TARTU ÜLIKOOL
LOODUS- JA TÄPPISTEADUSTE VALDKOND
Arvutiteaduse instituut
Informaatika õppekava

Kadi Sammul

Gümnaasiumi programmeerimise eksami koostamine ja katsetamine

Bakalaureusetöö (9 EAP)

Juhendaja: Tauno Palts

Tartu 2022

Gümnaasiumi programmeerimise eksami koostamine ja katsetamine

Lühikokkuvõte:

Bakalaureusetöö eesmärgiks on luua eksamiülesannete komplekt, millega kontrollitakse sooritajate programmeerimise algteadmisi keeles Python. Selleks uuriti erinevaid programmeerimise algteadmisi õpetavaid avalikult kättesaadavaid eestikeelseid materjale ning loodi nende põhjal teemade nimekiri, mida algajad programmeerijad oskama peaksid. Vastavate teadmiste kontrollimiseks loodi kaks eksamiülesannete komplekti, mida huvilised lahendasid Tartus ja Tallinnas. Analüüsiti läbiviidud eksami tulemusi ning tagasisidet. Eksam koosnes nii paberil kui arvutis lahendatavast osast. Arvutiosas võis kasutada internetis leiduvaid materjale, paberosas mitte. Eksami edukas sooritamine tagas koha Tartu Ülikooli informaatika ja arvutitehnika õppekavadel.

Võtmesõnad:

Eksamiülesannete komplekt, Python, programmeerimise algkursus

CERCS: P175 Informaatika, süsteemiteooria

Creating and testing a programming exam for gymnasium

Abstract:

The goal of the Bachelor's thesis is to create a set of tasks for an exam, that test the participants' knowledge in basic programming in Python. For that different publicly available starting courses in Python programming in Estonian were analyzed. Based on the analysis was formed a list composed of what a beginner in programming should know. To test this knowledge two task sets were created and solved in Tartu and Tallinn. The results of these exams were analyzed along with gotten feedback. The exam consisted of two parts – one solved on paper and other in the computer, during the second of which was allowed to use materials found on the web, which was not allowed in the part on paper. The successful result of the exam guaranteed a place to study in University of Tartu in Informatics and Computer Science.

Keywords:

Set of exam tasks, Python, basic programming course

CERCS: P175 Informatics, systems theory

Sisukord

1. Sissejuhatus

Selle töö käigus luuakse eksamiülesannete komplekt, millega kontrollitakse sooritajate programmeerimise algteadmisi keeles Python. Eksamikomplekti loomine on tingitud soovist võimaldada õpilastel kindlustada endale õppekoht ülikoolis eraldiseisvana riigieksamite tulemustest. Selline võimalus tagab, et õpilased ei pea panema kogu lootust riigieksamitele ja saavad oma võimete tõestamiseks mitu võimalust.

Eksam koosneb kahest osast – paberil lahendatavast ja arvutis lahendatavast osast (edaspidi paberosa ja arvutiosa). Paberosa tuleb sooritada ilma väliste abivahenditeta ning see sisaldab viit lihtsamat sorti ülesannet, mida on võimalik kergesti paberil läbi mängides lahendada. Arvutiosa sisaldab kahte ülesannet, mis nõuavad sooritajalt programmi kirjutamist. Arvutiosas võib kasutada arvutit ja abimaterjale (suhtlemine ja teistelt abi küsimine ei ole lubatud).

Eksami edukaks läbimiseks tuleb saada mõlemast osast vähemalt 50% tulemus. Paberosast on kokku võimalik saada 40 punkti ja arvutiosast 60 punkti. Sooritades eksami vähemalt 50% tulemusega võetakse sooritaja vastu Tartu Ülikooli bakalaureuse informaatika ja arvutitehnika õppekavadele. Tulemusega vähemalt 90% saab sooritaja lisaks ka Tartu Ülikooli (edaspidi TÜ) kursuse „Programmeerimine“ (LTAT.03.001) läbitud.

Erinevaid võimalusi ülikooli astumiseks on Tartu Ülikoolil kasutusel mitmeid. Sõltuvalt erialast on lisaks riigieksamitele on võimalik endale õppekoht kindlustada akadeemilise testiga, silmapaistvate olümpiaaditulemustega või informaatikasse astudes ka näiteks TÜ kursuse „Tehnoloogia tarbijast loojaks“ (LTAT.TK.011) [1] ja TÜ MOOCi „Programmeerimise alused“ [2] abil. Nüüd lisandus viimastele ka käesoleva töö raames valminud eksamikomplekt.

Erinevalt „Tehnoloogia tarbijast loojaks“ ja „Programmeerimise alused“ kursustest, mis eeldavad kursuse läbimist – materjali läbitöötamist, jooksvate tööde lahendamist ja testide või muude kontrollivate tööde sooritamist – loob antud eksamikomplekt lisavõimaluse eelkõige neile, kes on antud materjali juba omandanud ning ei vaja selle sooritamiseks terve kursuse läbi töötamist. Seega võib ühe näitena selle võimaluse põhiliseks sihtrühmaks tuua need gümnaasiumiõpilased, kes koolis suuremal määral omandavad programmeerimise põhitõed ja soovivad nende põhjal kindlustada endale õppekoha Tartu Ülikooli informaatika ja arvutitehnika õppekavadel. Samamoodi kuuluvad siia ka need, kes on isiklikust huvist või mõnel muul põhjusel iseseisvalt vastavad teadmised omandanud.

Programmeerimise õpetamine gümnaasiumides pole reguleeritud, seega on võimalik loodud eksami abil määratleda, milliseid teadmisi ja mil määral võiksid õpilased omandada. See annab aluse luua edaspidi programmeerimise õpetamiseks konkreetseid ühiseid õppematerjale, mis käsitlevad vastavaid teemasid ja tagavad, et gümnaasiumihariduse raames saadud programmeerimisalased teadmised on enam-vähem ühesugused.

Lisavõimaluste loomine ülikooli astumiseks annab ka võimaluse oma võimeid proovile panna korduvalt, tänu millele on võimalik minimeerida olukordi, kus kandidaadi pinge kasvab töö ajal, sest kõik sõltub ühest ainsast katsest. Suure pinge all töötades võib olla inimestel raskem keskenduda, mistõttu võivad tulemused olla madalamad kui muidu oleks [3]. Seetõttu ongi hea luua võimalusi sisse astumiseks juurde, et ei peaks lootma vaid ühele võimalusele (näiteks riigieksamite tulemustele).

2. Programmeerimise algkursused Eestis

Selles peatükis saab ülevaate levinumatest avalikest materjalidest, mille abil on võimalik Eestis Pythoni baasil programmeerimise algteadmisi omandada. Käsitlusel on nii gümnaasiumides kasutusel olev programmeerimise õpik kui ka erinevad Tartu Ülikooli kursused. Materjalide valiku aluseks sai nende suuremas osas vaba kättesaadavus huvilistele, õppekeel ning programmeerimisalaste eelteadmiste vajalikkuse puudumine. Kuna töö raames valminud eksamikomplekti väga edukal sooritamisel saab TÜ kursuses „Programmeerimine“ (LTAT.03.001) arvestuse, siis lisandus ka see antud valikusse.

2.1 Programmeerimise õpik

Eno Tõnissoni ja teiste poolt loodud veebiõpik [4] on kasutusel programmeerimise õpetamisel mitmetes gümnaasiumides üle Eesti. Õpik on ka loodava informaatikaõppekava kursust „Programmeerimine“ ametlikult toetav õpik [5].

