulisel probleemil ning ootama õpilaselt sellele probleemile ka elulist lahendust. Õpilastel tuleks ülesannetes lubada keskenduda mingile kindlale õpitavale teemale, mitte üritada haarata erinevaid uusi teemasid, kuid ülesanded peaksid sellegipoolest olema õpilastele väljakutseid pakkuvad ja huvitavad. Vältimaks ainult õpetatavate programmeerimise põhitõdede kasutamist ülesannetes võiks olla õpilastel võimalus ka kasutada erinevaid API-sid (ingl *application programming interface*) programmi kirjutamise hõlbustamise tööriistadena. Lisaks tuleks rõhuda loovusele ja uute lähenemiste katsetamisele, mida API-de rakendamine toestab.

Wolfe [7] arutleb ülesannete elulise sisu juures inimfaktori kaasamise olulisuse üle ülesannete tekstides. 81 õpilasega läbiviidud uuringus selgus, et õpilased tajusid huvitavamana ülesandeid, mis rõhusid sõnastuses selgelt elulisusele ja sidusid ülesande käigus loodavat programmi selle lõppkasutajaga. Wolfe sõnul on võimalik anda ülesande tekstiga selle tegelikku eesmärki paremini edasi, kui kirjeldada nõudeid kasutaja pilgu läbi. Selle asemel, et kirjeldada, mida peaks programm tegema, kui klaviatuuril vajutatakse tühikut, oleks parem rõhutada, kuidas programm käitub, kui kasutaja vajutab tühikut. Wolfe lisab, et kui on ülesande koostamisel teada, kes on programmi lõppkasutaja, võib kasutada ka sõna “kasutaja” asemel nt “õpetaja” või “arst”. Wolfe rõhutab, et sõnastuse poolest ei tundu erinevus suur olevat, kuid pika õpingute perioodi vältel aitab selline kirjeldusviis mõista õppijal inimese ja arvuti vahelist suhtlust, mis edendab paljuski õpilase motivatsiooni tehnilises valdkonnas töötada.

Praktilise ja elulise sisuga ülesannete väljatöötamist peavad oluliseks ka Robins jt [6], kes lisaks leiavad, et programmeerimist õpetades peaks materjali keerukus ajas kasvama, alustades lihtsatest teemadest ja liikudes süstemaatiliselt keerukamate teemade juurde, kus lihtsate teemade tundmine toestab keerulisemate teemade selgeksõppimist. Materjali keerukust võib siinkohal samastada ka ülesandepüstituse keerukusega. On võimalik koostada lihtsaid tsükli kasutamist nõudvaid ülesandeid ja on võimalik koostada keerukaid tsükli kasutamist nõudvaid ülesandeid. Oluline on siinkohal järgida, et esimeste teemade juures ei püstitataks liiga keerulisi ülesandeid, et õppijal oleks keeruline lahendust välja mõelda. Keerulisemateks

teemadeks on 167 Java programmeerimiskeelt õppiva õpilase peal läbi viidud uuringu järgi järjendid, meetodite deklareerimine, objektorienteeritud programmeerimise kontseptsioonid ja programmi ning objektide disain [43]. Paljud õpilased näevad ka testimist keeruka teemana. Käesolevas bakalaureusetöös väljatöötatav ülesannete kogu keskendub eelnimetatud teemadest järjenditele ja meetodite deklareerimisele. Morgani ja Butleri sõnul tuleks keerukamate teemade puhul pöörata rohkem tähelepanu vahetule tagasisidele, et mõista, mis elemendid teemades täpsemalt õppijatele keeruliseks osutuvad. Ülesannete tajutavat keerukust võib mõjutada ka õppurite enesekindlus arvutite käsitlemisel [44], mistõttu võib olla ka mõistlik õppekavasse integreerida sissejuhatavaid ainekursusi, mis keskenduvad tegemistele arvutis, mis võivad olla paljuski seotud programmeerimiskeele redaktori ülesseadmise ja erinevate n-ö tarkvaraarenduse operatsioonide (ingl *dev ops*) kontseptsioonidega.

Programmeerimisülesannete või üldisemalt programmeerimise õppematerjali koostamisel esineb tihti probleem, kus materjali koostaja pädevus teemades võib tekitada olukorra, kus teatud õpitavad teemad on lakooniliselt kirjeldatud või üldsegi välja jäetud. Üks viis selle probleemi ennetamiseks või olukorra esinemise parandamiseks on ülesandeid piloteerida - meetod, mida kasutatakse ka käesolevas bakalaureusetöös. Lisaks on loodud erinevaid vahendeid mõistmaks, missugustele teemadele materjali koostamisel tuleks rohkem tähelepanu pöörata. Üheks selliseks vahendiks on Leppänen jt [45] välja töötatud JavaScripti programmeerimiskeelel tuginev veebikomponent nimega Pheromones, mille abil uuriti, mis teemad veebipõhisest materjalikogumikust on õppuritel veebilehitsejas kõige rohkem avatud, kui nad programmeerimist õpivad.

Kokkuvõtlikult võiksid programmeerimiskursuse algul olla probleemülesanded piisavalt lihtsad ja ühele teemale orienteeritud, et õppija suudaks peas välja mõelda nendele ülesannetele lahenduskäigud ja seejärel need programmina kirja panna. Kursuse arenedes ja keerukamate teemade juurde jõudes tuleks probleemülesannete näol varem õpitud teemasid korrata, luues õpilaste jaoks ülesanded, mis nõuavad varasemate ülesannete jaoks õpitud ja kasutatud programmeerimisvõtteid, et toimuks õppimine läbi kordamise. Sellisel kujul ongi võimalik ülesannete keerukust ajas ka tõsta, sest kasvab programmeerimise põhitõdede hulk, mida ülesande lahendamisel vaja läheb. Ülesannete keerukuse kasvamisel tuleb olla tähelepanelik, et ei suureneks ülesannete sisu abstraktsus, vaid vastupidiselt säiliks ülesannete praktilisus ja elulisus. Praktilised ja elulised ülesanded motiveerivad õpilasi neid lahendama ja läbi lahendamise õpib õpilane programmeerima. Lisaks on soovituslik lasta õpilastel

eksperimenteerida väljatöötatud teekidega, et õpetada juba olemasoleva koodi taaskasutamist ja lubada programmeerimisalasel loovusel areneda. Väljatoodud lähenemisi rakendatakse ka käesoleva töö raames koostatud algajatele suunatud programmeerimisülesannete kogu väljatöötamisel.

4. Metoodika

Käesoleva bakalaureusetöö raames töötati välja ülesannete kogu (Lisa I) Tartu Ülikooli kursuse "Programmeerimise alused" jaoks. Protsess ülesannete kogu väljatöötamisel jaotus kaheks - ülesannete väljatöötamine ja osade väljatöötatud ülesannete piloteerimine 2022. aasta kevadel "Programmeerimise aluste" kursusel.

4.1 Programmeerimisülesannete väljatöötamine

Kursuse "Programmeerimise alused" jaoks töötati välja kuue õppenädala teemade alusel kokku 33 ülesannet, mis jaotuvad kodu- ja praktikumiülesanneteks. Kursuse esimese nädala jaoks töötati välja ainult praktikumiülesanded, kuid 2.-6. nädala jaoks töötati välja 3 koduülesannet ja 3 praktikumiülesannet iga nädala kohta. Paralleelselt ülesannete väljatöötamisega leidis aset pidev tagasisidevestlus käesoleva bakalaureusetöö juhendajaga, kes tegi ettepanekuid nii sõnastuse kui ka ülesannete sisulise poole parendamiseks.

Ülesannete väljatöötamisel lähtuti varem kasutusel olnud struktuurist. Iga ülesande alguses on sissejuhatav ja probleemi sisu lahtiseletav tekst. Sissejuhatavale tekstile järgneb täpne juhend, milline programm tuleb koostada: millist sisendit programm küsib ja kuidas käitub programm pärast sisendi andmist. Ülesannete puhul, mis puudutavad funktsioone, järgneb sissejuhatavale tekstile algul funktsiooni koostamise juhend ja seejärel selle programmi koostamise juhend, mis koostatud funktsiooni peab rakendama. Õpilaste jaoks keerukamate või senitundmatute lähenemiste puhul järgneb tavapäraselt programmi koostamise juhendile vihje või näpunäide. Ülesande lõpus on sõltuvalt ülesande tüübist kas näide või mitu näidet erinevate sisenditega, illustreerimaks, milline peaks loodav programm tervikuna välja nägema (Joonis 1).

Ürituste korraldaja rollis tuleb tihti ette olukordi, kus mingisuguse mängu või esinemise jaoks on vaja luua järjekord. Ausaim viis on järjekord moodustada juhuslikult. Lihtsustame ürituste korraldaja elu programmi abil, mis võtab sisendiks osalejate nimed ning loob neist juhusliku järjekorra. Koostada programm, mis

- küsib kasutajalt osalejate nimed komadega eraldatult ühe sõnena, nt "Karl,Peeter,Madis,Taavi,Ilmar" ja loob nimedest järjendi;
- väljastab juhusliku indeksi alusel ükshaaval järjendist nimesid, lisades ette järjekorranumbri, kuni järjendis rohkem nimesid ei ole.

Sisestatud sõne on lihtne teha sobivaks järjendiks käsu sõne.split(",") abil ja järjendi elementi muutujale omistada. Elemente saab järjendist eemaldada käsu list.pop() abil.