Esimese asjana tutvustatakse kasutajale algoritmilist mõtlemist ja põhilisi andmestruktuure ning lihtsamaid sisend-väljund-käske. Algoritmiline mõtlemine on lähtepunkt, millest on võimalik hakata teadmisi ja oskusi edasi arendama, kuna kõik järgnevad konstruktsioonid on samuti algoritmilise ülesehitusega.

Esimene käsitletav konstruktsioon on tingimuslause, et õpilastel oleks võimalik juba pisemaid programme ka reaalselt luua ja seeläbi keele loogikat mõista. Tutvustatakse erinevaid hargnemisi, loogilisi avaldisi ja ka keerulisemaid juhte, näiteks olukordi, kus tingimuslause sisaldab omakorda tingimuslauset. Tingimuslausete tööpõhimõtte mõistmine on alguses äärmiselt oluline, kuna seda omandamata on väga keeruline luua isegi lihtsamaid programme.

Olles aru saanud tingimuslausetest ja selle kaudu ka loogiliste avaldiste tööpõhimõtetest tutvustatakse järgmise konstruktsioonina tsüklit. Tsüklite abil on võimalik õpilastele tutvustada üht programmeerimise alustala – algoritmilist mõtlemist. Ka lihtsama ülesande lahendamine tsükliga nõuab, et õpilane suudaks vastava osa ülesandest taandada olukorrale, kus mingit sama toimingut tehakse korduvalt mingi suurema eesmärgi saavutamiseks. *While*-tsükli käsitlemisel tutvustatakse ka juhuarvude genereerimist.

Programmide loomise juures ei tasuks kunagi ära unustada suhtlust kasutajaga. Programmi koodiga tutvumata on väga raske teada, mis kujul sisendinfot programm tööks vajab kui seda

kasutajatele ei öelda. Seega pühendatakse järgmiseks aega sõnetöötlusele, et saaks olemasoleva ja/või küsitava info viia sobivale kujule ning sobivalt kasutajaga suhelda. Lisaks käsitletakse lihtsamate graafiliste liideste loomist, mis võimaldab teha kasutajale programmi kasutamise mugavamaks ja arusaadavamaks.

Kuna küllaltki tihti on vaja kasutuses hoida rohkem infot korraga, siis võetakse käsitlusele suurema andmestruktuurina järjend, mille abil tutvutakse võimalusega hoida rohkem andmeid ühes muutujas. Selle raames tuuakse välja *for*-tsükli töö, mis lihtsustab järjendi elemendilist läbivaatust ning annab *while*-tsüklile alternatiivse valiku. Järjendi käsitlemise käigus tutvustatakse lihtsamaid Pythoni oma funktsioone, kuna järjendite omaduste kasutamisel on need vajalikud (näiteks *len*, *min*, *max*, *sum*, *sorted* jne). Lisaks võetakse ette ka selliseid, mida tsüklites rohkem kasutatakse (näiteks *range*). Kuna suurema hulga andmetega töötades on kasutajalt neid tülikas küsida, siis õpitakse *for*-tsükliga ka olemasolevaid faile avama ja sealt teksti lugema ning siis neid näiteks järjendisse talletama.

Pärast seda, kui järjendite raames on juba olemasolevate funktsioonide kasutamisega tutvust tehtud, tutvustatakse funktsioone detailsemalt ja võetakse ette nende loomine. Sellega näidatakse, kuidas on võimalik ülesannetest leida korduvaid tegevusi, mida on võimalik just funktsioonide abil vaid ühekordselt defineerida ja seejärel mitmekordselt kasutada. Üritatakse välja tuua võimalikult palju erinevaid variante funktsioonide parameetrite ja tagastusväärtuste osas, et näidata, kuivõrd palju funktsioonid saavad erineda vastavalt nende otstarbele ja kasutusviisile.

Andmete lugemisele ja töötlemisele lisaks võib olla vaja osata neid ka talletada, seega võetakse viimasena ette failidesse kirjutamine. Failidest andmete lugemisega oli eelnevalt vähesel määral küll kokkupuuteid olnud, aga põhjalikumalt võetakse see ette viimase teemana. Et õppimise käigus ei peaks andmete lugemiseks alati oma arvutisse faile looma, siis on käsitlusel ka nende veebist lugemine

2.2 Programmeerimisest maalähedaselt

Kursus „Programmeerimisest maalähedaselt“ on e-kursus [6], mille õppematerjalid on avalikult kättesaadavad. Materjal sisaldab pisikesi enda kontrollimise eesmärgil loodud küsimusi ja mõningaid kontrollülesandeid. Ülesannete juures on viidad murelahendajatele, mis aitavad

tähelepanu juhtida levinumatele probleemidele juhuks, kui ei suuda mõista miks lahendus ei tööta.

Sissejuhatusena programmeerimisse näidatakse esmalt, kuidas saab tavalisi tegevusi teisendada algoritmideks, et oleks võimalik luua arvutile mõistetavaid programme. Kohe võetakse ette ka väike paarirealine näidisprogramm, millega luuakse selgust, milline saab olla arvuti jaoks mõistetav programm ja millele peab tähelepanu pöörama (näiteks taanded).

Programmide loomisega tutvumise esimeseks sammuks võetakse muutujate väärtustamine. Selle jaoks tutvustatakse põhiliste andmetüüpidena täisarve, ujukomaarve ja sõnesid ning tuuakse välja nendega tehtavad võimalikud operatsioonid. Kuna esimeste näidete puhul on oluline saada nii sisendandmeid kui väljastada tulemusi, et demonstreerida erinevate operatsioonide mõju, siis võetakse kohe ette ka sisendi küsimise ja info väljastamise võimalused, kuigi vaid piisavalt, et oleks alguses võimalik elementaarne vajadus katta.

Info küsimise järel on tihtipeale vaja saadud informatsiooni põhjal teha erinevaid otsuseid ja selle tõttu tuuakse järgmise teemana sisse tingimuslause. Selleks tuuakse sisse järgmiste andmetüüpidena tõeväärtused ja erinevad operatsioonid, mida nendega teha saab. Et saadavat informatsiooni ka endale sobivale kujule viia oskaks, tutvustatakse andmetüüpide omavahelisi teisendusi ja mõningaid lihtsamaid sõnefunktsioone, mis aitavad suurte ja väikeste tähtede vahelisi operatsioone sooritada. Käsitletakse ka suvaliste arvude genereerimist, et otsused saaksid toimuda ka juhuslikult.

Põhjalikult võetakse ette erinevad sõneoperatsioonid ja nende väljastamisviisid, et oleks võimalik väljastada informatsiooni olukorrale sobivale kujul. Graafilisi kasutajaliideseid ei käsitleta, selle asemel tutvutakse pisut mooduliga *turtle*, millega näidatakse lisavõimalusena moodulite importimist ning õpitakse rakendama olemasolevaid funktsioone.

Tsüklitest käsitletakse põhjalikult *while*-tsüklit, põgusalt mainitakse ka *for*-tsükli olemasolu, aga täpsemalt selle mõistmiseks tuleb mujalt ise infot juurde otsida. Tsüklite kasulikkuse demonstreerimiseks pakutakse allikaid, mille abil on võimalik tutvuda levinud labürintide läbimise algoritmidega. Põhirõhk tsüklite käsitlemisel on aga sellel, et saaks garanteerida, et kasutajalt saadav info on sobival kujul.

Erinevalt teistest käsitlusel olnud materjalidest tuuakse kursusel „Programmeerimisest maalähedasest“ sisse ka regulaaravaldised, et oleks võimalik paremini mõista rohkelt kasutusel olevate otsingumootorite tööd. Tuuakse välja võimalusi leida tekstidest vajalikku infot ja

regulaaravaldiste tööpõhimõtete selgitamisega loodetakse mõista anda, kuidas saab võimalikult efektiivselt koostada ka igapäevaselt vajalikke päringuid.

Programmide ratsionaliseerimiseks ja alamülesannete loomiseks ning võimalusel selle kaudu korduvate osade eraldamiseks tuuakse mängu funktsioonid. Lisaks funktsioonide loomisele ja nende rakendamisele tutvutakse parameetrite ja tagastusväärtuste abil võimalustega erinevat tüüpi ülesannete täitmiseks võimalikult otstarbelisel viisil.

Programmi ja kasutaja vahelise suhtluse vältimiseks tutvustatakse lõpuks ka võimalusi lugeda andmeid failidest (nii kettalt kui veebist) ning ka andmete failidesse kirjutamist. Kogu õpitu demonstreerimiseks ja paremini mõistmiseks analüüsitakse eelnevast veidi suuremat programminäidet, kus võimalikult paljusid läbi käinud teemasid ka demonstreeritakse.

Terve kursuse vältel tuuakse välja lugusid olukordadest, mis võiksid panna mõtlema, kuidas on võimalik olukordi programmeerimise abil lihtsustada, ja näiteid kõikvõimalikest tegevustest igapäevaelus, mis töötavad vaid tänu programmeerimisoskusele või mis on tänu sellele palju lihtsamad nagu telefonid, külmkapid, pesumasinad jne.

2.3 Programmeerimise alused I

Kursus „Programmeerimisest alused I“ on e-kursus [7], mille õppematerjalid on samuti avalikult kättesaadavad. Kursuse ülesehitus on suuremalt jaolt sarnane kursusega „Programmeerimisest maalähedaselt“, aga kui seal oli kõigest juttu natuke ja vaid elementaarvajaduste piires, siis „Programmeerimise alused I“ käsitleb teemasid pikemalt ja põhjalikumalt, tuues välja erinevaid levinud olukordi ja muid erijuhte, mille peale esimese hooga võibolla ei mõtle, aga mis sageli esinevad.

Sissejuhatavalt algab kursus eelpooltoodud materjalidega sarnaselt algoritmilise mõtlemise, andmetüüpide ning muutujate tutvustuse ja sisendi küsimisega. Need on praktiliselt kõikides programmeerimise algkursustes alustalad, mida mõistmata on väga raske teiste teemadega edasi minna.

Tingimuslauseid käsitletakse selles kursuses väga põhjalikult. Lisaks üheosalisele tingimusele, mida enamasti alati programmeerimise algkursustes käsitletakse, tuuakse siin eraldi välja kõikvõimalikke variante tingimuslausete kasutamisest: luuakse loogiliste operatsioonide abil mitmeosalisi tingimusi, asetatakse tingimuslauseid teiste tingimuslausete sisse, tuuakse mängu tingimuslausete teised harud (Pythonis *elif* ja *else*). Sellega näidatakse algajale, et õpitud

elemente ei pea kasutama vaid eraldiseisvalt ja suunatakse nad mõtlema kombineerimisvõimalustele.

Sarnaselt kursusega „Programmeerimisest maalähedaselt“ käsitletakse sellel kursusel *while*-tsüklit põhjalikult, ning lisamaterjalidena tuuakse välja tsüklitest saadavast kasust labürintide lahendamisel. Ka on sarnane sõnede ja nendega võimalike operatsioonide käsitlus. Lisana on „Programmeerimise alused I“ kursusel võimalik omandada algteadmisi graafilise liidese loomiseks. Selle jaoks kasutatakse moodulit *tkinter*, kuna tegu on Pythoni standardse mooduliga, mis pakub palju võimalusi ja mida on seeläbi hea lasta huvilistel ise edasi uurida.

Sarnaselt programmeerimise õpikuga [4], käsitletakse rohkemate andmete hoidmise võimalusena järjendit. Sisse tuuakse ka teise alternatiivina *while*-tsüklile *for*-tsükkel ja selle erinevad kasutusviisid, kuna järjenditega töötades on see palju kasutajasõbralikum. Erinevalt eelpooltoodud kursustest kasutatakse kohe tsüklite kasutamise oskuse süvendamiseks ka võimalust neid mõlemaid rakendada failist andmete lugemisel.

Funktsioonide teemal saadavad baasteadmised on sarnased eelnevate materjalidega, vabatahtliku lisamaterjalina tuuakse sisse rekursiivne funktsioon, mida demonstreeritakse graafiliste programmide loomise kaudu, et oleks lihtsam selle olemust mõista ning katsetades teemast arusaamiseni jõuda.

Kursus „Programmeerimise alused I“ käsitleb andmete saamiseks samuti veebist ja kettalt failidest lugemise võimalusi ning ka failidesse kirjutamist. Programmide loomisel kasutamismugavuse tõstmiseks tuuakse sisse veel üks kasutajaliidese loomise vahend – moodul EasyGUI, mis annab lihtsa võimaluse tõsta kasutamismugavust andmete sisestamisel.

Võrreldes kursusega „Programmeerimisest maalähedaselt“ on „Programmeerimise alused I“ materjalides palju rohkem kontrollülesandeid ja lisamaterjale, millega soovi ja huvi korral on võimalik iseseisvalt tutvuda.

2.4 Programmeerimise alused II

Kursus „Programmeerimise alused II“ [8] on loodud jätkukursusena kursusele „Programmeerimise alused I“ [7] ning eeldab, et viimasena nimetatul käsitletud teemad on osalejal omandatud. Erinevalt esimesest kursusest ei sisalda jätkukursuse avalikud materjalid kontrollülesandeid, aga vabatahtlikke lisamaterjale iseseisvaks täpsemaks uurimiseks on praktiliselt igas teemapeatükis ning materjalidesse peidetuna võib leida küsimusi enese

kontrollimiseks. Ühtegi „Programmeerimise alused II“ teemat käesoleva töö raames eelnevalt käsitletud materjalides esinenud ei ole.

Esmalt võetakse uue konstruktsioonina käsitlusele maatriks ehk järjend järjenditest. Selle abil võimaldatakse andmeid näiliselt hoida tabelis ning sellisel kujul olevate väärtuste paremaks käsitlemiseks tuuakse välja kõikvõimalikke erinevaid viise, mis hõlmavad tsükleid erineval kujul, kaasa arvatud teineteise sees.

Kuna andmeid võib vaja olla ka muul kujul, mitte vaid järjendis või tabelis, siis võetakse ette veel mitmeid andmestruktuurid: ennik (ingl k *tuple*), hulk (ingl k *set*), sõnastik (ingl k *dictionary*). Ka sõned võetakse uuesti käsitlusele, aga seekord vaadeldakse neid kui andmestruktuure mitte teksti.

Erinevate andmestruktuuride kasutuselevõtmisega on kasutusele tulnud viittüüpi muutujad ja erinevad andmestruktuurid, siis seletatakse põhjalikult lahti ka viitade olemus ja andmestruktuuride muteerimine. Programmeerimise algkursustes seda pahatihti nii detailselt ei käsitleta, mistõttu on erinevate andmestruktuuridega korraga tutvudes enamasti küllaltki raske orienteeruda ja neid üksteisest eristada.