Näide programmi võimalikust tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl4.4b.py'
Sisesta nimed: Karl,Peeter,Madis,Taavi,Ilmar
1. Peeter
2. Taavi
3. Madis
4. Karl
5. Ilmar
```

Joonis 1. Loodud ülesande näide: Ülesanne "4.4b Järjekorra looja"

Ülesannete väljatöötamisel keskenduti suuresti ülesannete sisu elulisusele ja praktilisusele, sest programmeerimisülesannete elulisus ja praktilisus mõjutavad varasemate uuringute [6, 7, 42] kohaselt positiivselt õpilaste motivatsiooni programmeerimise õppimise osas. Eesmärgiks oli ainekursuse teemade arenedes progressiivselt keerulisemaid ülesandeid koostada, mis sisaldavad endas nii varasemaid kui ka parasjagu õpitavaid teemasid, sest on leitud, et ülesannete keerukuse progressiivne kasv soodustab efektiivset õppimist [6]. Ülesannete tekstid said koostatud hoolikalt õiget sõnastust valides, et vältida mitmetimõistetavust nõutava programmi töökäigu puhul.

Ühe õppenädala jaoks programmeerimisülesannete loomisel lähtuti tööprotsessis paljuski sarnase teemaga bakalaureusetöös rakendatud lähenemisest [46], tutvudes algselt läbitavate teemadega ja olemasolevate ülesannetega. Seejärel mõeldi hüpoteetilisele probleemile, kus vastava teema tundmine võib tulla kasuks lahenduse leidmisel ja interneti otsingumootorite abil leiti ülesannetele detailsem kontekst. Ülesande üldise sisu väljamõtlemise järgselt kirjutati probleemi lahendamiseks valmis programm, mida täiendati vastavalt jooksvalt tekkinud ideedele. Programmi põhjal pandi kirja ülesande sissejuhatav tekst ja programmi töötamise tingimused ning lisati vihjeid ja näiteid programmi võimalikust väljundist.

4.2 Väljatöötatud ülesannete piloteerimine ja tagasiside

Väljatöötatud ülesandeid piloteeriti osaliselt 2022. aasta kevadel aset leidnud "Programmeerimise aluste" kursusel. Igal nädalal, v.a 1. ja 6. õppenädalal, piloteeriti kaht kodu- ja üht praktikumiülesannet. Esimesel õppenädalal piloteeriti kolme praktikumiülesannet ja kuuendal õppenädalal üht kodu- ja üht praktikumiülesannet (Tabel 1).

Tabel 1. Kursuse "Programmeerimise alused" uus ülesannete kogu

Teemad	Ülesanded (piloteeritud ülesanded on paksus kirjas ja tärn tähistab lisaülesannet huvilistele)
1. Andmetüübid; muutujad; muutujale väärtuse andmine; arvulise väärtuste omistamine muutujale; tekstilise väärtuste omistamine muutujale, sõne; sisend kasutajalt; tõeväärtuse omistamine muutujale	**1.3 Ringi ümbermõõdu arvutamine** **1.4 Sammulugeja** **1.5 Kergejõustiku kahevõistlus**
2. Sõnede võrdlemine tingimuses; arvude võrdlemine; tingimuslause tingimuslause sees; keerulisemad tingimused; loogilised tehted ja avaldised; mitmeharuline tingimuslause elif; juhusliku arvu genereerimine; kilpkonnagraafika	**2.1 Parool** 2.2 Kolmnurkade liigitamine **2.3 Elektriauto** **2.4a Rannailm** 2.4b Kalkulaator 2.4c Jututoa kasutajanime generaator
3. Tsükkel; juhuslik arv	**3.1 Pooliku püramiidi süsteem** 3.2 Kohaloleku kontroll **3.3 Iga kolmas võidab** 3.4a Äraarvamismäng **3.4b Investeerimine indeksfondidesse** 3.4c Sularahaautomaat 3.4d Ringmajandus*
4. Järjend; järjendi operatsioonid, indeksid; järjendi viilutamine; for-tsükkel; funktsioon *range*; andmed failist	**4.1 Töögraafik** **4.2 Palindroomid** 4.3 Andmete korrastamine **4.4a Kampaaniahinnad** 4.4b Järjekorra looja 4.4c Lotomäng
5. Funktsioon; funktsiooni defineerimine; funktsiooni kirjeldus; käsureaaken; funktsiooni üldistamine, argumendid; funktsioon tagastab väärtuse; väärtuse tagastamine ja väljastamine; mitu funktsiooni koos; lokaalsed ja globaalsed	**5.1 Täishäälikute tuvastaja** 5.2 Ruutvõrrandi lahendaja **5.3 Nimede järjestaja** **5.4a Tüdrukute nimed** 5.4b Peastarvutamine aja peale 5.4c Õigekirja korrigeerija

muutujad; graafika ja funktsioon	
6. Lugemine veebist; lugemine failist; faili kirjutamine; lihtne kasutajaliides EasyGuiga	6.1 Peoliste nimekiri **6.2 Logi** 6.3 Arvkoodi genereerija **6.4a Vigane pangaautomaat** 6.4b Linna tuli rekkatäis.. 6.4c CAPTCHA

Tudengitelt küsiti piloteerimise vältel nii piloteeritavate ülesannete kui ka varasemast ülesannete kogust võetud ülesannete kohta iga kahe nädala tagant tagasisidet. Tagasisidevormis (Lisa II) paluti tudengitel hinnata lahendatud ülesannete keerukust, huvitavust ja tekstide selgust. Tudengid said jätta keerukuse, huvitavuse ja tekstide selguse hinnangud skaalal 1-5 või valida variandi "Ei lahendanud seda ülesannet". Võimalusel said tudengid jätta iga hinnangu järel kommentaare ja tagasisidevormi lõpus jätta vabas vormis ka üldisemat tagasisidet ülesannete sisu, ülesehituse ja ülesannetes käsitletavate teemade kohta.

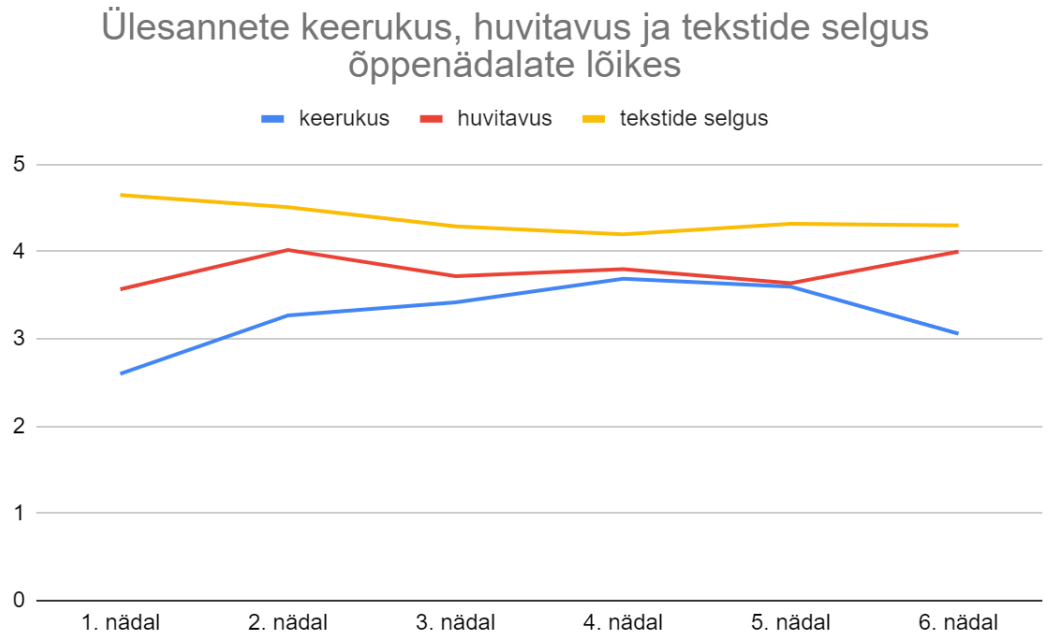
Tagasisidest saadud arvulisi vastuseid analüüsiti kasutades Pythoni programmeerimiskeelt ja seda toetavat veebipõhist interaktiivset arvutusplatvormi Jupyter Notebook. Vastuste analüüsimisel rakendati peamiselt keskmiste otsimist, et mõista ülesannete keerukuse, huvitavuse ja tekstide selguse üldist tajutavat hinnangut kursuse vältel ja võrrelda eelnimetatud tunnuseid varasemate ja käesoleva töö käigus väljatöötatud ülesannete puhul. Analüüsis ei võetud vastava ülesande keskmiste üldhinnangus arvesse tudengite vastuseid, kes vastasid ülesande kohta "Ei lahendanud seda ülesannet". Kõik tagasisidevormid olid tudengitele avatud kaks nädalat ja motiveerimaks tudengeid vormidele vastama, jagati vastamise eest ainekursusel boonuspunkte. Tagasisidevormide vastustest ilmnesid mõningad murekohad ülesannete tekstide ja ülesehituse osas, mille puhul viis käesoleva töö autor sisse vastavasisulised parandused. Parandused kajastuvad ka järgneval semestril loetava ainekursuse ülesannetes.

5. Tulemused ja arutelu

Kokku võttis 2022. kevadsemestril kursusest “Programmeerimise alused” osa 154 tudengit, kellel paluti täita kolm tagasisidevormi: 1.-2. nädala ülesannete, 3.-4. nädala ülesannete ja 5.-6. nädala ülesannete tagasisidevormid. Esimesele tagasisidevormile vastas 64, teisele 49 ja kolmandale 38 tudengit. Individuaalseid vastajaid oli kolme tagasisidevormi peale kokku 89, kellest 20 vastasid kolmele, 22 kahele ja 27 ühele tagasisidevormile. Käesoleva töö eesmärgiks oli välja töötada ülesannete kogu, kus teemade arenedes ülesannete keerukus kasvab, kuid säilib ülesannete elulisus ja sellest paljuski sõltuv huvitavus. Seetõttu oli käesoleva töö autori hinnangul oluline uurida eeskätt kolmele ja kahele tagasisidele vastuse andnud tudengite ülesannete suhtes tajutava keerukuse ja huvitavuse suhet ajas, sealjuures ka tekstide selgesõnalisuse säilimist või muutumist teemade arenedes. Töö autori hinnangul oli samuti oluline võrrelda piloteeritavaid ülesandeid kolme kategooria - huvitavuse, keerukuse ja tekstide selgesõnalisuse suhtes varasemate ülesannetega tuvastamaks loodud ülesannete puhul potentsiaalseid murekohti. Vabas vormis jäetud kommentaaride puhul otsustati teha ülevaated üldise meelestatuse osas igal õppenädalal.

5.1 Kolmele ja kahele tagasisidevormile vastanute tajutava ülesannete keerukuse, huvitavuse ja tekstide selguse analüüs

Joonisel 2 kujutatud graafiku hinnangulised keskmised arvutati järgmiselt: leiti kolmele (N=20) ja kahele tagasisidevormile vastanud tudengite (N=22) keskmised tajutavad ülesannete keerukuse, huvitavuse ja tekstide selguse hinnangud ning nädalate kaupa leiti nende tudengite (N=42) hinnangute keskmiste üldkeskmised. Näiteks: kui tudengi tajutav 1. nädala ülesannete keerukus oli 2,78 ja teise tudengi tajutav 1. nädala ülesannete keerukus oli 3,32, arvutati kahe tudengi keskmiste üldkeskmiseks 3,05. Joonisel 2 kujutatud graafiku koostamisel ei arvestatud kahele tagasisidevormile vastanud tudengi vastamata jäetud nädalate puhul tema vastavate nädalate hinnangulist keskmist. Töö autor pidas oluliseks uurida nii kolmel kui ka kahel korral tagasisidevormile vastanute tudengite hinnanguid koos detailsema üldpildi saamiseks.



Joonis 2. Kolmele ja kahele tagasisidevormile vastanute vastuste hinnangulised keskmised õppenädalate lõikes

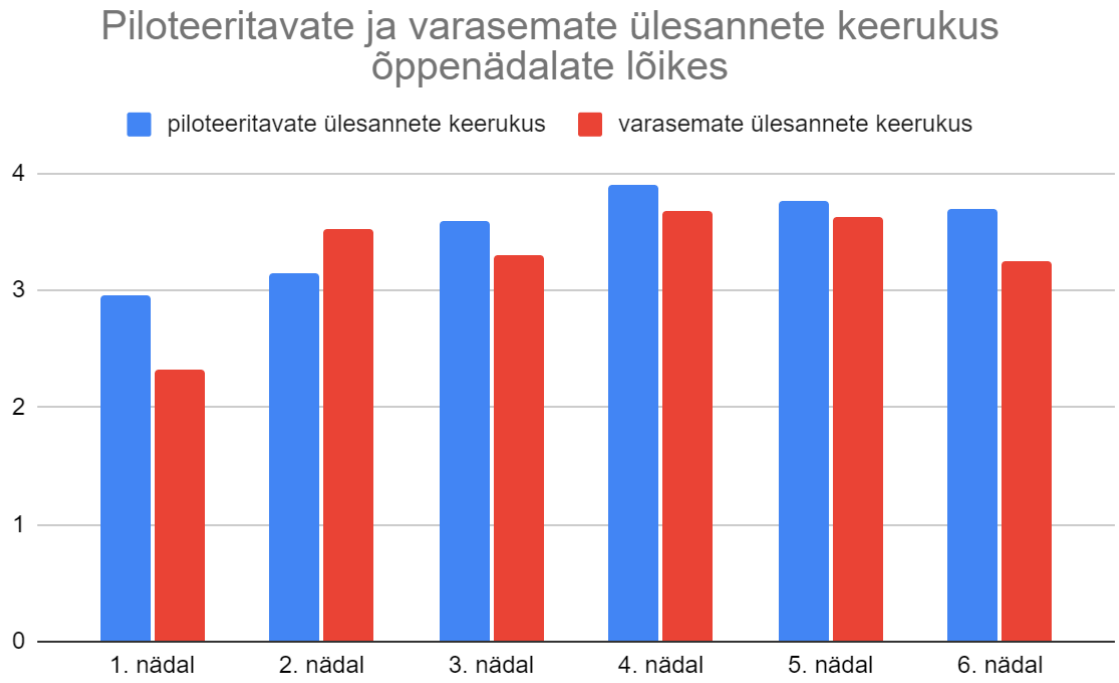
Kolmele ja kahele tagasisidevormile vastanute hinnangute järgi leidis tajutavas tekstide selguses kursuse vältel aset järkjärguline langus. Autori hinnangul muutusid hilisemates teemades ülesannete tekstid koos funktsiooni ja/või programmi töö kirjeldusega pikemaks ja võisid seetõttu ka tunduda tervikuna keerulisemalt hoomatavad. Keskmiselt hinnati ülesannete tekstide selgust kuue õppenädala lõikes hindega 4,38 (maksimumist 5), mille alusel võib hinnata, et tekstid olid tudengite jaoks selgelt kirjutatud.

Ülesannete hinnangulise huvitavuse ja keerukuse puhul on märgata teisest õppenädalast alates negatiivset korrelatsiooni: ülesannete hinnangulise keerukuse kasvades ülesannete hinnanguline huvitavus langeb. Ülesannete keerukuse progressiivne kasv oli ülesannete kogu väljatöötamisel taotluslik, sest sellisel lähenemisel on leitud positiivseid mõjusid programmeerimise õppimise efektiivsusele [6]. Ülesannete huvitavuse langus teisest õppenädalast võib olla põhjustatud pikematest ja keerukamatest ülesannetest, mis võivad lahendamiseks rohkem aega nõuda ja seeläbi võivad vähendada ülesande huvitavust. Teisalt võisid kursuse vältel ülesanded teatud määral korduvaks muutuda, seeläbi ka vähem huvitavaks muutuda, sest ülesannetel on läbivalt üsna sarnane struktuur: kasutaja peab programmi saama midagi sisestada ja programm peab midagi sisendi alusel kalkuleerima või muul moel modifitseerima ja seejärel ekraanile väljastama. Viimast oletust toetab ülesannete tajutava huvitavuse tõus kuuendal õppenädalal, sealjuures ka tajutava keerukuse langus, kui tudengid

said katsetada graafilist kasutajaliidest, lahendades sealjuures varasemast erineva struktuuriga ja erinevat lähenemisvõtet nõudvat ülesannet ja saades võrreldes harjumuspärase tulemuse ekraanile väljastamisega lahenduseks midagi kasutajasõbralikku ja intuitiivset - programmi, millega arvutikasutajatena tuttavamad ollakse. Keskmiselt hinnati kolmele ja kahele tagasisidevormile vastanute poolt ülesannete keerukust kuue õppenädala lõikes hindega 3,27 ja ülesannete huvitavust hindega 3,79.

5.2 Piloteeritud ülesannete tajutava keerukuse, huvitavuse ja tekstide selguse võrdlus varasemate ülesannetega

Ainekursuse vältel hinnati tudengite (N=89) poolt piloteeritavaid ülesandeid keskmiselt varem kasutusel olnud ülesannetest keerukamaks, mil ainult teisel õppenädalal olid tudengite hinnangul varasemad ülesanded keerulisemad (Joonis 3).



Joonis 3. Piloteeritavate ja varasemate ülesannete keerukused õppenädalate lõikes

Keerukuse tulemuste paremaks mõistmiseks tuleb autori hinnangul seda hüpoteetilisest aspektist lähemalt uurida. Koduülesannete puhul oli iga tudengi jaoks tingimus, et kodutöö maksimaalsete punktide kättesaamiseks tuleb lahendada kõik ülesanded. Praktikumiülesannete puhul piisab maksimaalsete punktide saamiseks ühe ülesande lahendamisest õppenädalas. Nii

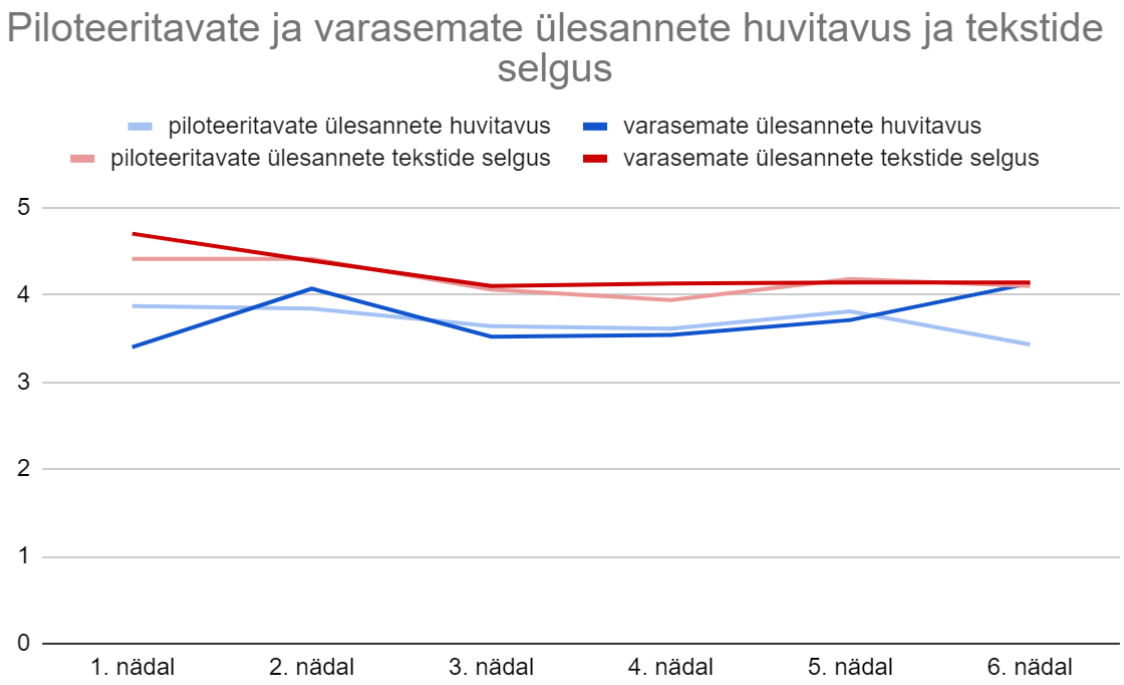
kodu- kui praktikumiülesannetel on astmeline keerukus ka ühe õppenädala sees, st esimene ülesanne on lihtsam kui teine ja teine ülesanne on lihtsam kui kolmas. Autori loodud ülesannetest olid piloteeritavad praktikumiülesanded vastavate õppenädalate esimesed ja teised praktikumiülesanded, ehk astmelise raskuse loogika järgi nädala lihtsamad ja keskmise raskusastmega praktikumiülesanded. Leitud tulemustest tingituna tekkis eraldiseisev hüpotees, millele vastuse leidmine võib selgitada keerukuse hinnangute erinevust: Tudengid, kes lahendavad õppenädalas vaid ühe praktikumiülesande, tajuvad seda konkreetset üht ülesannet keerukamana kui tudengid, kes lahendavad mitu praktikumiülesannet.

Tabel 2. Praktikumiülesannete lahendanute osakaalud ja praktikumiülesannete tajutavad keerukused õppenädalate lõikes

Õppenädalad	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**
Vaid autori loodud praktikumiülesande lahendanute osakaal	0%	20.3%	34.7%	4%	0%	21%
Kõigi praktikumiülesannete lahendanute tajutav keerukus (skaalal 1-5)	2.63	3.55	3.48	4	3.67	3.83
Vähemalt ühe praktikumiülesande lahendanu tajutav keerukus (skaalal 1-5)	2.44	3.44	3.6	3.95	3.8	3.54
Autori loodud koduülesannete tajutav keerukus (skaalal 1-5)	-	3.15	3.57	3.77	3.82	3.39
Varasemate koduülesannete tajutav keerukus (skaalal 1-5)	-	3.21	2.82	3.51	2.94	3.11

Madal keskmine vaid autori loodud praktikumiülesande lahendanute osakaalu ja väikesed erinevused tajutava keerukuse puhul kõigi praktikumiülesannete lahendanute ja vähemalt ühe praktikumiülesande lahendanute vahel kummutavad seatud hüpoteesi (Tabel 2). Kuigi õpilaste motivatsioon lahendada igal õppenädalal kõik koduülesanded võib olla hindamissüsteemist tulenevalt kõrgem (227 koduülesannetele keerukuse hinnangu andnud 238-st vastanust lahendas kõiki ülesandeid), ja hüpoteesi kohaselt võiks see vähendada tajutavat keerukust, leidub märgatav erinevus tajutava keerukuse puhul ka autori loodud ja varasemate koduülesannete vahel. Võib järeldada, et autori loodud kodu- ja praktikumiülesanded osutusid keerukamaks kui varasemad ülesanded.