Oma koodi korrektsuse ja töökindluse kontrollimiseks tutvustatakse põgusalt testjuhtude loomise ja programmi testimise olulisust. Enamasti sellisel algsel tasemel testimise vajalikkust eraldi välja ei tooda, ning tegeletakse sellega eraldi vaid testimisele pühendatud kursustes ja materjalides.

Uue suure teemana tuuakse sisse ka rekursioon, mille mõistmisele kaasa aitamiseks tuuakse põhiliselt paralleele tsüklitega. Käsitlusel on põhiliselt lineaarsed rekursiivsed funktsioonid, kuigi põgusalt tutvustatakse ka rekursiivse hargnemise põhimõtteid.

2.5 TÜ kursus „Programmeerimine“ (LTAT.03.001)

Tartu Ülikooli kursus „Programmeerimine“ (LTAT.03.001) [9] on teemade valiku poolest väga sarnane „Programmeerimise alused I“ ja „Programmeerimise alused II“ teemadega, kuigi teemasid käsitletakse teistsuguses järjekorras ja enamasti põhjalikumalt. Materjal toetub Aivar Annamaa õpikule [10], mis on avalikult kättesaadav.

Sissejuhatavalt demonstreeritakse sisendi küsimist ja info väljastamist, arvutamist ning matemaatiliste funktsioonide importimist ja kasutamist. Lisaks matemaatika moodulile

tutvustatakse kohe ka kilpkonna moodulit (*turtle*). Tegu on eelnevalt käsitletud kursustega võrreldes teistsuguse lähenemisega, sest esmalt antakse ette mõned baasvõimalused ja näited nende kasutamisest, julgustatakse ja suunatakse ise neid võimalusi katsetama ja edasi arendama, ning alles seejärel võetakse ette täpsem teemakäsitlus.

Sarnaselt eelnevalt käsitletud kursustele tutvustatakse alguses põhjalikult erinevaid andmetüüpe ja nende võimalusi, aga lisaks tuuakse sisse lausete ja avaldiste mõisted ja nende erinevused. Samuti võetakse kohe alguses ette põhjalikumalt sisendi ja väljundi võimalused, mis erinevalt eelmistest käsitletud kursustest annab materjali läbijale kohe võimaluse harjutada ennast informatiivselt väljendama.

Veel ühe väga suure erinevusena on A. Annamaa õpikus [10] kohe alguses käsitlusel failide kettalt ja veebist lugemine ning failidesse kirjutamine, mis annab teiste kursuste ees eelise, kus on kohe alguses võimalik sisendina käsitleda rohkem infot, sest katsetamise käigus peab käsitsi sisestama vähem infot.

Tingimuslausete käsitlus on põhjalik, lisaks võimalikele harudele (*if, elif, else*) võetakse ette loogiliste avaldiste ühendamine ning selle raames käsitletakse ka tehete järjekorda. Siinkohal tuuakse sisse tõeväärtused ning käsitletakse ka *while*-tsüklit, tuues paralleele tsüklite ja tingimuslausete vahel.

Funktsioonide käsitlus on küllaltki sarnane eelnevatega, põhilise erinevusena tuuakse siinkohal sisse uue teemana erindite tekitamine ja püüdmine, mida eelnevates materjalides üldse ei käsitletud. Lisaks on käsitlusel globaalsed muutujad ja nende kasutusviisid, kuigi need soovitatakse asendada funktsiooni parameetritega.

TÜ kursus „Programmeerimine“ tutvustab tavaelu tegevuste teisendamist algoritmideks tükk maad hiljem kui eespool käsitletud kursused ning tegu on pigem põgusa kõrvalepõikega. Suurem rõhk on järjenditel, mida käsitletakse väga sarnaselt varasemalt välja toodud materjalides, kuigi tükk maad põhjalikumalt. Erinevusena tuuakse sisse ka ennikud (*tuple*), kuna tegu on järjenditega küllaltki sarnase andmestruktuuriga. Põhjalikult võrreldakse ka *for*-tsükli ja *while*-tsükli kasutusviise.

Lisaks ennikutele võetakse kasutusele veel mitmed erinevad andmestruktuurid. Käsitletakse hulkasid (*set*) ja sõnastikke (*dictionary*), samuti mitmemõõtmelisi andmestruktuure, põhiliselt siiski maatrikseid ehk kahekordseid järjendeid.

Viimase suure teemana on käsitlusel rekursioon. Põhirõhk on lineaarsel rekursioonil, põgusalt võetakse ette ka rekursiivselt hargnevad funktsioonid. Käsitletakse ka järjendite ning mitmemõõtmeliste andmestruktuuride läbimist rekursiivselt.

Terve kursuse raames on materjali vahel mitmeid kontrollküsimusi, näidisülesandeid lahendusvihjetega ning soovitusi erinevate ülesannete lahendamiseks. Lisamaterjalidena on välja toodud mitmeid levinud mooduleid, mida iseseisvalt uurida ja mille tundmisest võiks potentsiaalselt rohkem kasu olla.

3. Metoodika

Selles peatükis keskendutakse materjalide kavandamise ja loomise protsessile, mis sisaldab kahte osa. Esimeses osas tutvustatakse teemade valikuni jõudmist, mida antud eksamil küsitakse ja selle põhjal eksami vormi kujundamist. Teises osas tutvustatakse teemade põhjal näidise loomist, millega analoogsed ülesanded reaalselt eksamil lahendamiseks tulevad, ja selle käigus toimunud muutusi töö iseloomus.

3.1 Teemade valik

Ülesannete koostamisele eelnevalt tutvuti mitmete Eestis leiduvate kursustega, mis õpetavad programmeerimist keeles Python (ülevaate peamistest saab eelnevas peatükis). Analüüsiti materjale, mida kasutatakse erinevates gümnaasiumides programmeerimise õpetamisel nagu Eno Tõnissoni jt loodud materjali „Programmeerimine“ [4] ning tutvuti teaduskooli kursustega, mis samuti programmeerimise alustalasid õpetavad nagu „Programmeerimisest maalähedaselt“ [11], „Programmeerimise alused“ [12], „Programmeerimise alused II“ [13]. Analüüsi käigus töötati materjalid läbi, kaardistati käsitletavad teemad ja nende põhjalikkus ning võrreldi kursuseid omavahel.

Eelpoolmainitud kursuste analüüsi põhjal sai välja tuua nimekirja teemadest, mida klassikaliselt algajad esimesena õpivad. Täiendusena lisandusid nimekirja ka teemad, mida käsitletakse Tartu Ülikooli kursusel „Programmeerimine (LTAT.03.001)“, kuna eksami sooritamine vähemalt 90% tulemusega tagab lisaks pääsule informaatika ja arvutitehnika õppekavadele ka arvestuse nimetatud aines. Vastavalt teemapunkti keerukusele ja sõltuvalt sellest, kas teema sai nimekirja eelpoolmainitud kursuse tõttu, jagunesid teemad kaheks – 50% lävendi ületamiseks vajalikud teemad ja ülejäänud teemad (lisa I).

Esialgse ülesannete kogu mustandi teostamiseks leidsid kasutust varasemalt kursusel „Programmeerimine“ olnud kontrolltööde ja eksamite ülesanded. Nende põhjal kujunes esialgne pilt, milline eksam potentsiaalselt välja võiks näha, kuna teemad suures osas kattuvad ja eksami ülesehitus on sarnane nimetatud kursusel läbiviidavate tööde ja eksamiga. Reaalseks eksamiks valmisid täiesti uued ülesanded, aga olemasoleva materjali põhjal kujunes välja ülesande püstituse vorm ja kuju, mis on reaalselt pidevalt kasutuses olnud ja seetõttu sooritajale võimalikult arusaadavale kujule viidud. Vormi ja kuju all on mõeldud lisaks tekstilisele

püstitusele ka näited programmi kasutusest ning näidisülesannete puhul ka näidislahendusi ja hindamisjuhendeid.

Koostamise käigus ja juhendajaga nõu pidades kujunes arvutiosasse kaks programmeerimist nõudvat ülesannet – üks, kus on lihtsad ja elementaarsed teemad 50% lävendist, ja teine, mis sisaldab valikut ülejäänud teemadest. Teemade jaotust vaata lisa I.

3.2 Näidiseksami koostamine

Programmeerimise eksami läbiviimiseks oli vaja, et gümnaasiumite programmeerimise õpetajad saaksid sellest võimalusest varakult teada, et oleks võimalik õpilastele antud võimalust tutvustada. Seega tuli luua juhend, et oleks selge, mis tingimustel ja kuidas antud eksam aset leiab, ning varustada neid näidiskomplektiga, et oleks selge ettekujutus eksami olemusest, kuna konkreetne eksam toimus esmakordselt. Näidiskomplekte valmis kaks – üks, mis sisaldas ülesannete püsitusi, näidislahendusi ja hindamisjuhendit, ning teine, milles olid vaid ülesannete püstitused.

Eksami näidiskomplektide loomisel leidsid kasutust mitmed ülesanded esialgsest ülesannete kogust, mis valmis pärast eksamil käsitlevate teemade selgumist eesmärgil kujundada selgem pilt eksami olemusest, kasutades algmaterjalina TÜ kursusel „Programmeerimine“ toimunud töid. Lisaks valmisid ülesanded koostatud teemade nimekirjast valitud teemadel, mis enne piisavalt kajastust ei leidnud. Suurem osa ülesandeid esialgsest näidiskomplektist jäid välja, kuna sealne teemade valik ei olnud niivõrd tasakaalukalt käsitletud kui vaja ning ülesannete soovitud mahukus ei olnud samaväärne olemasolevatega.

Esialgne plaan oli luua paberil lahendamiseks kümme võrdselt punkte andvat ülesannet, mis käsitlevad erinevaid teemasid. Suuremas osas oli plaanis luua lühikesi koodijuppe, mille juures on kirjas koodijupi eesmärk ning lahendaja ülesanne oleks täita lüngad, et kood töötaks nii nagu soovitud. Kuna aga vastav ülesehitus nõuab, et sooritaja teaks peast mitmeid erinevaid fikseeritud kirjapilte olukorras, kus tal pole võimalik nende õigsust kuidagi katsetada või kontrollida, siis arutelu käigus sündis otsus rakendada paberil vaid ülesandeid, kus lahendajatele antakse ette valmis koodijupp erinevate algandmetega ja lahendaja peab suutma analüüsida ning tuvastada, mida programm lahendamise käigus väljastab (läbimänguülesanded). Otsusele aitas kaasa teadmine, et arvutis lahendatavates ülesannetes

kontrollitakse eksami sooritajate oskust ise luua töötavat koodi. Ajalistel kaalutlustel vähenes ülesannete arv kümnelt viiele ja iga ülesande antav punktide arv kahekordistus.

Näidiseksami iga paberosa ülesande juurde sai informatiivsel eesmärgil lisatud ka teema, mida antud ülesanne põhiliselt kontrollib ning võimalik saadav punktisumma. Lisaks oli olemas iga ülesande juures sooritajalt eeldatav vastus. Pärast paberosa ülesehituse muutust valmis uus näidis, millele vastuseid juurde ei lisatud, kuna seda oli võimalik harjutamise käigus kõigil ise kontrollida. Lahendusteta näidiskomplekti ja uut paberosa näidis vaata lisadest (lisa II, lisa III).

Arvutis lahendatava osa puhul oli vaja kindlustada, et esimese ploki ehk 50% teemade oskamisel on võimalik eksami sooritajal saada kätte positiivne tulemus, ning et kõiki teemasid valdamata ei oleks võimalik saada eksamil vähemalt 90% sooritust, mis tagaks arvestuse aines „Programmeerimine“. Seetõttu oli arvutiosas kaks võrdselt punkte andvat programmeerimisülesannet, rohkem ülesandeid ei lisatud ajalistel kaalutlustel. Samamoodi paberosa esialgsete näidistega oli ka paberosas kaks näidiskomplekti. Mõlemas oli iga ülesande juures püstitus ja maksimaalne võimalik saadav punktisumma, aga ühes komplektis olid ka näidislahendused ning hindamisjuhend, kus oli täpsemalt märgitud, mitu punkti mingi osa ülesandest annab. Kuna arvutiosa ülesanded on suuremad ja kajastavad mitmeid teemasid, siis seal eraldi teemasid välja ei toodud.

4. Tulemused

Selles peatükis keskendutakse läbiviidud eksamile. Esimeses osas tutvustatakse lõpptulemusena valminud eksamiülesannete komplekti, teises osas analüüsitakse eksami sooritajate tulemusi ja tagasisidet ning kolmandas osas arutletakse, mida saaks tulevikus paremini teha.

4.1 Eksamiülesannete komplekt

Lõplike näidisülesannete ja nende hindamisjuhendite põhjal valmisid täiesti uute ülesannetega kaks analoogset ülesannete komplekti (vaata lisa IV ja lisa V), mis läksid kasutusse 25. märtsil 2022 Tartus ja 26. märtsil Tallinnas toimunud eksamil, kus soovijad said võimaluse kontrollitud tingimustes kolme tunni jooksul ülesandeid lahendada.

Mõlema valminud komplekti paberosas oli viis ülesannet, iga ülesanne sisaldas koodijuppi ja ootas lahendajalt vastuseks väljundit, mis koodijupi käivitamisel saadakse (koodijupi näidis joonis 1). Iga koodijupp väljastas mitu väärtust või küsis väljundit mitme sisendi korral. Igal ülesandel oli juurde märgitud teema, mida põhiliselt vastav ülesanne kontrollis, mõlemas komplektis olid teemadeks tingimuslause, järjend, funktsioon ning kahemõõtmeline järjend koos kahekordse tsükliga. Viimase ülesande teema oli komplektides erinev – ühes komplektis oli rekursioon ja teises sõnastik koos funktsiooniga.

```
def liida(arv1, arv2):
    return arv1 + arv2

a = 3
b = 4
print(liida(a, b))
print("---")

print(liida(liida(a, b), b))
print("---")

while a < 15:
    print(a)
    a = liida(a, liida(a, b))
```

Joonis 1. Koodijupp eksami paberosa ülesandest (lisa IV).

Mõlema komplekti arvutiosa koosnes kahest programmeerimisülesandest. Neid ülesandeid lahendades oli lubatud kasutada internetis leiduvaid materjale ja enda varasemaid töid (suhtlemine teistega ei olnud lubatud). Lisaks ülesande lahendusele tuli esitada ka lahendamisaega kajastav logi, et oleks võimalik mingil määral kontrollida, et lahenduseni on jõutud iseseisvalt, mitte kopeerimise teel, mistõttu paluti lahendajatel kasutada keskkonda Thonny.