Piloteeritavate ja varasemate ülesannete huvitavus oli tudengite hinnangul õppenädalate lõikes keskmiselt võrdsel tasemel (Joonis 4), mil 1. nädalal hinnati piloteeritavate ülesannete huvitavust kõrgemalt ja 6. nädalal varasemate ülesannete huvitavust kõrgemalt. Ülesannete tekstide selgus vähenes õppenädalate lõikes sarnaselt piloteeritavate ja ka varasemate ülesannete puhul (Joonis 4).



Joonis 4. Piloteeritavate ja varasemate ülesannete huvitavus ja tekstide selgus õppenädalate lõikes

5.3 Parandusettepanekud ülesannetele ja tudengite meelestatus ülesannete suhtes

Tudengitel ja 2022. aasta kevadsemestril kursust "Programmeerimise alused" läbiviivatel praktikumijuhendajatel oli võimalus jätta ka kirjalikku tagasisidet. Sõltuvalt kirjalikust tagasisidest viidi ülesannete kogus sisse parandusi. Ülesande "1.5 Kergejõustiku kahevõistlus" puhul leiti, et puudub täpsustus, kuidas tulemused täisarvudeks teisendatakse. Lisaks ei olnud märgitud, et tähised A, B ja C on konstandid. Täpsustus lisati programmi koostamise nõuetesse ja konstantide tabeli ette kirjutati selgitus tähiste kohta. Ülesandes "4.1 Töögraafik" oli kirjeldatud, et näitefailis on 31 rida (jaanuaris on 31 päeva), kuid failis oli 30 rida. Faili sai lisatud üks rida juurde. Ülesandes "5.1 Täishäälikute tuvastaja" parandati funktsiooni kirjelduses sõna "väljastab" sõnaga "tagastab". Ülesandest "Ruutvõrrandi lahendamine" sai

ülesanne 5.2. Lisaks tagastab selles ülesandes funktsioon lahendid järjendina, mitte ennikuna. Ülesannet “5.1 Täishäälikute tuvastaja” muudeti selliselt, et programm otsib ainult täishäälikute arvu (varem oli nõue, et programm peab otsima ka kaashäälikute arvu), eemaldati while-tsükli kasutamine ja järjendi abil kontrollimise osas jäeti vihje. Ülesannet “5.3 Nimede järjestaja” muudeti selliselt, et loodud funktsiooni kasutatakse kahel korral. Ülesande “5.4a Tüdrukute nimed” puhul tehti ettepanek väljastada nimed üksteise alla, mitte komaga eraldatult, sest viimasel juhul tuleks ka tudengitele selgitada, kuidas viimase nime puhul koma vältida. Ülesande näidet muudeti selliselt, et nimed väljastatakse üksteise alla. Ülesande “6.2 Logi” puhul kirjutati soovituse alusel käsklused väljastatava küsilause sisse sulgudesse. Lisaks sisulistele ettepanekutele tehti ka palju muudatusettepanekuid kirjastiili osas, mis suuremas osas rakendust leidsid.

Tudengite kirjalikus vormis jäetud hinnangud ülesannete sisu ja struktuuri kohta olid valdavalt positiivse meelestatusega - kiideti ülesannete teksti ja selle huvitavust kuue õppenädala lõikes. Mõned tudengid leidsid, et ülesanded on liiga keerulised, mil mõned tudengid leidsid, et ülesanded on lihtsad. Kohati leiti, et ülesannetel oleks võinud rohkem vihjeid olla. Avaldati soovi rohkem kasutajaliidesele suunatud ülesandeid lahendada - neid ülesandeid peeti huvitavaks. Tagasisidevormide osas esitati soovitus küsida tagasisidet tihedamini, mitte kahenädalaste vahedega, sest tudengitel võib minna meelest, kuidas nädal tagasi lahendatud ülesanded sisult olid.

6. Kokkuvõte

Bakalaureusetöö eesmärk oli luua uus ülesannete kogu Tartu Ülikoolis loetava ainekursuse "Programmeerimise alused" jaoks. Ülesannete loomise eel tutvuti töö teoreetilises osas programmeerimiskursusi ja programmeerimise õpetamist kajastava kirjandusega, et läheneda ülesannete loomisele teaduspõhise meelestatusega, mis tagaks võimalikult kvaliteetsed ülesanded. Lisaks tutvuti kursuse "Programmeerimise alused" varasemate aastate kodu- ja praktikumiülesannetega ning loeti läbi kõik kursuse materjalid, et olla kursis teemadega, mis kodu- ja praktikumiülesannetega tuli katta. Töö praktilises osas koostati 33 ülesandest (18 praktikumiülesannet, 15 koduülesannet) koosnev ülesannete kogu, mille valmimist toetas käesoleva töö juhendaja soovituste ja parandusettepanekutega. 17 loodud ülesannet piloteeriti 2022. aasta kevadsemestril. Parandusettepanekute ja soovitustega tegeleti ülesannete kogu loomise vältel ja pärast ülesannete piloteerimist.

Iga kahe õppenädala järel, kokku kolm korda, küsiti tudengite käest veebipõhise tagasisidevormi kaudu tagasisidet möödunud kahe nädala jooksul lahendatud kodu- ja praktikumiülesannete keerukuse, huvitavuse ja tekstide selguse kohta. Tagasisidet anti 151 korda, kellest 89 olid individuaalsed vastajad. Ülesannete hinnangulise huvitavuse ja keerukuse puhul oli märgata teisest õppenädalast alates negatiivset korrelatsiooni, mil ülesannete hinnangulise keerukuse kasvades ülesannete hinnanguline huvitavus langes. Ülesannete tekstide selgus langes tudengite hinnangul õppenädalate vältel, kuid hinne oli kuue õppenädala vältel 5-palli skaalal keskmiselt 4,38. Kodu- ja praktikumiülesannete keerukust hinnati kuue õppenädala lõikes keskmiselt hindega 3,27 ja ülesannete huvitavust hindega 3,79. Piloteeritavad ülesanded osutusid tudengitele varasemalt kasutusel olnud ja ka 2022. kevadsemestril kasutatud ülesannetest hinnanguliselt keerukamaks, mil ülesannete huvitavuse ja tekstide selguse puhul märkimisväärset erinevust ei olnud. Tudengid ja praktikumijuhendajad jäid nimetatud kevadsemestri kursuse vältel ülesannete koguga rahule.

Käesoleva tööga ei olda kindlad, kas 16 piloteerimata jäänud ülesannet vajavad sisuliselt muutmist ja millised oleksid tajutava keerukuse, huvitavuse ja tekstide selguse hinnangud käesoleva töö raames loodud ülesannete kogu tervikliku kasutamise puhul. Tudengite hinnangute tervikpildi saamiseks tuleks järgmisel semestril, mil "Programmeerimise aluste"

kursust antakse, küsida sarnast tagasisidet kõigi käesolevas bakalaureusetöös loodud ülesannete kohta.

7. Viidatud kirjandus

[1] Tartu Ülikool. Arvutiteaduse instituut. Programmeerimise alused õppeaastal 2020/21. https://courses.cs.ut.ee/2021/prog-alused/spring/Main/HomePage (05.02.2022)

[2] Tartu Ülikool. Arvutiteaduse instituut. Programmeerimise alused õppeaastal 2016/18. https://courses.cs.ut.ee/2017/prog-alused/spring (01.05.2022)

[3] Tartu Ülikool. Arvutiteaduse instituut. Programmeerimise alused õppeaastal 2017/18. https://courses.cs.ut.ee/2018/prog-alused/spring (01.05.2022)

[4] Tartu Ülikool. Arvutiteaduse instituut. Programmeerimise alused õppeaastal 2018/19. https://courses.cs.ut.ee/2019/prog-alused/spring (01.05.2022)

[5] Tartu Ülikool. Arvutiteaduse instituut. Programmeerimise alused õppeaastal 2019/20. https://courses.cs.ut.ee/2020/prog-alused/spring (01.05.2022)

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

[7] Wolfe, J. Why the rhetoric of CS programming assignments matters. *Computer Science Education*, 2004, *14*(2), 147-163.

[8] Collins, A., Brown, J. S., & Newman, S. E. Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. *Thinking: The journal of philosophy for children*, 1988, *8(1)*, 2-10.

[9] MOOC.fi. Python Programming MOOC 2021. https://programming-21.mooc.fi/ (25.10.2021)

[10] Python Principles. Python Programming Lessons. https://pythonprinciples.com/ (25.10.2021)

[11] Python Core. Sololearn. https://www.sololearn.com/learning/1073 (25.10.2021)

[12] edX. Introduction to Computer Science and Programming Using Python. https://www.edx.org/course/introduction-to-computer-science-and-programming-7 (08.11.2021)

[13] Michigan Online. Python for Everybody. https://online.umich.edu/series/python-for-everybody/ (08.11.2021)

[14] Tartu Ülikooli õppeinfosüsteem ÕIS. Programmeerimine (6 EAP). https://ois2.ut.ee/#/courses/LTAT.03.001/details (08.11.2021)

[15] Tartu Ülikooli õppeinfosüsteem ÕIS. Programmeerimise alused (3 EAP). https://ois2.ut.ee/#/courses/MTAT.03.236/details (08.11.2021)

[16] Tartu Ülikool. Arvutiteaduse instituut. Programmeerimise alused II. https://courses.cs.ut.ee/2021/progalused2/spring (08.11.2021)

[17] Macsuga-Gage, A. S., Simonsen, B., & Briere, D. E. Effective Teaching Practices: Effective Teaching Practices that Promote a Positive Classroom Environment. *Beyond Behavior*, 2012, 22(1), 14–22.

[18] Jenkins, T. Teaching programming-a journey from teacher to motivator. In *The 2nd Annual Conference of the LSTN Center for Information and Computer Science.* 2001.

[19] Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research,* 1986, *2*(1), 37–55.

[20] Vihavainen, A., Paksula, M., & Luukkainen, M. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, 2011, 93-98.

[21] Xinogalos, S., Pitner, T., Ivanovic, M. & Savic, M. Students' Perspective on the First Programming Language: C-Like or Pascal-Like Languages?. *Education and Information Technologies*, 2018, vol. 23, no. 1, 287–302

[22] Chen, C., Sonnert, G., Sadler, P., Haduong, P. & Brennan, K. The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages, *Computer Science Education*, 2018, vol. 29, no. 1, 23–48.

[23] Wiedenbeck, S., & Ramalingam, V. Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies*, 1999, 51(1), 71-87.

[24] Good J., Brna P., Cox, R. Novices and program comprehension: Does language make a difference? *Proceedings of 19th Annual Conference of the Cognitive Science Society.* Stanford University, 1997, 936-937.

[25] Corritore, C., Wiedenbeck, S. What do novices learn during program comprehension? *International Journal of Human-Computer Interaction,* 1991, *3*(2), 199-222

[26] Pennington, N. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology,* 1987, *19*, 295-341

[27] Pellet, J. P., Dame, A., & Parriaux, G. How beginner-friendly is a programming language? A short analysis based on Java and Python examples. 2019.

[28] Kruglyk, V., & Lvov, M. Choosing the first educational programming language. In ICT in Education, Research and Industrial Applications: Integration, Harmonization and

Knowledge Transfer: Proceedings of the 8th International Conference ICTERI, 2012 188-189.

[29] TIOBE. TIOBE Index for October 2021. https://www.tiobe.com/tiobe-index/ (25.10.2021)

[30] Press, G. Salaries And Job Opportunities For Data Scientists Continue To Rise. *Forbes*, 2021. https://www.forbes.com/sites/gilpress/2021/06/27/salaries-and-job-opportunities-for-data-scientists-continue-to-rise/?sh=7c9545da4276 (25.10.2021)

[31] EdX Blog. 9 Top Programming Languages for Data Science. https://blog.edx.org/9-top-programming-languages-for-data-science (25.10.2021)

[32] Wainer, J., & Xavier, E. C. A controlled experiment on Python vs C for an Introductory Programming course: students' outcomes. *ACM Transactions on Computing Education (TOCE)*, 2018, *18*(3), 1-16.

[33] Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, 2014, 39-44.

[34] Oldham, J. D. What happens after Python in CS1?. *Journal of computing sciences in colleges*, 2005, *20*(6), 7-13.

[35] Tillmann, N., De Halleux, J., Xie, T., Gulwani, S., & Bishop, J. Teaching and learning programming and software engineering via interactive gaming. In *2013 35th International Conference on Software Engineering (ICSE)*, 2013, 1117-1126.

[36] Reduct Redux: A Game to Teach Coding Comprehension. https://reduct-game.github.io/play/ (27.12.2021)

[37] Arawjo, I., Wang, C. Y., Myers, A. C., Andersen, E., & Guimbretière, F. Teaching programming with gamified semantics. In *Proceedings of the 2017 CHI conference on human factors in computing systems* 2017, 4911-4923.

[38] Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science*, 2012, *9*, 522-531.

[39] Baldwin, L. P., & Kuljis, J. Learning programming using program visualization techniques. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 2001, 8.

[40] Rudder, A., Bernard, M., & Mohammed, S. Teaching programming using visualization. In *Proceedings of the Sixth IASTED International Conference on Web-Based Education* 2007, 487-492.

[41] Kirschner, P. A., Sweller, J. & Clark, R. E. “Why Minimal Guidance During Instruction Does not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching.” *Educational Psychologist*, 2006, *41 (2)*: 75–86.

[42] Stevenson, D. E., & Wagner, P. J. Developing real-world programming assignments for CS1. *ACM SIGCSE Bulletin*, 2006, *38*(3), 158-162.

[43] 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.

[44] 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.

[45] Leppanen, L., Leinonen, J., Ihantola, P., & Hellas, A. Using and collecting fine-grained usage data to improve online learning materials. *In 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* 2017, 4-12. IEEE.

[46] Puusepp, S. Programmeerimisülesannete kogu gümnaasiumi valikkursusele „Tarkvaraarendus“. Tartu Ülikool, Arvutiteaduse instituut, 2019.

Lisad

I. Ülesannete kogu

Praktikumiülesanded - 1. nädal

Ülesanne 1.3. Ringi ümbermõõdu arvutamine

Koostada programm, mis

- küsib kasutajalt ringi raadiuse (ujukomaarv) ja väljastab ringi ümbermõõdu (2 * pii * raadius).

Näiteks kui kasutaja sisestab raadiuseks 4, siis väljastab programm talle 2 * 3.14 * 4 = 25.12. Pii tegelik väärtus ei ole päris täpselt 3.14, kuid kokkuleppeliselt on ta kahe komakohani ümardades sellise väärtusega.

Näide programmi tööst:

```
>>> %Run 'yl1.3.py'
Sisesta ringi raadius: 4
Ringi ümbermõõt on: 25.12
```

Ülesanne 1.4. Sammulugeja

Kui on teada sammu pikkus ja läbitav vahemaa, siis on võimalik jagamistehtega arvutada, mitu sammu vahemaa läbimiseks on tarvis teha. Näiteks kui on teada, et sammupikkus on 0.8 meetrit ja läbitav vahemaa on 10000 meetrit, siis on tehtavate sammude arv 10000 / 0.8 = 12500.

Koostada programm, mis

- küsib kasutajalt sammu pikkuse (meetrites, ujukomaarv);
- küsib kasutajalt läbitava vahemaa (meetrites, täisarv);
- väljastab ekraanile tehtavate sammude arvu, mis on ümardatud täisarvuni.

Täisarvuks ümardamisel saab kasutada funktsiooni round, näiteks round(sammude_arv) väärtuseks on 14194, kui kasutaja on sisestanud sammu pikkuseks 0.93 ja läbitavaks vahemaaks 13200. Funktsioon int ei ole siin sobiv, kuna ümardab arvu alla, kuigi sellisel juhul jääks meil pool sammu puudu.

Näide programmi tööst:

```
>>> %Run 'yl1.4.py'
Sisesta sammu pikkus: 0.85
Sisesta läbitav vahemaa: 12355
Selle vahemaa läbimiseks on tarvis teha 14535 sammu.
```

Ülesanne 1.5. Kergejõustiku kahevõistlus

Tegelikkuses kergejõustikus kahevõistlust ei eksisteeri ja kahevõistluseks nimetatakse talispordiala, mis koosneb suusahüpetest ja murdmaasuusatamisest. Loeme siinkohal kergejõustiku kahevõistluseks ala, mis koosneb klassikalise kümnevõistluse kahest alast: 100 meetri jooksust ja kaugushüppest. Kümnevõistluse aladel on omamoodi punktisüsteem, mida on üsna keeruline pähe õppida, kui tahta näiteks sõpradega staadionil mõõtu võttes punktid välja arvutada. Valemeid rakendades, eriti programmi näol, mis arvutuse sinu eest ära teeb, muutub asi märksa lihtsamaks.

Kümnevõistluse alade punkte arvutatakse kahe erineva valemi alusel:

- Ajalist sooritust nõudvad alad (100 meetri jooks): int(A(B — P)^C)
- Distantsilist sooritust nõudvad alad (kaugushüpe): int(A(P — B)^C)

Valemeid vaadates tekib ilmselt mõte, et mida need tähed täpselt tähistavad. Selleks on loodud kümnevõistluse punktide arvutamise tabel, mida on 1984. aastast sellisel kujul üleilmselt kasutatud. **P** tähistab valemites võistleja sooritust. 100 meetri jooksu puhul on see sekundites, kaugushüppe puhul sentimeetrites. **A**, **B** ja **C** on konstandid, mis tuleb vastava ala puhul tabelist leida.

Spordiala	A	B	C
100 meetri jooks	25.4347	18	1.81
Kaugushüpe	0.14354	220	1.4

Koostada programm, mis

- küsib kasutajalt 100 meetri jooksu tulemust (ujukomaarv, sekundites). Arvestame sellega, et jooksualadel võetakse aega kahe komakoha täpsusega;
- küsib kasutajalt kaugushüppe tulemust (täisarv, sentimeetrites);
- arvutab mõlemalt alalt saadud punktid (pidada meeles, et punktid on täisarvulised. Punkte saab teisendada täisarvulisteks funktsiooniga `int`), liidab need kokku ja väljastab kasutajale.

Näide programmi tööst:

```
>>> %Run 'yl1.5.py'
Sisesta 100 meetri jooksu tulemus: 9.88
Sisesta kaugushüppe tulemus: 843
Kahevõistlusest koguti 2298 punkti.
```

Koduülesanded - 2. nädal

2.1. Parool

Üldiselt on kõigil tõsiseltvõetavatel veebilehtedel, kus kasutaja saab end süsteemi sisse logida, parooli seadistamisel omad tingimused, mis peavad olema täidetud. Kõige tavalisemaks tingimuseks on minimaalne tähemärkide arv, millest parool koosneb. Tähemärkide arvu kontrollimiseks on võimalik koostada lihtne programm - sarnane sellele, mida veebilehed ka kasutavad!

Koostada programm, mis

- küsib kasutajalt parooli (sõne);
- väljastab kasutajale “Parool sobib.”, kui parooli pikkus on vähemalt 8 tähemärki, vastasel juhul väljastab “Parool ei sobi.”

Näited programmi tööst:

```
>>> %Run 'yl2.1.py'
Sisesta parool: pikkparool
Parool sobib.

Sisesta parool: lühike
Parool ei sobi.
```

Vihje: Sõne pikkuse leidmisel on abiks funktsioon `len`, nt muutuja ‘parool’ korral pikkus = len(parool)

2.2. Kolmnurkade liigitamine

Kolmnurki saab külgede pikkuste alusel liigitada võrdkülgseteks, võrdhaarseteks ja erikülgseteks kolmnurkadeks. Liigitamise lihtsustamiseks saame koostada programmi.

Koostada programm, mis

- küsib kasutajalt kolmes jaos külgede pikkused (ujukomaarvud);
- väljastab kasutajale
    - “Võrdkülgne kolmnurk”, kui kõik küljed on ühepikkused;

- “Võrdhaarne kolmnurk”, kui kaks külge on ühepikkused;
- “Erikülgne kolmnurk”, kui kõik küljed on erineva pikkusega.

Näited programmi tööst:

```
>>> %Run 'yl2.2.py'
Sisesta kolmnurga külgede pikkused:
x: 2.2
y: 2.2
z: 5
Võrdhaarne kolmnurk

>>> %Run 'yl2.2.py'
Sisesta kolmnurga külgede pikkused:
x: 5
y: 3
z: 4
Erikülgne kolmnurk
```

2.3. Elektriauto

Jan soovib sõita oma uue elektriautoga Tallinnast Tartusse. Autosse istudes näeb ta armatuuril arvu, mis näitab, mitu kilomeetrit tal veel järgmise laadimiseni sõita on. Jani huvitab, kas tal on võimalik näidatud arvu alusel Tallinnast muretult Tartusse sõita. Koostame ta aitamiseks programmi. Teame akude kohta seda, et külma ilma puhul võib aku kestvus osutuda oodatust lühemaks ja ilm on pahatihti meteoroloogiavõhikute jaoks üsna juhuslik nähtus, mistõttu peame selle juhuslikkuse aku kestvusesse sisse arvestama.

Koostada programm, mis

- küsib kasutajalt läbitava vahemaa (ujukomaarv);
- leiab juhusliku täisarvu vahemikus 0 kuni 1. 0 tähistab siinkohal, et väljas ei ole külm. 1 tähistab siinkohal, et väljas on külm. Juhul kui väljas on külm, on tegelik näit 80% esialgsest arvust, ehk sisestatud kaugusest tuleb arvestada ainult 80%. Uuest väljaarvutatud kaugusest tuleb kasutajale märku anda.
- saadud kauguse alusel väljastab ühe alljärgnevatest vastustest:
    - kui arv on suurem või võrdne 186-ga, siis väljastada “Jõuame kenasti Tartusse.”;
    - kui arv on väiksem kui 186 aga suurem või võrdne 120-ga, siis väljastada “Mäo ristis võiks peatuse teha ja pisut laadida.”;
    - kui arv on väiksem kui 120, siis väljastada “Võiksime laadida enne sõitma hakkamist.”

Näited programmi võimalikust tööst:

```
>>> %Run 'yl2.3.py'
```

```
Sisesta, mitu kilomeetrit aku järgi veel sõita saab: 186
Jõuame kenasti Tartusse.

>>> %Run 'yl2.3.py'
Sisesta, mitu kilomeetrit aku järgi veel sõita saab: 186
Väljas on külm, tegelikult saab veel 148.8 km sõita.
Mäo ristis võiks peatuse teha ja pisut laadida.

>>> %Run 'yl2.3.py'
Sisesta, mitu kilomeetrit aku järgi veel sõita saab: 113
Võiksime laadida enne sõitma hakkamist.
```

Praktikumiülesanded - 2. nädal

2.4a Rannailm

Randa külastatakse üldiselt siis, kui väljas on soe ja päike paistab. Langetamaks otsust, kas minna randa või mitte, saame koostada lihtsa abiprogrammi.

Koostada programm, mis

- küsib kasutajalt välistemperatuuri (täisarv);
- küsib kasutajalt, kas päike paistab (“jah” või “ei”);
- küsib kasutajalt, kas rannas lehvib roheline lipp (“jah” või “ei”);
- annab kasutajale teada, et täna võib randa minna, kui välistemperatuur on 20 kraadi või rohkem ja päike paistab **või** rannas lehvib roheline lipp. Randa minekuks piisab ainuüksi sellest, kui rannas lehvib roheline lipp, ehk sellisel juhul ei pea teised tingimused olema täidetud. Kui ei kehti esimene pool tingimustest (õige temperatuur ja päike) ja ei kehti teine pool tingimusest (roheline lipp), annab programm kasutajale teada, et täna ei ole rannailm.

Näited programmi tööst:

```
>>> %Run 'yl2.4a.py'
Mis temperatuur väljas on? 15
Kas päike paistab? jah
Kas rannas on roheline lipp? jah
Võid minna randa!

>>> %Run 'yl2.4a.py'
Mis temperatuur väljas on? 25
Kas päike paistab? jah
Kas rannas on roheline lipp? ei
Võid minna randa!
```

```
>>> %Run 'yl2.4a.py'
Mis temperatuur väljas on? 23
Kas päike paistab? ei
Kas rannas on roheline lipp? ei
Täna ei tasu randa minna.
```

2.4b Kalkulaator

Koostada programm, mis

- küsib kasutajalt esimese täisarvu;
- küsib kasutajalt ühe järgnevatest matemaatilistest operatsioonidest:
    - liitmine (+)
    - lahutamine (-)
    - korrutamine (*)
    - jagamine (/)
- küsib kasutajalt teise täisarvu;
- sõltuvalt valitud matemaatilisest operatsioonist liidab programm teisena sisestatud arvu esimesele, lahutab teisena sisestatud arvu esimesest, korrutab esimesena sisestatud arvu teisena sisestatud arvuga või jagab esimesena sisestatud arvu teisega, ja väljastab tulemuse ekraanile.

Pea siinkohal meeles matemaatilist reeglit, et nulliga ei ole võimalik jagada ja loo programmi selline tingimuslause, mis nulliga jagamisel kasutajale veast märku annab. Programmi koostamisel eeldame, et kasutaja sisestab ühe neljast eeltoodud matemaatilisest operatsioonist ja muu veahaldusega arvestama ei pea.

Näide programmi tööst:

```
>>> %Run 'yl2.4b.py'
Sisesta esimene arv: 10
Sisesta matemaatiline operatsioon: *
Sisesta teine arv: 4
Vastus on: 40

>>> %Run 'yl2.4b.py'
Sisesta esimene arv: 5
Sisesta matemaatiline operatsioon: /
Sisesta teine arv: 0
Nulliga ei saa jagada.
```

2.4c Jututoa kasutajanime generaator

Rein asub arendajana tööle üht pealinlastele ja ülikoolilinlastele suunatud virtuaalset jututuba haldava ettevõtte juurde. Kuna uutel kasutajatel on tihti probleemiks see, et nad ei tea, missugune kasutajanimi valida, siis leiab ülemus, et Rein võiks koostada programmi, mis genereerib kasutajatele kolme küsimuse põhjal kasutajanime, mida võib soovi korral kasutada.

Koostada programm, mis

- küsib kasutaja sugu (“mees” või “naine”);
- küsib kasutaja vanust (täisarv vahemikus 18-99);
- kui kasutaja sisestatud vastus on väiksem kui 18 või suurem kui 99, annab programm sellest märku ja lõpetab töö;
- küsib kasutaja elukohta (“Tartu” või “Tallinn”);
    - kui kasutaja sisestab midagi muud, annab programm märku, et ta sisestas midagi valesti, ning lõpetab töö.
- genereerib saadud vastuste põhjal kuuest tähemärgist koosneva kasutajanime, kus
    - 1. tähemärk on vastavalt “m”, kui kasutaja sisestas enda sooks “mees” või “n”, kui kasutaja sisestas enda sooks “naine”
    - 2. ja 3. tähemärk on kasutaja vanus (18-99)
    - viimased 3 tähemärki on vastavalt “trt”, kui kasutaja sisestas elukohaks “Tartu” või “tln”, kui kasutaja sisestas elukohaks “Tallinn”
- väljastab genereeritud kasutajanime ekraanile ja lõpetab töö.

Näited programmi tööst:

```
>>> %Run 'yl2.4c.py'
Kas Sa oled mees või naine? naine
Kui vana Sa oled? 130
Sisestasid midagi valesti. Proovi uuesti.

>>> %Run 'yl2.4c.py'
Kas Sa oled mees või naine? naine
Kui vana Sa oled? 42
Mis on sinu elukoht? Paide
Sisestasid midagi valesti. Proovi uuesti.

>>> %Run 'yl2.4c.py'
Kas Sa oled mees või naine? mees
Kui vana Sa oled? 13
Oled liiga noor, et jututoas osaleda.

>>> %Run 'yl2.4c.py'
Kas Sa oled mees või naine? mees
Kui vana Sa oled? 25
```

```
Mis on sinu elukoht? Tallinn
Sinu kasutajanimi võiks olla näiteks: m25tln
```

Eesmärk on programmi töö (ja vajadusel selle enneaegne lõpetamine) katta tingimuslausetega, kuid kui jääd hätta, on võimalik programmi töö lõpetada käsu `sys.exit(0)` abil. Selle kasutamiseks on vaja programmi algusesse `import sys` lisada.

NB! Pythonis saab sõnesid "ehitada" kasutades operaatorit "+", näiteks `täisnimi = eesnimi + " " + perenimi` ja juba eksisteerivale sõnele saab lisada lõppu uut sõne operaatori "+=" abil, näiteks kui sõne tähestik väärtus on "abcde", siis kasutades `tähestik += "f".` on sõne tähestik väärtus "abcdef".

Koduülesanded - 3. nädal

3.1 Pooliku püramiidi süsteem

Paljudes treeningkavades kasutatakse n-ö pooliku püramiidi süsteemi, kus iga seeria puhul tehakse üks kordus rohkem kui eelmises kuni kindlaksmääratud lõppseeriani. Tihti tekib aga selliste harjutuste puhul küsimus, kui palju terve harjutuse peale kordusi sooritatakse, sest hea on päevikusse kirja panna, kui palju mingil päeval mingit harjutust tehtud sai. Selle väljaselgitamiseks tuleb liita seeriad kokku. Näiteks kui treeningkava soovitab teha pooliku püramiidi süsteemis kätekõverdusi kuni 10 korduseni, siis kujuneb seeriate põhjal kogu kätekõverduste arv välja järgmiselt: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55 kordust. 10-se pooliku püramiidi puhul on arvutus veel võrdlemisi kiire, kuid suuremate lõppseeriate arvude juurde jõudes oleks mõistlik kirja panna lihtne programm, mis teeb töö kasutaja eest ära.

Koostada programm, mis

- küsib kasutajalt naturaalarvu (positiivne täisarv);
- arvutab pooliku püramiidi süsteemis naturaalarvude summa eelnevalt sisestatud naturaalarvuni (kaasa arvatud);
- väljastab saadud summa.

Näited programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.1.py'
Sisesta naturaalarv: 15
Naturaalarvude summa sisestatud arvuni: 120

Sisesta naturaalarv: 10
Naturaalarvude summa sisestatud arvuni: 55
```

Üldisemaks teadmiseks huvilistele: täieliku püramiidi süsteem arvuni 10 tähendab ronimist 1…10…1, ehk tippu jõudes vähenevad arvud tagasi arvuni 1. Summa arvutamisel saaksime siin vastuseks 100 (10 liidetakse vaid ühel korral).