Arvutiosa esimene ülesanne oli lihtsam programmeerimise ülesanne, kus mõlema komplekti puhul oli ette antud fail mingit kindlat sorti ühesuguste andmetega, ehk igal faili real oli üksainus sõna või arv, terve faili vältel ühtemoodi (joonis 2).

```
maasikad
mustikad
mustikad
vaarikad
maasikad
vaarikad
mustikad
```

Joonis 2. Faili sisu eksami arvutiosa esimesest ülesandest (lisa V).

Lahendaja pidi kirjutama ühe funktsiooni, mis saab parameetriks kaks arvu ning leiab, mitu protsenti esimene teisest moodustab. Samuti tuli lugeda kokku failis olnud andmete esinemissagedus ja andmete kogus, mille põhjal iga erineva kategooria kohta tuli väljastada esinemise arv ja selle osakaal koguhulgast viisakal kujul. Tartus kasutusele läinud komplekti puhul oli tegu vanuste jaotamine klassidesse etteantud tingimuste põhjal ning Tallinnas kasutatud komplektis tuli vaadata, kui palju marjadest moosi tehti.

Teine ülesanne arvutiosas oli suurem programmeerimisülesanne, mistõttu oli eksamikomplektide vahel selle ülesande puhul suuremaid erinevusi kui esimese ülesande puhul. Suuremad erinevused olid tingitud eksami toimumisest Tartus ja Tallinnas erinevatel päevadel. Siiski oli mõlema ülesande puhul vaja lugeda failist algandmed nõutud andmestruktuuri (sõnastik ja maatriks) ning nendes võimaldada kasutajal teha erinevaid tegevusi kuni kasutaja valib programmi sulgemise. Selleks oli lahendajale kirjeldatud iga võimaliku operatsiooni jaoks funktsioon, kus olid antud ette parameetrid, nõutav tagastusväärtus ja mida funktsioon tegema peaks. Samuti tuli luua põhiprogramm, mis kasutajal operatsioone sooritada laseb kuni valitakse programmi sulgemine, mille puhul salvestatakse andmed esialgse failiga samal kujul uude faili ja lõpetatakse töö.

Tartus kasutusele läinud eksamiülesannete komplekti puhul oli ülesande sisuks puslekataloog, kus etteantud süsteemi alusel tuli erinevate suurustega puslesid hoiustada, välja laenutada ja tagastada, kasutajale kuvada ning kokku lugeda, luues iga operatsiooni jaoks eraldi funktsiooni (näide joonis 3). Lisaks funktsiooni kirjeldusele (parameetrid, eesmärk, tagastusväärtus) oli igal funktsioonil selgitavalt juures ka näide funktsiooni tööst. Algandmed olid kirjutatud faili, mille lugemine nõudis lahendajalt erinevalt esimesest ülesandest sõnetöötluse oskust (faili sisu näide joonis 4). Samuti tuli osata kataloogi ka ise samas formaadis faili kirjutada. Luua tuli ka põhiprogramm, mis võimaldab kasutajal lõputult erinevaid operatsioone sooritada, kuni valitakse programmi sulgemine.

1. Kirjuta funktsioon `loe_kollektsioon`, mis võtab parameetriks faili nime ja tagastab sõnastiku, mille võtmeks on pusle identifitseerimise kombinatsioon ja väärtuseks isik, kelle käes vastav pusle on. Kui seda puslet ei eksisteeri (failis on selle koha peal "-"), siis vastavat kirjet sõnastikku ei lisata.

Joonis 3. Funktsiooni kirjeldus eksami (Tartu) arvutiosa teisest ülesandest (lisa IV).

```
  1 2 3 4
A * * * Mari
B * - - -
C * * Kalle -
```

Joonis 4. Faili sisu eksami (Tartu) arvutiosa teisest ülesandest (lisa IV).

Tallinnas kasutusele läinud eksamiülesannete komplekti puhul oli ülesande sisuks aiasaaduste müümine, kus taaskord etteantud süsteemi alusel tuli aiasaaduseid erinevates kogustes ja erinevate hindadega hoiustada, juurde lisada, müüa ehk eemaldada ning koguhinda arvutada, luues ka siin iga operatsiooni jaoks eraldi funktsiooni. Funktsioonide esitus ja etteantud näited olid samal kujul Tartus toimunud eksamiga (joonis 3). Ka see ülesanne nõudis algandmete failist lugemisel ning hiljem samal kujul uude faili kirjutamisel sõnetöötluse oskust (faili sisu näide joonis 5). Samuti tuli luua põhiprogramm operatsioonide sooritamiseks kuni programmi sulgemiseni (joonis 6).

```
kurk 5 0.75
tomat 7 1.25
kapsas 3 0.5
```

Joonis 5. Faili sisu eksami (Tallinn) arvutiosa teisest ülesandest (lisa V).

7. Koosta põhiprogramm, kus saab valida erinevate tegevuste vahel. Programm salvestab esmalt aiasaaduste seisu funktsiooni `loe_saadused` abil maatriksisse. Pärast igat tegevust küsib programm uuesti, mida kasutaja soovib teha, kuni ta valib programmi töö lõpetamise. Kui kasutaja sisestab

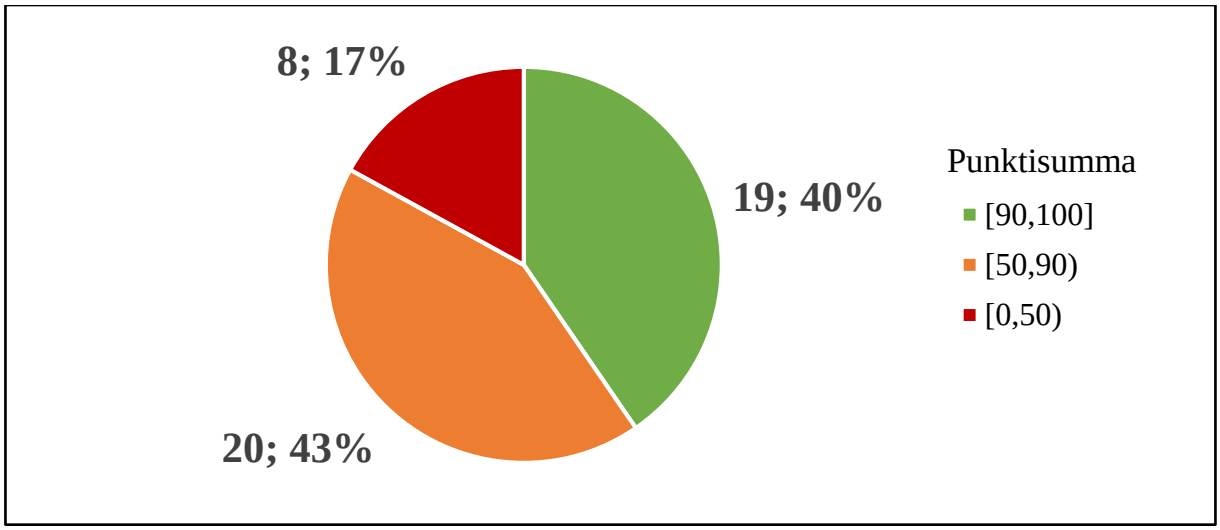
- "1", kuvab programm aiasaaduste kataloogi funktsiooni `vaata_saaduseid` abil,
- "2", lisab kataloogi uued aiasaadused funktsiooni `lisa_saadus` abil,
- "3", eemaldab kataloogist mingi arvu aiasaaduseid funktsiooni `müü_saadus` abil,
- "4", leiab ja väljastab müüdavate aiasaaduste väärtuse funktsiooni `kogu_väärtus` abil,
- "5", teeb programm järgmised sammud:
    - salvestab aiasaaduste maatriksi seisu faili *tooted_uus.txt* funktsiooni `kirjuta_faili` abil,
    - väljastab lõpetamise kohta teate,
    - lõpetab töö.

Joonis 6. Põhiprogrammi püstitus eksami (Tallinn) arvutiosa teisest ülesandest (lisa V).

4.2 Tulemused ja tagasiside

Eksam viidi läbi 25. märtsil 2022 Tartus ja 26. märtsil 2022 Tallinnas, kus soovijad said võimaluse kontrollitud tingimustes kolme tunni jooksul ülesandeid lahendada. Kokku oli registreerunuid 53 (Tartus 44 ja Tallinnas 9), kohale tuli 49 sooritajat. Nendest 49 osalejast kaks rikkusid eksamil kehtestatud nõudeid ja nende eksamitulemus arvesse ei läinud, seetõttu käsitleme edaspidi sooritajatena 47 inimest.

Eksami 47st sooritajast 37 ületas nii paberil kui arvutis sooritatavas osas 50%, mis kindlustas koha informaatika ja arvutitehnika erialadel. Ülejäänud 10 osalejast seitse suutis siiski ühes osas ületada 50% piiri, kuigi eksami sooritamiseks sellest ei piisanud. Neid, kes suutsid saada vähemalt 90% tulemuse ja sellega saada TÜ aines „Programmeerimine" (LTAT.03.001) arvestuse oli 37 läbija seas 19 ehk ligikaudu pooled (punktijaotus sektordiagrammil joonis 7).



Joonis 7. Eksami tulemuste jaotus sektordiagrammil.

Selline eksami sooritajate tulemuste jaotumine on üsna ootuspärane arvestades eksami raskusastet. Eksamit tulid sooritama põhiliselt need, kellel oli suur potentsiaal eksamist läbi saada ja ligikaudu 4/5 seda ka suutsid. Kõigist tegijatest oli neid, kes said kätte 90% umbes 2/5, mis tähendab, et eksami raskusaste on enam-vähem optimaalne, kuigi võib olla saaks raskemasse arvutiosa ülesandesse lisada veel mingeid nüansse, mis keerukust veidi tõstaks.

Eksamil osalenutest 29 vastasid tagasisideküsitlusele ning praktiliselt nende kõigi hinnangul on selline võimalus ülikoolikoha tagamiseks väga oluline. Pooled vastanutest olid õppinud programmeerimist kas koolis või mõnel e-kursusel ning peaaegu kõik (27 vastanut) tegelesid lisaks ka iseseisva õppimisega. Valmistumisel olid suurimaks abiks näidisülesanded ja väljatoodud teemad, mida eksam kajastas – päris mitu sai tänu sellele teha enne eksamit endale selgeks ka selle üksiku teema, mille osas varasemalt teadmised puudusid. Ligikaudu 90% vastanutest leidis, et eksamil oleva sisu kohta oli olemas piisavalt infot ja materjali, et selleks hästi valmis olla, probleemina tõsteti üles korralduslik punkt – info, millal ja kus registreerimine avatakse, oli puudulik (selle poolega tegeles Tartu Ülikool).

Ülesannete arusaadavus oli tagasiside põhjal hea, aga leiti ka kaks punkti, kus ülesande tekstis ja näidetes kaks erinevat kohta omavahel täielikult kokku ei läinud – need muudatused tekkisid sellest, et eksami lähenedes jäi suhtlusesse auke, mille käigus teatud muudatustega ei hoidnud ülesandega tegelejad (töö autor ja juhendaja) teineteist kursis, mistõttu jäid kriitilised kohad üle kontrollimata. See ei olnud aga nii suur probleem, et poleks aru saada, mida soovitakse ning mõlemad juhud, mis ülesandes esinesid, loeti õigeks. Positiivsena toodi välja, et kõige keerulisemate kohtade peal olid juures vihjed, mis aitasid välja mõelda lahendusi, mille peale algajatel programmeerijatel on raske kiiresti tulla.

Eksami juures raskust valmistavaks elemendiks hindasid pooled vastajad arvutiosa teise ülesande ja veerand paberosa, ülejäänute hinnangul midagi raskusi valmistavat eksam ei sisaldanud.

4.3 Võimalikud parandused edaspidi

Loodud eksamiülesannete komplektide puhul oli tagasiside ja tulemused pigem positiivsed. Loodud komplektide paberosa viienda ülesande puhul läksid kaks teemat jagamisele – rekursioon ja sõnastik funktsiooniga. Käesolevas versioonis oli otsus, et mõlemasse komplekti jääb viis ülesannet, kuna enne esimest läbiviimist ei olnud selge, kui palju võiks paberil programmijuppide läbi mängimine aega võtta ja oli kartus, et tekib liigne ajapuudus. Siiski oli eksami läbiviimise käigus näha, et paberosa ei valmistanud sooritajatele eriti suurt probleemi, mistõttu võiks edaspidi rakendada mõlemat teemat igas komplektis.

Samuti võiks edaspidi luua ülesannete parandamiseks teistsuguse hindamisjuhendi – hindemaatriksi. Praegu oli hindamisjuhend lihtsalt välja kirjutatud alampunktidena ja vastavate punktiskooridega, hindemaatriks võimaldaks aga omada paremat ülevaadet hindamise käigus ning väldiks pidevat edasi-tagasi kerimist õige punkti leidmiseks.

Eksamikomplekti loomine ja eksami läbiviimine antud töö raames oli edukas – ka edaspidi on plaanis selletaolist eksamit korraldada võimalusena Tartu Ülikool astumiseks. See annab gümnaasiumitele põhjust ja motivatsiooni õpetada programmeerimist sel tasemel, et õpilased oleksid võimelised soovi korral eksamit edukalt sooritama. Lisaks on läbiviidud eksam andnud eeskuju teistele erialadele, kes samuti sisseastumiseks lisavõimaluste loomise suunas vaatavad.

5. Kokkuvõte

Töö eesmärk oli luua uue võimalusena Tartu Ülikooli informaatika ja arvutitehnika õppekavadele sisse astumiseks programmeerimise eksamiülesannete komplekt ning seda eksami läbiviimisel katsetada. Lisaks nimetatud õppekavadele õppekoha garanteerimisele vähemalt 50% tulemusega tagas vähemalt 90% tulemus arvestuse TÜ aines „Programmeerimine“ (LTAT.03.001).

Eksami vormi väljatöötamiseks analüüsiti erinevaid TÜ poolt pakutavaid programmeerimise algkursuseid ja neis käsitletavaid teemasid. Uuest eksamist ülevaate saamiseks valmisid ülesannete näidiskomplektid nii lahenduste ja hindamisjuhendiga kui ilma.

Lõpptulemusena valmis kaks uut eksamiülesannete komplekti, mõlemas komplektis kaks osa. Esimene osa sisaldas viit ülesannet, mida tuli paberil lahendada ja mis põhinesid lühikese koodijupi läbimängimisel. Teine osa sisaldas kahte erineva raskusastmega programmeerimise ülesannet. Esimese ülesandes tuli osata lihtsal kujul andmeid failist lugeda ja funktsioone defineerida, teises ülesandes kontrolliti laiemat oskuste ringi.