3.2 Kohaloleku kontroll

Õpetaja kontrollib kohalolijate nimesid ja soovib need sisestada programmi, et need pärast üksteise all talle väljastataks.

Koostada programm, mis

- küsib kasutajalt sisendiks kas õpilase eesnime (igasugune sõne, väljaarvatud “lõpp”) või programmi töö lõpetamise märguande (“lõpp”);
- õpilase nime sisestamisel liidab kõikide õpilaste nimede ühissõnele nime koos reavahetuse käsuga, näiteks `koik_nimed += sisend + "\n"`;
- töö lõpetamisel käsuga “lõpp” väljastab tühirea ja seejärel kõik õpilase nimed ühe sõnena.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.2.py'
Sisesta õpilase eesnimi või lõpeta töö: Brandon
Sisesta õpilase eesnimi või lõpeta töö: Eero
Sisesta õpilase eesnimi või lõpeta töö: Tauno
Sisesta õpilase eesnimi või lõpeta töö: Margus
Sisesta õpilase eesnimi või lõpeta töö: Ott
Sisesta õpilase eesnimi või lõpeta töö: Kristi
Sisesta õpilase eesnimi või lõpeta töö: Helena
Sisesta õpilase eesnimi või lõpeta töö: Kert
Sisesta õpilase eesnimi või lõpeta töö: Annika
Sisesta õpilase eesnimi või lõpeta töö: lõpp

Brandon
Eero
Tauno
Margus
Ott
Kristi
Helena
Kert
Annika
```

3.3 Iga kolmas võidab

Ühel limonaadil oli pikalt kampaania, mille raames oli iga kolmanda pudeli korgi all tähis, mille saamisel oli pudeli omanikul võimalus tasuta uus pudel limonaadi saada. Eerole limonaad tohutult maitseb, seega otsustab ta õnne proovida, ostes teatud arv limonaade ja lootes sellega mingi hulga juurde võita.

Koostada programm, mis

- küsib kasutajalt ostetud limonaadide arvu (täisarv);
- otsustab iga ostu puhul juhusliku arvu alusel, kas limonaadipudel on võidutähisega - võimalus on üks kolmest;
- loendab võidutähisega pudeleid;
- väljastab iga ostu puhul teate “Osteti limonaad.”, kui osteti võidutähiseta limonaadipudel ja “Osteti limonaad, millega võideti uus limonaad!”, kui osteti võidutähisega limonaadipudel;
- ostude lõppedes väljastab, mitu limonaadi kokku võideti.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.3.py'
Sisesta, mitu limonaadi ostad: 10
Osteti limonaad.
Osteti limonaad, millega võideti uus limonaad!
Osteti limonaad.
Osteti limonaad.
Osteti limonaad, millega võideti uus limonaad!
Osteti limonaad, millega võideti uus limonaad!
Osteti limonaad.
Osteti limonaad.
Osteti limonaad.
Osteti limonaad.
Kokku võideti 3 limonaadi!
```

Praktikumiülesanded - 3. nädal

3.4a Äraarvamismäng

Koostada programm, mis

- leiab enne iga arvamist uue juhusliku arvu vahemikus [0,100];
- küsib kasutajalt, kas juhuslik arv on suurem kui 50 (sisestatakse “suurem”), väiksem kui 50 (sisestatakse “väiksem”) või võrdne arvuga 50 (sisestatakse “võrdne”);
- väljastab kasutajale vastuse “Õige!” kui pakutu on õige, vastasel juhul väljastab “Ei ole suurem” kui sisestati “suurem”, ”Ei ole väiksem”, kui sisestati “väiksem”, ”Ei ole võrdne”, kui sisestati “võrdne”;
- väljastab “Mängu lõpp.”, kui sisestatakse ükskõik mis sõne või arv, mis ei ole “suurem”, “väiksem” või “võrdne” ja programm lõpetab töö.

Näide programmi võimalikust tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.4a.py'
```

```
Kas arv on suurem, väiksem või võrdne 50-ga? suurem
Õige!
Kas arv on suurem, väiksem või võrdne 50-ga? väiksem
Ei ole väiksem.
Kas arv on suurem, väiksem või võrdne 50-ga? väiksem
Õige!
Kas arv on suurem, väiksem või võrdne 50-ga? võrdne
Ei ole võrdne.
Kas arv on suurem, väiksem või võrdne 50-ga? x
Mängu lõpp.
```

3.4b Investeerimine indeksfondidesse

Ilmar saab hästi tehtud töö eest lisatasu ja leiab, et tahaks paigutada seda indeksfondidesse, et oma vara väärtust tõstma hakata. Enne investeerima hakkamist sooviks Ilmar programmi, mis annaks talle mingisuguse arusaama teenitavast summast, kui tal on teada, kaua ta investeerida soovib ja milline on tema soovitud aastatootlus.

Koostada programm, mis

- küsib kasutajalt investeeritava summa (ujukomaarv);
- küsib kasutajalt investeeringu kogupikkuse (aastates, täisarv);
- küsib kasutajalt oodatava aastatootluse (protsentides, ujukomaarv);
- väljastab kasutajale tehtavast investeeringust saadava lõppsumma (ümardatuna sajandikeni).

Liitintressi arvutamisest võib mõelda nii, et igal aastal võetakse eelmisel aastal teenitud summa ja liidetakse sellele aastatootluse ning eelmisel aastal teenitud summa korrutis. Kui Ilmaril on tänaseks aasta möödas alginvesteeringust, milleks osutus 10000 eurot aastatootluse 10.25% juures, siis tema aastase teenistuse saab arvutada järgmiselt: teenistus = 10000 + 10000 * 0.1025 = 11025. Pärast teist aastat saab arvutada juba järgmiselt: teenistus = 11025 + 11025 * 0.1025 = 12155 jne. Siinkohal muutub tsükkel väga kasulikuks, sest saame aastaid silmas pidades arvutada nii pika plaaniga investeeringu, kui ise soovime.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.4b.py'
Sisesta summa, mida soovid investeerida: 10000
Sisesta aastates, kaua soovid investeerida: 5
Sisesta oodatav iga-aastane tootlus: 10.25
10000.0 eurost saab 10.25% tootluse juures 5 aastaga 16288.95 eurot.
```

3.4c Sularahaautomaat

Kuigi sularaha kasutamine on tänasel päeval jäänud tehnoloogiliste arengute tõttu pigem tagaplaanile, siis tihtilugu leiab laatadelt ja kodukohvikute päevadelt väikeettevõtteid, kes end pangaterminaliga varustanud ei ole. Lennart on suundumas kodukohvikute päevadele ja teab varasemast külastuskorrast, et ainult pangakaardiga võib jääda hätta, seega otsustab ta pisut raha automaadist välja võtta. Lennartil on sellisteks heaoluga seonduvateks toiminguteks eraldi pangakaart, kus tal on alati 100 kuni 200 eurot kontol. Simuleerime Lennarti käiku automaadi juures programmiga.

Koostada programm, mis

- leiab juhusliku täisarvu vahemikus [100,200] ja salvestab selle kontoseisuna;
- küsib kasutajalt: “Kas Sa soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?”. Raha väljavõtmiseks peab kasutaja sisestama “raha”, kontoseisu vaatamiseks “seis”, sularahaautomaadi kasutamise lõpetamiseks “lõpp”
- kui kasutaja vastab “raha”, küsib programm kasutajalt täisarvulise summa, mida soovitakse välja võtta. Kui summa on väiksem või võrdne kontoseisuga, arvutatakse summa kontoseisust maha ja väljastatakse kontoseis. Kui summa on suurem kui kontoseis, antakse kasutajale sellest märku. Mõlemal juhul küsib programm uuesti “Kas Sa soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?”;
- kui kasutaja vastab “seis”, väljastab programm kontoseisu ja küsib uuesti “Kas Sa soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?”;
- kui kasutaja vastab “lõpp”, väljastab programm “Sessioon on lõppenud.” ja programm lõpetab töö.

Näide programmi võimalikust tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.4c.py'
Kas soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?
seis
Kontol on praegu 112 €.

Kas soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?
raha
Sisesta summa, mida soovid välja võtta: 50
Võtsid 50 € välja, kontol alles 62 €.

Kas soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?
seis
Kontol on praegu 62 €.

Kas soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?
raha
Sisesta summa, mida soovid välja võtta: 72
```

```
Kontol ei ole piisavalt vahendeid.

Kas soovid raha välja võtta, vaadata kontoseisu või lõpetada sessioon?
lõpp
Sessioon on lõppenud.
```

3.4d Ringmajandus* (tärnülesanne huvilistele)

Karl armastab metsikult mullivett ja ostab seda koju iga kuu lõpututes kogustes. Ühel hetkel leiab ta, et oleks mõistlik igakuistel sissetulekutel ja väljaminekutel silma peal hoida ja hea viis selleks on koostada eelarve, kust ei saa kindlasti puududa mullivee kuluartikkel. Karl aga teab, et kui tal trenni peale kulub 35 eurot kuus, siis ta saabki selle eest terve kuu trennis käia, samas mulliveega on asi pisut keerulisem - tuleb arvestada ka taaraga, mille loodusest hooliv inimene kenasti tagasi viib. Ei ole mõistlik arvestada 1 euro maksva mullivee jaoks 100 eurot, kui plaan on juua 100 pudelit vett, sest iga pudeli pealt saab 10 senti tagasi. Tsüklilise arvutuskäiguga mõistame, et 100 euro eest saab tegelikult 111 pudelit mullivett. Selline tsükliline arvutus on aga käsitsi tehes võrdlemisi tüütu, mistõttu aitame Karli siinkohal programmiga, mis aitab tal välja arvutada, kui palju mullivett ta tegelikult planeeritud rahasumma eest osta saab.

Koostada programm, mis

- küsib rahasumma, mida soovitakse vee peale kulutada (ujukomaarv);
- küsib ühe pudeli hinna (ujukomaarv);
- väljastab kasutajale teate "Liiga vähe raha, et üht pudelit osta.", kui rahasumma on väiksem kui ühe pudeli hind;
- arvutab tsükliliselt igal poeskäigul saadud pudelite koguse, kus võtab järgmiseks võimalikuks ostutsükliks arvesse:
    - ostetud pudelite eest saadud taara (iga pudeli eest saab 0.1 eurot tagasi);
    - allesjäänud raha (kui ostjal on 1.5 eurot ja pudel maksab 0.8, siis jääb alles 0.7 eurot);
- iga ostutsükli lõpus väljastab, mitmes tsükkel on teostatud, mitu pudelit vett on kokku ostetud ja palju raha tsüklist alles jäi (taararaha + allesjäänud raha, ümardatud sajandikeni);
- juhul kui allesjäänud ja taararaha summa võimaldab uut ostutsüklit, viib läbi uue ostutsükli, arvestades eeltoodud punktidega;
- lõpetab töö, kui raha ei võimalda enam uut ostutsüklit läbi viia, väljastades mitu pudelit vett antud rahasumma eest osteti ja palju raha alles jäi.

Tegemist on ülesandega, mis võib tunduda keerukas, kuid sellele tasub läheneda protseduuriliselt, mõeldes välja üks lihtne näide ja tehes arvutuskäigud paberil või mõnes tekstiredaktoris läbi. Kui suudad sellist protseduuri läbi viia paberil, suudad selle seni õpitud teadmistega ka programmijupina kirja panna!

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl3.4d.py'
Sisesta pudelivee eelarve:  127
Sisesta ühe veepudeli hind:  0.88

1. tsükkel
Kokku on meil 144 pudelit ostetud.
Raha (taararaha + jääk) jäi meil alles 14.68 eurot.

2. tsükkel
Kokku on meil 160 pudelit ostetud.
Raha (taararaha + jääk) jäi meil alles 2.2 eurot.

3. tsükkel
Kokku on meil 162 pudelit ostetud.
Raha (taararaha + jääk) jäi meil alles 0.64 eurot.

Poeskäigud on lõppenud, saime 127.0 euro eest 162 pudelit vett ja raha
jäi alles 0.64 eurot.
```

Koduülesanded - 4. nädal

4.1. Töögraafik

Väikeettevõte on määranud jaanuarikuu igaks päevaks ühe inimese tööle. Töötajad on kirjas tekstifailis töögraafik.txt, kus esimesel real on märgitud töötaja, kes 1. jaanuaril töötab, teisel real on märgitud töötaja, kes 2. jaanuaril töötab, ja nii kuni 31. reani välja.

Koostada programm, mis

- loeb failist töögraafik.txt sisse töötajate nimed ja sisestab need järjendisse;
- küsib kasutajalt täisarvu, mis tähistab mitmenda kuupäeva töötaja nime soovitakse teada (tasub meeles pidada, et järjendis on esimesel kohal element indeksiga 0);
- väljastab töötaja nime, kes nimetatud kuupäeval töötab.

Andmete sisse lugemiseks ja järjendisse lisamiseks saab järgida toodud koodijuppi:

```
fail = open("töögraafik.txt", encoding="UTF-8")
andmed = []
for rida in fail:
    andmed.append(rida)
fail.close()
```

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl4.1.py'
```

```
Mitmenda kuupäeva töötaja nime soovid teada: 6
6. kuupäeval töötab Madis
```

4.2. Palindroomid

Palindroom on keelend, mis on nii päri- kui ka tagurpidi lugedes täpselt samasugune. Lühikeste sõnade puhul nagu näiteks “aga” on lihtne mõista, et tegemist on palindroomiga, kuid pikemate sõnade puhul on palindroomi tuvastamine keerulisem.

Koostada programm, mis

- loob tühja järjendi, kuhu hakkab sisestama otsuseid, kas sõna on palindroom või mitte;
- küsib kasutajalt *while*-tsükli abil sõnu seni, kuni kasutaja sisestab sõna “lõpp”;
- iga sõna puhul, mis ei ole sõna “lõpp”, uurib, kas sõna on palindroom. Sõna on lihtne tagurpidi pöörata käsu `sõna[::-1]` abil;
- juhul, kui sõna on palindroom, lisab sõna loodud järjendisse nt sõna “aga” puhul kujul “Aga on palindroom.”, muutes sisestatud sõna esitähe suureks;
- juhul, kui sõna ei ole palindroom, lisab sõna loodud järjendisse nt sõna “koer” puhul kujul “Koer ei ole palindroom.”, muutes sisestatud sõna esitähe suureks;
- juhul, kui sõna on “lõpp”, väljastab tühja rea ja seejärel kõik järjendi elemendid üksteise alla ning lõpetab töö.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl4.2.py'
Sisesta sõna: aga
Sisesta sõna: koer
Sisesta sõna: sadam
Sisesta sõna: sadas
Sisesta sõna: puuluup
Sisesta sõna: oo
Sisesta sõna: sõna
Sisesta sõna: kuulilennuteetunneliluuk
Sisesta sõna: lõpp

Aga on palindroom.
Koer ei ole palindroom.
Sadam ei ole palindroom.
Sadas on palindroom.
Puuluup on palindroom.
Oo on palindroom.
Sõna ei ole palindroom.
Kuulilennuteetunneliluuk on palindroom.
```

4.3. Andmete korrastamine

Loomaarst märkis päeva jooksul visiidil käinud ja kaalutud koerad üles ühte tekstifaili (koerad.txt), kuid tegi seda pisut ebaorganiseeritult. Ta esialgne plaan oli kirjutada esimesele reale koera nimi ja tõug ning teisele reale koera kaal, ehk kõik koerad järgemööda üksteise alla, kuid päeva lõpus märkas ta, et vahepeal kirjutas hoopis kaalu enne nime ja tõugu.

Organiseeritud faili sisu ilmestab järgmine järjestus:

```
Bessy, Tiibeti spanjel
2,3
Pimmu, labrador
32,4
Götze, Pembroke'i Walesi korgi
12,0
Muki, Dalmaatsia koer
18,0
Tommi, kuldne retriiver
27,0
```

Ebaorganiseeritud faili sisu ilmestab järgmine järjestus (paksus kirjas on read, kus kaal on kirjas enne nime ja tõugu):

```
2,3
Bessy, Tiibeti spanjel
Pimmu, labrador
32,4
12,0
Götze, Pembroke'i Walesi korgi
Muki, Dalmaatsia koer
18,0
27,0
Tommi, kuldne retriiver
```

Faili sisu tuleks programmi abil sisse lugeda, organiseerida ja korrektselt väljastada.

Koostada programm, mis

- loob kaks tühja järjendit (näiteks koerad ja kaalud);
- küsib kasutajalt failinime;
- loeb sisestatud failist sisse kõik read, sisestab koera nime ja tõugu sisaldava rea järjendisse ühte järjendisse ja koera kaalu sisaldava rea järjendisse teise järjendisse. Koera nime ja tõugu on lihtne eristada kaalust selle poolest, et vastav rida algab suure tähega;
- väljastab *for*-tsükli ja funktsiooni *range*() abil koerad kaaludega kujul `koerad[i] + ", kaal: " + kaalud[i]` - saame võtta ühe järjendi (vabal valikul) pikkuse käsuga *len*(), näiteks `len(koerad)` ja kasutada saadud pikkust tsükli teostamiseks.

Andmete sisselugemiseks võib aluseks võtta esimeses ülesandes toodud koodijupi. Küll aga tuleb tähele panna, et toodud koodijupp on kõigest mall ja selle ülesande puhul tuleb tingimuslauseid rakendades faili sisu eri järjenditesse paigutada.

Seda, kas rea esimene täht on suurtäht, saame kontrollida käsuga `rida[0].isupper()`. Mäletame loengust, et indeksil 0 asub sõne puhul selle esimene sümbol ja käsuga `.isupper()` saame kontrollida, kas see sümbol on suurtäht või mitte. Siinkohal ei saa me kasutada `rida.isupper()` indeksi järgi sümbolit otsimata, kuna sellisel juhul kontrollitakse kõiki sõne sümboleid.

Näide programmi tööst:

```
>>> %Run 'yl4.3.py'
Bessy, Tiibeti spanjel, kaal: 2,3
Pimmu, labrador, kaal: 32,4
Götze, Pembroke'i Walesi korgi, kaal: 12,0
Muki, Dalmaatsia koer, kaal: 18,0
Tommi, kuldne retriiver, kaal: 27,0
```

Praktikumiülesanded - 4. nädal

4.4a Kampaaniahinnad

On kaks järjendit: üks koosneb täisarvudest 100,45,25,37,19 ja teine koosneb ujukomaarvudest 0.8, 0.3, 0.2, 0.5, 0.95. Igale elemendile esimeses järjendis, mis kujutab alghindasid, vastab üks element teises järjendis, mis kujutab kampaania hinnakordajaid. Ostule summaga 100 vastab kordaja 0.8, ostule summaga 45 vastab kordaja 0.3 jne.

Koostada programm, mis *for*-tsüklit kasutades

- arvutab iga alghinna puhul kampaaniahinna (ujukomaarv, näiteks: 100 * 0.8 = 80);
- väljastab toote järjekorranumbri, alghinna ja kampaaniahinna kujul `i. toote alghind: x, kampaaniahind: y`, kus i tähistab järjekorranumbrit, x alghinda ja y kampaaniahinda;
- liidab kõik saadud kampaaniahinnad kokku ja väljastab tulemuse ekraanile.

Näide programmi tööst:

```
>>> %Run 'yl4.4a.py'
1. toote alghind: 100, kampaaniahind: 80.0
2. toote alghind: 45, kampaaniahind: 13.5
3. toote alghind: 25, kampaaniahind: 5.0
4. toote alghind: 37, kampaaniahind: 18.5
```

```
5. toote alghind: 19, kampaaniahind: 18.05
Toodete kogumaksumus on 135.05
```

4.4b Järjekorra looja

Ürituste korraldaja rollis tuleb tihti ette olukordi, kus mingisuguse mängu või esinemise jaoks on vaja luua järjekord. Ausaim viis on järjekord moodustada juhuslikult. Lihtsustame ürituste korraldaja elu programmi abil, mis võtab sisendiks osalejate nimed ning loob neist juhusliku järjekorra.

Koostada programm, mis

- küsib kasutajalt osalejate nimed komadega eraldatult ühe sõnena, nt "Karl,Peeter,Madis,Taavi,Ilmar" ja loob nimedest järjendi;
- väljastab juhusliku indeksi alusel ükshaaval järjendist nimesid, lisades ette järjekorranumbri, kuni järjendis rohkem nimesid ei ole.

Sisestatud sõne on lihtne teha sobivaks järjendiks käsu `sõne.split(",")` abil ja järjendi elementi muutujale omistada. Elemente saab järjendist eemaldada käsu `list.pop()` abil.

Näide programmi võimalikust tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl4.4b.py'
Sisesta nimed: Karl,Peeter,Madis,Taavi,Ilmar
1. Peeter
2. Taavi
3. Madis
4. Karl
5. Ilmar
```

4.4c Lotomäng

Lotomängija leiab end olukorrast, kus tal puudub kirjutusvahend, et arvudele pihta saamisel neile riste peale tõmmata. Küll aga on tal võimalik kasutada arvutit, et võimalusel selle abil arvudel mugavalt pilku peal hoida. Koostame lotomängijale abiks järgmise programmi.

Koostada programm, mis

- küsib kasutajalt mänguks valitud arvud komadega eraldatult ühe sõnena (näiteks "14,20,33,44,28,10");
- loob valitud arvudest sõnede järjendi;
- loob tühja järjendi, kuhu sisestatakse kõik mängus loositud arvud;
- küsib kasutajalt mängus loositud arve seni, kuni järjend on tühi või kasutaja sisestab -1;
- igal arvu (sõne) sisestamisel lisab selle mängus loositud arvude järjendisse;

- juhul, kui loositud arv (sõne) on üks kasutaja sisestatud arvudest, eemaldab nimetatud arvu valitud arvude järjendist. Kui tegemist on järjendi viimase arvuga, väljastab kasutajale “Oled võitnud!” ja lõpetab töö;
- juhul, kui kasutaja sisestab arvu -1, väljastab kõik loositud arvud (tühikutega eraldatult) ja kõik allesjäänud arvud (tühikutega eraldatult) ning lõpetab töö.

Tuletame meelde, et sõne saab eemaldada käsu `list.remove(element)` abil.

Näited programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl4.4c.py'
Sisesta arvude valik: 10,22,30,41,49
Sisesta loositud arv: 55
Sisesta loositud arv: 13
Sisesta loositud arv: 10
Sisesta loositud arv: 22
Sisesta loositud arv: 65
Sisesta loositud arv: 30
Sisesta loositud arv: 41
Sisesta loositud arv: 33
Sisesta loositud arv: 34
Sisesta loositud arv: 49
Oled võitnud!

Sisesta arvude valik: 55,5,18
Sisesta loositud arv: 34
Sisesta loositud arv: 66
Sisesta loositud arv: 5
Sisesta loositud arv: 13
Sisesta loositud arv: 12
Sisesta loositud arv: 24
Sisesta loositud arv: -1
Loositi arvud 34 66 5 13 12 24
Alles jäid arvud 55 18
```

Koduülesanded - 5. nädal

5.1 Täishäälikute tuvastaja

Taavi laps asub sügisel teise klassi õppima ning Taavi mõistab, et võiks lapsega igaks juhuks teadmiste värskendust teha. Üheks teemaks, millega lapsed eesti keele tunnis kohe alguses kokku puutuvad, on häälikud. Tihti antakse õpilastele ülesandeks lugeda sõnas kokku kõik täishäälikud, kaashäälikud ning sulghäälikud. Ülesanne on tegelikult häälikuid tundes lihtne, kuid eksimine on inimlik ja sage, eriti lapse puhul. Taavi mõtleb, et alustuseks võiks lasta lapsel loendada täishäälikuid, mistõttu oleks mõistlik lapsele luua programm, mille abil ta saab kontrollida, kas tema kokku loetud täishäälikute arvud on õiged.

Esmalt koostada funktsioon `täishäälikud`, mis

- võtab argumendiks väikeste tähtedega sõna;
- tagastab sõnas leiduvate täishäälikute arvu.

Näide funktsiooni tööst:

```
>>> täishäälikud("kontsert")
2
```

Vihje: seda, kas täht on täishäälik saab kontrollida luues täishäälikutest järjendi ja uurides, kas täht kuulub järjendisse (nt `if täht in täishäälikute_järjend`).

Seejärel koostada põhiprogramm, mis

- küsib kasutajalt sõna, mille täishäälikute arvu soovitakse teada;
- väljastab sisestatud sõna ja funktsiooni `täishäälikud` kasutades täishäälikute arvu lausena.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl5.1.py'
Sisesta sõna: jääääär
Sõnas jääääär on täishäälikuid 4

>>> %Run 'yl5.1.py'
Sisesta sõna: kuulilennuteetunneliluuk
Sõnas kuulilennuteetunneliluuk on täishäälikuid 12
```

5.2 Ruutvõrrandi lahendaja

Ruutvõrrandi $\boldsymbol{ax^2 + bx + c = 0}$ lahendamine on valemit $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ teades lihtne, kuid igal juhul ajakulukas protsess. Arvutusprotsessi saab kiirendada, kui koostada selleks funktsioon.

Esmalt koostada funktsioon `lahenda`, mis

- võtab argumendiks täisarvulised muutujad a, b ja c;
- tagastab järjendi, mille elementideks on ruutvõrrandi mõlemad lahendid.

Näide funktsiooni tööst:

```
>>> lahenda(2, -11, 5)
[5.0, 0.5]
```

Seejärel koostada põhiprogramm, mis

- küsib kasutajalt ükshaaval muutujate a, b ja c väärtused;
- leiab funktsiooni `lahenda` abil ruutvõrrandi lahendid;
- väljastab ruutvõrrandi lahendid ekraanile lausena.

```
>>> %Run 'yl5.1.py'
Sisesta muutuja a väärtus: 2
Sisesta muutuja b väärtus: -11
Sisesta muutuja c väärtus: 5
Lahendid on 5.0 ja 0.5.
```

Programmi võib katsetada järgmiste võrranditega:

$\mathbf{3}\boldsymbol{x}^{\mathbf{2}} + \mathbf{9}\boldsymbol{x} = \mathbf{0}$ … (c = 0)

$\mathbf{5}\boldsymbol{x}^{\mathbf{2}} + \mathbf{25}\boldsymbol{x} = \mathbf{0}$

$\mathbf{2}\boldsymbol{x}^{\mathbf{2}} - \mathbf{11}\boldsymbol{x} + \mathbf{5} = \mathbf{0}$

$\mathbf{5}\boldsymbol{x}^{\mathbf{2}} + \mathbf{6}\boldsymbol{x} + \mathbf{1} = \mathbf{0}$

5.3 Nimede järjestaja

Andmete sisestajal on hulk nimesid, mida ta soovib kord tähestiku järjekorras, kord tagurpidises tähestiku järjekorras ning sõltumata valikust soovib väljastada need järjekorranumbriga nummerdatult.

Esmalt koostada funktsioon `järjestaja`, mis

- võtab argumendiks koma ja tühikuga eraldatud (“, “) nimed ühe sõnena (nt “John H. Watson, Sherlock Holmes, Jim Moriarty”) ja lisaargumendina tõeväärtuse, mis `True` korral sorteerib järjendi tagurpidises tähestiku järjekorras (vaikimisi on tõeväärtus `False`);
- väljastab vastavalt lisaargumendi väärtusele sisestatud nimed üksteise alla koos järjekorranumbriga tähestiku järjekorras, kui lisaargumendi tõeväärtus on `False` või see puudub, vastasel juhul väljastab sisestatud nimed üksteise alla koos järjekorranumbriga tagurpidises tähestiku järjekorras.

Näited funktsiooni tööst:

```
>>> järjestaja("John H. Watson, Sherlock Holmes, Jim Moriarty")
1. Jim Moriarty
```

```
2. John H. Watson
3. Sherlock Holmes

>>> järjestaja("John H. Watson, Sherlock Holmes, Jim Moriarty", True)
1. Sherlock Holmes
2. John H. Watson
3. Jim Moriarty
```

Vihje: funktsioon `sorted` sorteerib sõnedest koosneva järjendi vaikimisi tähestiku järjekorras, kuid tagurpidi järjekorra saamiseks võib abi olla meetodist `reverse`.

Seejärel koostada põhiprogramm, mis

- küsib kasutajalt sorteeritavaid nimesid koma ja tühikuga eraldatult;
- sorteerib nimed kasutades funktsiooni `järjestaja` tähestiku järjekorras ja väljastab need ekraanile;
- küsib kasutajalt, kas ta soovib nimesid väljastada ka tagurpidises tähestiku järjekorras, mispuhul peab kasutaja sisestama “jah”. Vastasel juhul tuleb vajutada ENTER;
- vastuse “jah” puhul sorteerib ja väljastab kasutades funktsiooni `järjestaja` sisestatud nimed tagurpidises tähestiku järjekorras ja lõpetab töö;
- ENTER sisestamise puhul väljastab “Programmi töö on lõppenud” ja programm lõpetab töö.

Näited programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl5.3.py'
Sisesta nimed, mida soovid järjestada: Sherlock Holmes, Jim Moriarty,
John H. Watson
1. Jim Moriarty
2. John H. Watson
3. Sherlock Holmes

Kas soovid nimesid ka tagurpidises tähestiku järjekorras järjestada? jah
1. Sherlock Holmes
2. John H. Watson
3. Jim Moriarty

>>> %Run 'yl5.3.py'
Sisesta nimed, mida soovid järjestada: James Hunt, Niki Lauda, Michael
Schumacher
1. James Hunt
2. Michael Schumacher
3. Niki Lauda
```

```
Kas soovid nimesid ka tagurpidises tähestiku järjekorras järjestada?
Programmi töö on lõppenud.
```

Praktikumiülesanded - 5. nädal

5.4a Tüdrukute nimed

Peatselt lapsevanemate rolli astuv paar murrab pead, sest nad ei tea, mis tütrele nimeks panna. Väikese eeltöö tulemusena on ühelt veebilehelt saadud erinevad tüdrukute nimed ühte tekstifaili koondatud. Paarile meeldivad näiteks E-tähega algavad nimed, mis on vähemalt 5 tähte pikad. Samas meeldivad neile ka K-tähega algavad nimed, mis on vähemalt 7 tähte pikad. Käsitsi nimesid failist otsida on väga aeganõudev tegevus, mistõttu on mõistlik abistada neid programmiga.

Esmalt koostada funktsioon `nimeotsing`, mis

- võtab argumendiks failist sisse loetud nimede järjendi, soovitud nime esitähe (sõne, ei ole vahet, kas suur- või väiketäht) ja nime minimaalse pikkuse (täisarv);
- leiab nime algustähte ja minimaalset pikkust arvesse võttes nimede järjendist kõik sobivad nimed ning tagastab need ühe sõnena koma ja tühikuga eraldatult (nt “Kristel, Karolin, Kristin”). Tuleb arvestada sellega, et viimase nime järel ei oleks koma.

Näited funktsiooni tööst:

```
>>> nimeotsing(["Ella", "Elis", "Helena", "Elianor", "Paula"], "E", 5)
'Elianor'
>>> nimeotsing(["Ella", "Elis", "Helena", "Elianor", "Paula"], "e", 0)
'Ella, Elis, Elianor'
```

Failis tydrukute_nimed.txt on kirjas tüdrukute nimed.

Seejärel koostada põhiprogramm, mis

- küsib kasutajalt faili nime;
- loeb failist sisse tüdrukute nimed, eemaldades igal real reavahetuse märgise (seda saab teha, kasutades meetodit rstrip);
- küsib kasutajalt sobilikku nime algustähte;
- küsib kasutajalt nime minimaalset pikkust;
- väljastab sisestatud nime algustähe ja minimaalse pikkuse alusel sobivad nimed, rakendades funktsiooni `nimeotsing`.

Näited programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl5.4a.py'
Sisesta faili nimi: tydrukute_nimed.txt
Sisesta nime algustäht: K
Sisesta nime minimaalne pikkus: 8

Sobilikud nimed on:
Karoliina
Karmeelia
Kristiina
Kristella
Kristina
Karoline

>>> %Run 'yl5.4a.py'
Sisesta faili nimi: tydrukute_nimed.txt
Sisesta nime algustäht: e
Sisesta nime minimaalne pikkus: 0

Sobilikud nimed on:
Edith
Eeva
Elisabeth
Eike
Elisa
Elora
Ellike
Ea
```

5.4b Peastarvutamine aja peale

Pillel on koolis tulemas harivate mängude päev, mille raames peab iga õpilane välja mõtlema ja valmis tegema mingisuguse haridusliku sisuga mängu. Pillele endale meeldib matemaatika, seega on tema esimene mõte, et võiks teha peastarvutamise mängu. Mäng oleks veelgi huvitavam, kui saaks kuidagi peastarvutamise aega mõõta.

Esmalt koostada funktsioon `arvutus`, mis

- võtab argumendiks sümboli, mis kujutab liitmistehet (+), lahutamistehet (-) või korrutamistehet (*);
- genereerib kaks juhuslikku täisarvu vahemikus 0-9;
- vastavalt sisestatud sümbolile tagastab kahe genereeritud arvu summa, vahe või korrutise ning sooritatava tehte sõne kujul ennikuna, nt (3, “9 - 6 = “).

Näited funktsiooni tööst:

```
>>> arvutus("+")
(11, '6 + 5 = ')

>>> arvutus("-")
(0, '5 - 5 = ')

>>> arvutus("*")
(18, '9 * 2 = ')
```

Seejärel koostada põhiprogramm, mis

- küsib kasutajalt tehte sümbolit, milleks saab olla liitmistehe (+), lahutamistehe (-) või korrutamistehe (*), seni, kuni kasutaja vastab tühisõnega (vajutab ENTER), mispuhul väljastab programm teate “Programmi töö on lõppenud.” ja lõpetab töö;
- vastavalt sisestatud sümbolile genereerib tehte sõne kujul, kasutades selleks funktsiooni `arvutus`;
- küsib kasutajalt sõne kujul saadud tehte vastust;
- mõõdab kasutaja vastamisaega, ümardades selle kahe komakohani;
- annab õige vastuse korral kasutajale sellest teada koos vastamiseks kulunud ajaga ning vale vastuse korral koos õige vastuse ja vastamiseks kulunud ajaga.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl5.4b.py'
Sisesta tehe: +
2 + 4 = 6
Õige vastus! Vastamiseks kulus aega 1.25 sekundit.

Sisesta tehe: -
5 - 6 = -1
Õige vastus! Vastamiseks kulus aega 1.48 sekundit.

Sisesta tehe: *
3 * 4 = 12
Õige vastus! Vastamiseks kulus aega 1.27 sekundit.

Sisesta tehe: +
6 + 2 = 200
Vale vastus! Õige vastus on 8. Vastamiseks kulus aega 1.05 sekundit.

Sisesta tehe:
Programmi töö on lõppenud.
```

```

```

Vihje: kuidas aega mõõta

```
import datetime

start = datetime.datetime.now()
#tegevused
lõpp = datetime.datetime.now()
vahe = lõpp - start
sekundites = vahe.total_seconds()
```

5.4c Õigekirja korrigeerija

Magnettormi tagajärjel on juhtunud õnnetus - ühes väga olulises veebipoele kuuluvas tekstifailis on osadele sõnadele, kus leidub kaks sama tähte järjest, näiteks “kuppel”, lisandunud ka kolmas sama täht juurde (nt “kupppel”). Kuigi fail ei ole suur, siis võib sarnaseid vigaseid faile tulevikus ilmneda veelgi, mistõttu oleks tarvilik koostada üks programm, mis korrigeerib ja väljastab vigased tootenimed.

Esmalt koostada funktsioon `korrigeeri`, mis

- võtab argumendiks tootenimetuse;
- kontrollib, kas tootenimetuses leidub kolm sama tähte järjest;
- kui tootenimetuses leidub kolm sama tähte järjest, korrigeerib tootenimetust, eemaldades üleliigse tähe ja tagastab korrigeeritud tootenimetuse;
- kui tootenimetuses ei leidu kolm sama tähte järjest, tagastab funktsioon tootenimetuse korrektsioonideta;

Näited funktsiooni tööst:

```
>>> korrigeeri("tugitool")
'tugitool'
>>> korrigeeri("tugitoool")
'tugitool'
```

Failis tooted.txt on tootenimetused eraldi ridadel.

Seejärel koostada põhiprogramm, mis

- küsib kasutajalt faili nime;
- loeb failist sisse tootenimetused ridade kaupa;
- kontrollib kõiki tootenimetusi funktsiooni `korrigeeri` abil;
- väljastab vigase tootenimetuse korral tootenimetuse vigase ja korrigeeritud variandi (kui tootenimetus oli korrektne, ei väljasta programm selle kohta midagi);

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl5.4c.py'
Sisesta faili nimi: tooted.txt
Korrigeeriti toode: lambikupppel -> lambikuppel
Korrigeeriti toode: põrandaliiistud -> põrandaliistud
Korrigeeriti toode: tugitoool -> tugitool
Korrigeeriti toode: diiivanipadjad -> diivanipadjad
Korrigeeriti toode: vannn -> vann
Korrigeeriti toode: külmkappp -> külmkapp
Korrigeeriti toode: raaamaturiiiul -> raamaturiiul
Korrigeeriti toode: kraanikausss -> kraanikauss
Korrigeeriti toode: sektsiooonkapp -> sektsioonkapp
Korrigeeriti toode: kummmut -> kummut
Korrigeeriti toode: põrandaplaaadid -> põrandaplaadid
Korrigeeriti toode: tapeeet -> tapeet
```

Koduülesanded - 6. nädal

6.1 Peoliste nimekiri

Priit ja Tarmo peavad koos sünnipäeva ning neil on peole kutsutute nimekiri ühises serveris asuvas failis, kuhu nad sooviksid mõlemad vajadusel nimesid juurde lisada. Et sünnipäevaliste tööd lihtsustada, on neid võimalik abistada programmiga, mis nimesid küsib ja neid faili lõppu juurde kirjutab.

Näidisfaili esialgne sisu:

```
Karl
Kertu
Jürgen
Kristin
```

Koostada programm, mis

- küsib kasutajalt peoliste nimekirja sisaldava faili nime;
- küsib kasutajalt lisatavate peoliste nimesid seni, kuni kasutaja vastab “lõpp”, mispuhul väljastab teate, et faili kirjutamine on lõppenud, ja programm lõpetab töö;
- pärast iga nime sisestamist väljastab teate kujul “Nimi 'Sille' on sisestatud!”
- kirjutab sisestatud nimed faili lõppu, iga nimi eraldi reale (tuletame meelde sõnede reavahetuse käsku);

Kindlasti tasub meeles pidada, et fail tuleb kirjutamise lõpus sulgeda.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl6.1.py'
Sisesta faili nimi: peolised.txt
Sisesta peolise nimi: Madis
Nimi 'Madis' sisestatud!
Sisesta peolise nimi: Sille
Nimi 'Sille' sisestatud!
Sisesta peolise nimi: lõpp
Faili kirjutamine on lõppenud!
```

Näidisfaili sisu pärast programmi tööd:

```
Karl
Kertu
Jürgen
Kristin
Madis
Sille
```

6.2 Logi

Kõik programmid, mida kasutame, jätavad kuhugi maha endast logi ehk arvuti/programmi tegevuste päeviku. Kristiinal on hilisemaks veebiarenduseks vaja välja töötada üks lihtsakoeline logiprogramm, mille abil saab logisse kirjutada ja logi sissekandeid lugeda. Aitame Kristiinat järgmise programmiga.

Näidisfaili esialgne sisu:

```
2022-02-24 16:03:03.099725: Testime, kas programm töötab
```

NB! Võib eeldada, et sissekande lõpus on reavahetus.

Koostada programm, mis

- küsib kasutajalt logifaili nime;
- küsib kasutajalt, kas soovitakse logisse kirjutada või logi sissekandeid lugeda. Logisse kirjutamiseks tuleb kasutajal sisestada “kirjuta”, logist lugemiseks tuleb sisestada “loe”;
- juhul, kui kasutaja soovib logisse kirjutada, küsib kasutajalt informatsiooni, mida soovitakse logisse kirjutada. Saadud informatsioon salvestatakse koos kuupäeva ja kellaajaga kujul “*2022-02-24 16:03:03.099725: Sissekanne”* logifaili lõppu eraldi reale.
- juhul, kui kasutaja soovib logi lugeda, väljastab ekraanile logifaili sisu.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl6.2.py'
Sisesta logifaili nimi: logi.txt
Kas tahad logisse kirjutada või logi lugeda? loe

2022-02-24 16:03:03.099725: Testime, kas programm töötab

>>> %Run 'yl6.2.py'
Sisesta logifaili nimi: logi.txt
Kas tahad logisse kirjutada või logi lugeda? kirjuta
Sisesta logi sissekanne: Programm töötab kenasti

>>> %Run 'yl6.2.py'
Sisesta logifaili nimi: logi.txt
Kas tahad logisse kirjutada või logi lugeda? loe

2022-02-24 16:03:03.099725: Testime, kas programm töötab
2022-02-24 16:05:14.472512: Programm töötab kenasti
```

Vihje: Kuupäeva ja kellaaja saab soovitud kujul, kui kasutada *datetime* teegist funktsiooni `datetime.today()`.

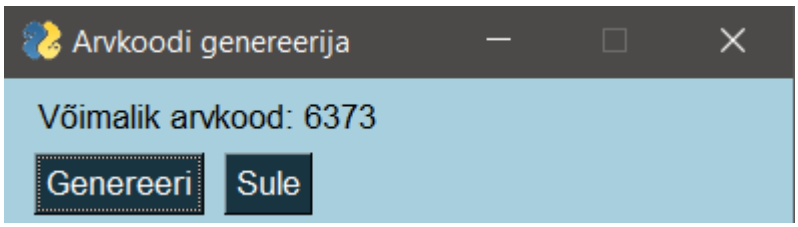
6.3 Arvkoodi genereerija

Uue arvkoodi seadmisel (näiteks nutitelefoni ekraaniluku tarbeks) võidakse valida kogemata liiga äraarvatav kood, mistõttu on hea lasta programmil genereerida juhuslik kood.

Koostada PySimpleGUI graafilise kasutajaliidesega arvkoodi genereerimise programm, mis

- genereerib kasutajale programmi avamisel neljakohalise juhuslikest täisarvudest koosneva arvkoodi;
- genereerib kasutajale uue neljakohalise arvkoodi, kui kasutaja vajutab nuppu “Genereeri”;
- suleb liidese ja programmi, kui kasutaja vajutab “Sule”.

Näited programmi tööst:



Praktikumiülesanded - 6. nädal

6.4a Vigane pangaautomaat

Pangaautomaat on läinud rikki ja lubab selle kasutajal võtta raha välja ja panna sisse, nii nagu süda lustib. Rikki läinud pangaautomaadil on lisaks säilinud varasemalt tehtud tehingute vaatamise funktsioon. Simuleerime programmi tööd koodijupiga.

Koostada programm, mis

- küsib kasutajalt, kas kasutaja soovib sooritada tehingut, vaadata kontoseisu või lõpetada töö;
- juhul kui kasutaja vastab “tehing”, palub kasutajal täisarvulise summa sisestada. Kui kasutaja on summa sisestanud, kirjutab programm summa ja tehingu kuupäeva ning kellaaja faili *pank.txt* lõppu (võib eeldada, et fail on enne programmi esimest käivitamist tühi) ning seejärel küsib programm kasutajalt uuesti tegevust;
- juhul kui kasutaja vastab “väljavõte”, loeb ja väljastab faili *pank.txt* sisu ekraanile ning seejärel küsib kasutajalt uuesti tegevust;
- juhul kui kasutaja vastab “lõpp”, väljastab teate “Sessioon on lõppenud.” ning lõpetab töö.

Näited programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl6.4a.py'
Tehing, väljavõte või lõpetamine?: tehing
Sisesta summa: 500
Tehing, väljavõte või lõpetamine?: tehing
Sisesta summa: -300
Tehing, väljavõte või lõpetamine?: väljavõte

Tehingu aeg: 2022-02-22 22:07:49.355839, summa: 500
Tehingu aeg: 2022-02-22 22:07:51.788771, summa: -300

Tehing, väljavõte või lõpetamine?: lõpp
Sessioon on lõppenud

>>> %Run 'yl6.4a.py'
Tehing, väljavõte või lõpetamine?: tehing
Sisesta summa: 150
Tehing, väljavõte või lõpetamine?: väljavõte

Tehingu aeg: 2022-02-22 22:07:49.355839, summa: 500
Tehingu aeg: 2022-02-22 22:07:51.788771, summa: -300
Tehingu aeg: 2022-02-22 22:08:06.920343, summa: 150
```

```
Tehing, väljavõte või lõpetamine?: lõpp
Sessioon on lõppenud
```

6.4b Linna tuli rekkatäis..

Pikkadel autosõitudel ja bussireisidel viiakse tihti enese lõbustamiseks läbi sõnamängu “Linna tuli rekkatäis..”. Mängu alustaja paneb paika kategooria, öeldes avalausena näiteks “Linna tuli rekkatäis erinevaid automarke”, misjärel peavad kõik seltskonnas viibijad hakkama ükshaaval nimetama erinevaid automarke, mis ei tohi korduda. Karmide reeglite korral võib korduva sõna öelnud mängija mängust välja arvata, kuid sõbralikes kollektiivides antakse lihtsalt märku, et sõna on juba öeldud, ja minnakse mänguga edasi. Kõige tulisem küsimus on väga kaua kestnud mängu puhul aga see, kas sõna on juba öeldud või ei ole. Sellest murest saab kiiresti lahti, kui kirjutada valmis programm, mis salvestab sõnad faili nii, et pärast tanklapausi saab seltskond kenasti edasi mängida.

Koostada programm, mis

- küsib kasutajalt eelnevalt loodud tühja või juba erinevatel ridadel sõnu sisaldava faili nime (näiteks sonamang.txt);
- loeb failist sisse kõik sõnad (kirjutatud eraldi ridadele);
- küsib kasutajalt uusi sõnu seni, kuni kasutaja sisestab “aitab”, misjärel kirjutab kõik failist saadud sõnad (kui neid on) ja praeguse programmi töö käigus sisestatud sõnad faili lõppu üksteise alla erinevatele ridadele ja väljastab kasutajale teate, et mäng on peatatud ja programm lõpetab töö;
- juhul, kui failis on sõnu, kontrollib, kas sõna eksisteerib juba öeldud sõnade nimistus või mitte. Kui sõna eksisteerib nimistus, annab sellest kasutajale märku ja küsib kasutajalt uue sõna. Kui sõna ei eksisteeri nimistus, salvestab selle ning väljastab teate kujul “Sõna 'programm' sisestatud. Mängus on kasutatud 9 erinevat sõna.” ning küsib kasutajalt uue sõna.

Antud programmi kirjapanemisel on tark kasutada konstruktsiooni `while True`. Sellise *while*-tsükli kasutamise puhul saab programmist tsüklist väljuda, kui kasutada käsku `break`.

Näide programmi tööst (kasutaja sisend on paksus kirjas):

```
>>> %Run 'yl6.4b.py'
Sisesta faili nimi: sonamang.txt
Sisesta sõna: koer
Sõna 'koer' sisestatud. Mängus öeldud erinevaid sõnu: 1.
Sisesta sõna: ahvipoeg
Sõna 'ahvipoeg' sisestatud. Mängus öeldud erinevaid sõnu: 2.
Sisesta sõna: kass
Sõna 'kass' sisestatud. Mängus öeldud erinevaid sõnu: 3.
Sisesta sõna: hamster
```

```
Sõna 'hamster' sisestatud. Mängus öeldud erinevaid sõnu: 4.
Sisesta sõna: merisiga
Sõna 'merisiga' sisestatud. Mängus öeldud erinevaid sõnu: 5.
Sisesta sõna: papagoi
Sõna 'papagoi' sisestatud. Mängus öeldud erinevaid sõnu: 6.
Sisesta sõna: kass
See sõna on juba öeldud.
Sisesta sõna: kilpkonn
Sõna 'kilpkonn' sisestatud. Mängus öeldud erinevaid sõnu: 7.
Sisesta sõna: koer
See sõna on juba öeldud.
Sisesta sõna: aitab
Mäng on peatatud.
```

Näide faili sisust pärast programmi tööd:

```
koer
ahvipoeg
kass
hamster
merisiga
papagoi
kilpkonn
```

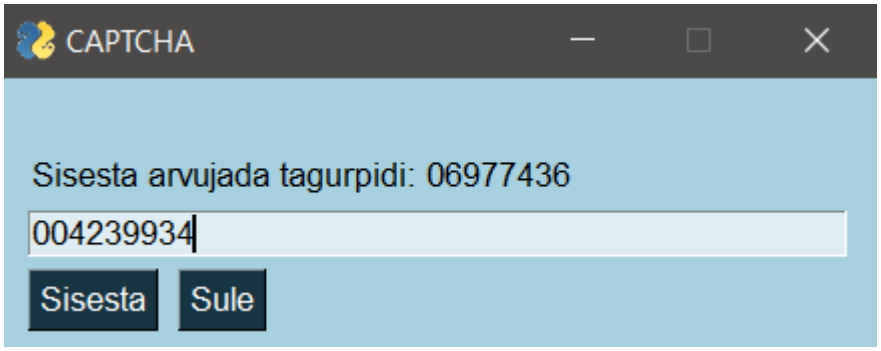
6.4c CAPTCHA

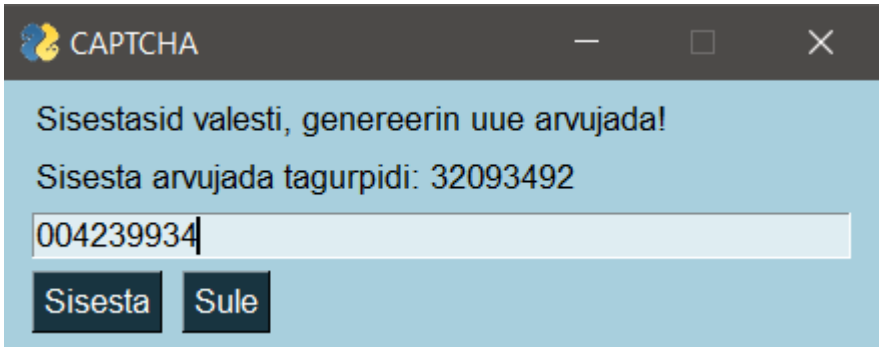
Kes veebis on parasjagu palju toimetanud, on ühel või teisel hetkel lühendiga CAPTCHA kokku puutunud. CAPTCHA ehk *Completely Automated Public Turing test to tell Computers and Humans Apart* on loodud selleks, et eristada päris kasutajat pahavarast või muusugusest arvutiskriptist, kes pakutavale sisule ligi üritab pääseda. Tavaliselt tuleb “testi” sooritades kasutajal sisestada korrektne tekst, mis näidatud pildil on hägustunud kujul, või nt valida kirjelduse järgi õige kujuga ese.

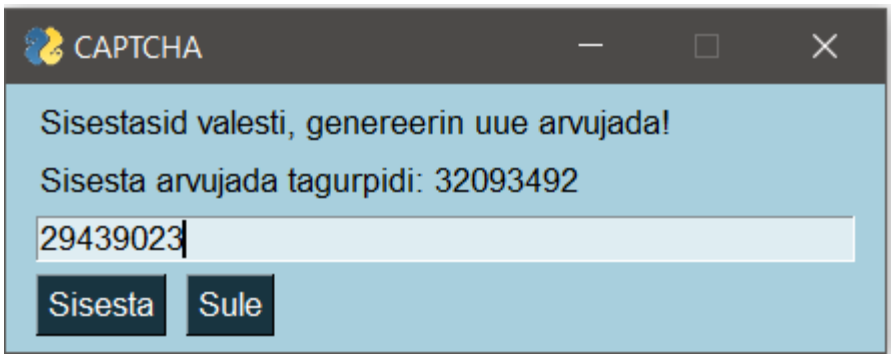
Koostada PySimpleGUI graafilise kasutajaliidesega CAPTCHA programm, mis

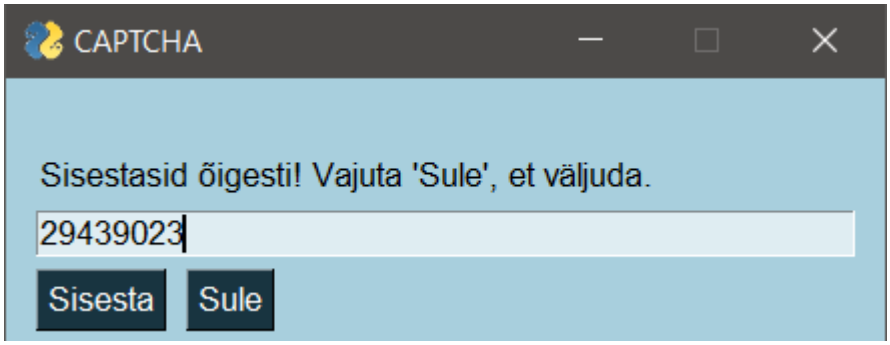
- väljastab kasutajale teksti “Sisesta arvujada tagurpidi:” ja selle alla juhuslikult genereeritud numbritest koosneva sõne, mis on pikkusega 8 tähemärki;
- palub kasutajal sisestada antud numbritest koosnev sõne tagurpidi;
- sulgeb liidese, kui kasutaja sisestab sõne vastavalt tingimustele;
- väljastab kasutajale “Sisestasid valesti, genereerin uue arvujada!”, kui kasutaja sisestab arvujada valesti, ja genereerib uue arvujada, mille kasutaja peab vastavalt tingimustele sisestama.

Näide programmi tööst:


CAPTCHA
Sisesta arvujada tagurpidi: 06977436
004239934
Sisesta
Sule


CAPTCHA
Sisestasid valesti, genereerin uue arvujada!
Sisesta arvujada tagurpidi: 32093492
004239934
Sisesta
Sule


CAPTCHA
Sisestasid valesti, genereerin uue arvujada!
Sisesta arvujada tagurpidi: 32093492
29439023
Sisesta
Sule


CAPTCHA
Sisestasid õigesti! Vajuta 'Sule', et väljuda.
29439023
Sisesta
Sule

II. Küsitluse ankeedi näidis

Tagasisidevorm tudengitele (1. ja 2. nädal)

Palume teie tagasisidet esimese ja teise nädala kodu- ja praktikumiülesannete kohta kursusel "Programmeerimise alused". Selle küsitluse palume täita siis, kui olete kõik esimese ja teise nädala kohustuslikud ülesanded täitnud. Teie tagasiside aitab kursuse ülesandeid paremaks muuta ning vastuseid kasutatakse üldistatult teadustöös. Küsitlusele on vastamiseks aega 3. märtsini ja vastamise eest saab kursusel 1 boonuspunkti. Vastamine ei ole anonüümne (et teaksime, kellele punkt anda), kuid küsimustikust tulenevaid vastuseid või analüüse ei avaldata isikuga seostatult.

* Kohustuslik

Ees- ja perekonnanimi *

Teie vastus

Palun hinnake alljärgneva skaala alusel 1. nädala praktikumiülesannete keerukust. *

	1 - Liiga lihtne	2	3	4	5 - Liiga keeruline	Ei lahendanud seda ülesannet
1.1. Tervitus	◯	◯	◯	◯	◯	◯
1.2. Aasta loom	◯	◯	◯	◯	◯	◯
1.3. Ringi ümbermõõdu arvutamine	◯	◯	◯	◯	◯	◯
1.4. Sammulugeja	◯	◯	◯	◯	◯	◯
1.5. Kergejõustiku kahevõistlus	◯	◯	◯	◯	◯	◯
1.6. Küpsisetort	◯	◯	◯	◯	◯	◯

Soovi korral palun kommenteerige oma vastust.

Teie vastus

Palun hinnake alljärgneva skaala alusel 1. nädala praktikumiülesannete huvitavust. *

	1 - Ebahuvitav	2	3	4	5 - Väga huvitav	Ei lahendanud seda ülesannet
1.1. Tervitus	○	○	○	○	○	○
1.2. Aasta loom	○	○	○	○	○	○
1.3. Ringi ümbermõõdu arvutamine	○	○	○	○	○	○
1.4. Sammulugeja	○	○	○	○	○	○
1.5. Kergejõustiku kahevõistlus	○	○	○	○	○	○
1.6. Küpsisetort	○	○	○	○	○	○

Soovi korral palun kommenteerige oma vastust.

Teie vastus

Palun hinnake alljärgneva skaala alusel 1. nädala praktikumiülesannete tekstide selgust. *

	1 - Tekst oli ebaselge	2	3	4	5 - Tekst oli väga selge	Ei lahendanud seda ülesannet
1.1. Tervitus	◯	◯	◯	◯	◯	◯
1.2. Aasta loom	◯	◯	◯	◯	◯	◯
1.3. Ringi ümbermõõdu arvutamine	◯	◯	◯	◯	◯	◯
1.4. Sammulugeja	◯	◯	◯	◯	◯	◯
1.5. Kergejõustiku kahevõistlus	◯	◯	◯	◯	◯	◯
1.6. Küpsisetort	◯	◯	◯	◯	◯	◯

Soovi korral palun kommenteerige oma vastust.

Teie vastus

Palun hinnake alljärgneva skaala alusel 2. nädala kodu- ja praktikumiülesannete keerukust. *

	1 - Liiga lihtne	2	3	4	5 - Liiga keeruline	Ei lahendanud seda ülesannet
2.1 Parool	◯	◯	◯	◯	◯	◯
2.2 Leedu perenimed	◯	◯	◯	◯	◯	◯
2.3 Elektriauto	◯	◯	◯	◯	◯	◯
2.4a Rannailm	◯	◯	◯	◯	◯	◯
2.4b Istekoht	◯	◯	◯	◯	◯	◯
2.4c Bussid	◯	◯	◯	◯	◯	◯
2.5 Nädalapalk	◯	◯	◯	◯	◯	◯

Soovi korral palun kommenteerige oma vastust.

Teie vastus

Palun hinnake alljärgneva skaala alusel 2. nädala kodu- ja praktikumiülesannete huvitavust. *

	1 - Ebahuvitav	2	3	4	5 - Väga huvitav	Ei lahendanud seda ülesannet
2.1 Parool	◯	◯	◯	◯	◯	◯
2.2 Leedu perenimed	◯	◯	◯	◯	◯	◯
2.3 Elektriauto	◯	◯	◯	◯	◯	◯
2.4a Rannailm	◯	◯	◯	◯	◯	◯
2.4b Istekoht	◯	◯	◯	◯	◯	◯
2.4c Bussid	◯	◯	◯	◯	◯	◯
2.5 Nädalapalk	◯	◯	◯	◯	◯	◯

Soovi korral palun kommenteerige oma vastust.

Teie vastus

Palun hinnake alljärgneva skaala alusel 2. nädala kodu- ja praktikumiülesannete tekstide selgust. *

	1 - Tekst oli ebaselge	2	3	4	5 - Tekst oli väga selge	Ei lahendanud seda ülesannet
2.1 Parool	◯	◯	◯	◯	◯	◯
2.2 Leedu perenimed	◯	◯	◯	◯	◯	◯
2.3 Elektriauto	◯	◯	◯	◯	◯	◯
2.4a Rannailm	◯	◯	◯	◯	◯	◯
2.4b Istekoht	◯	◯	◯	◯	◯	◯
2.4c Bussid	◯	◯	◯	◯	◯	◯
2.5 Nädalapalk	◯	◯	◯	◯	◯	◯

Soovi korral palun kommenteerige oma vastust.

Teie vastus

Millised ülesanded vajaksid Teie arvates muutmist?

Teie vastus

Millised teemad oleks võinud Teie arvates ülesannetes rohkem kajastust leida?

Teie vastus

Võite ülesannete ülesehituse, sisu, käsitletavate teemade kohta soovi korral kommentaare anda.

Teie vastus

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Patrik Pruunsild,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Tartu Ülikooli kursuse "Programmeerimise alused" uus ülesannete kogu,

   mille juhendaja on Heidi Meier,

   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 4.0, 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.

Patrik Pruunsild
10.05.2022