Eksam viidi läbi nii kahel järjestikusel päeval nii Tartus kui Tallinnas, kokku käis eksamit sooritamas 49 isikut, kellest 37 tagasid endale vähemalt 50% sooritusega õppekoha Tartu Ülikooli informaatika ja arvutitehnika erialadel. Ligikaudu pooled neist (ehk 19 isikut) sooritasid eksami vähemalt 90% tulemusega.

Suur osa eksami sooritajatest vastasid tagasisideküsitlusele, millest saadud tagasiside oli vägagi positiivne. Tagasiside ja eneseanalüüsi põhjal valmis paar soovitust, mida edaspidi eksami ülesannete komplekti loomisel ja eksami läbiviimisel arvesse võtta.

Viidatud kirjandus

[1] Tartu Ülikool https://ut.ee/et/sisu/eritingimusel-kandideerimine (06. 05. 2022)

[2] Tartu Ülikool https://ut.ee/et/oppekavad/informaatika (06. 05. 2022)

[3] Schillinger, F. L., Mosbacher, J. A., Brunner, C., Vogel, S. E., & Grabner, R. H. (2021). Revisiting the Role of Worries in Explaining the Link Between Test Anxiety and Test Performance. *Educational Psychology Review*, *33*(4), 1887–1906. https://doi.org/10.1007/s10648-021-09601-0 , p 1888

[4] Tõnisson E, Palts T, Säde M, Tõnisson K jt. Programmeerimine. https://web.htk.tlu.ee-/digitaru/programmeerimine/ (10. 12. 2021)

[5] HITSA https://www.hitsa.ee/ikt-haridus/progetiiger/gumnaasiumi-informaatika-ainekava-/opetajale (06. 05. 2022)

[6] Tartu Ülikooli arvutiteaduse instituudi programmeerimise õpetamise töörühm. Programmeerimisest maalähedaselt. https://courses.cs.ut.ee/2018/progmaa/fall/Main/HomePage (06. 05. 2022)

[7] Tartu Ülikooli arvutiteaduse instituudi programmeerimise õpetamise töörühm. Programmeerimise alused. https://courses.cs.ut.ee/2018/eprogalused-/fall/Main/Programm (06. 05. 2022)

[8] Tartu Ülikooli arvutiteaduse instituudi programmeerimise õpetamise töörühm. Programmeerimise alused II. https://courses.cs.ut.ee/2018/eprogalused2/spring/ (06. 05. 2022)

[9] Tartu Ülikool. Programmeerimine. https://courses.cs.ut.ee/2021/programmeerimine/fall (06. 05. 2022)

[10] TÜ Arvutiteaduse instituudi programmeerimise algkursuse õpik http://progeopik.cs.ut.ee/ (06. 05. 2022)

[11] Tõnisson E, Lepp E, Palts T, Suviste R, Velner M L, Jaansalu M-L, Talimaa H, Säde M, Papli K, Vaherpuu V, Reponen H-L, Rõmmel K, Hollo K. Programmeerimisest maalähedaselt. https://www.ut.ee/et/mooc/programmeerimisest-maalahedaselt (10. 12. 2021)

[12] Tõnisson E, Lepp E, Palts T, Suviste R, Jaansalu M-L, Säde M, Papli K, Vaherpuu V, Hollo K, Reponen H-L. Programmeerimise alused. https://www.ut.ee/et/mooc/programmeerimise-alused (10. 12. 2021)

[13] Tõnisson E, Lepp E, Palts T, Suviste R, Jaansalu M-L, Säde M, Papli K, Vaherpuu V, Hollo K. Programmeerimise alused II. https://www.ut.ee/et/mooc/programmeerimise-alused-ii (10. 12. 2021)

Lisad

I Teemaplokkide jaotus

Esimene teemade plokk ehk 50% teemad märksõnadena:

1. sisend-väljund
2. andmetüübid
3. tehted erinevate andmetüüpidega
4. sõnetöötlus
5. failid (lugemine ja kirjutamine)
6. tingimuslause
7. tsükkel
8. funktsioon
9. järjend

Teine teemade plokk märksõnadena:

1. ennik
2. sõnastik
3. hulk
4. maatriks ehk kahemõõtmeline järjend
5. kahekordne tsükkel
6. lineaarne rekursioon

II Eksamikomplekti näidis

Programmeerimise eksami paberosa näidis

Eksami sooritamiseks tuleb paberosa sooritada vähemalt 50% tulemusele.

Ülesanne 1. Tehted muutujatega

Mis on muutuja x väärtus pärast programmi täitmist,

```
x -= 3
if x % 2 == 0:
    x *= 2
    if x > 15:
        x = x - 1
    else:
        x = x + 2
```

kui x algväärtus on

a) 9

b) 10

c) 11

d) 12

Ülesanne 2. Tsükkel

Kirjuta programmilõik, mille täitmise järel on korrutise väärtuseks kõigi täisarvude korrutis 2-st 7-ni.

```
korrutis = ____
___ = 2
while _______:
    korrutis ________
    ______
```

Ülesanne 3. Funktsioon

Kirjuta funktsioon `summa`, mis etteantud järjendi a puhul tagastab selle järjendi elementide summa.

```
def _____________:
    k = ___
    for ___________:
        k = __________
    ___________
```

Ülesanne 4. Järjend

Mis arvud väljastatakse ekraanile?

```
li = [9, 8, 7, 6, 5, 4, 3, 2, 1]
li2 = []
for i in range(len(li)):
    if li[i] < 3 or li[i] >= 6:
        li2.append(li[i])
for el in li2:
    print(el)
```

Ülesanne 5. Failid, kahekordne tsükkel

Mille trükib välja programm,

```
f = open("andmed.txt")
for rida in f:
    rida = rida.strip("\n")
    rida = rida.split(" ")
    for i in range(len(rida)):
        print(i)
f.close()
```

kui faili *andmed.txt* sisu on

a)

```
1 2
3
```

b)

```
1 2 3
1 2 3
1 2 3
```

c)

```
1
3
```

d)

```
2 3 4
1 2
```

Ülesanne 6. Sõned

Too näide sõne s väärtusest, mille korral trükib programm välja.

```
if 'xy' in s:
    print("tekst1")
else:
    print("tekst2")
```

a) "tekst1"
b) "tekst2"

Ülesanne 7. Hulk

Mitu elementi on hulgas pärast järgmise programmilõigu täitmist?

```
hulk = {1, 2, 3, 4}
hulk.add(3)
hulk.add(5)
hulk.remove(2)
```

Ülesanne 8. Maatriks

Järgnev programm peaks leidma kahemõõtmelises järjendis olevate kõikide elementide arvu. Täida lüngad.

```
li = [[1, 2, 3], [4, 5, 6, 7], [8, 9, 10]]
summa = ___
for _______:
    for _______:
    summa += ____
>>> summa
10
```

Ülesanne 9. Sõnastik

Olgu meil muutujasse salvestatud sõnastik järgmiselt:

```
sõnastik = {'a': 1, 'b': 2, 'c': 3}
>>> f(sõnastik)
a 1
b 2
c 3
```

Kirjuta funktsioon, mis trükib välja eraldi ridadele sõnastiku võtmed ja nende väärtused (elementide järjekord pole oluline)

```
def f:
    for _________________:
        print(_______________)
```

Ülesanne 10. Rekursioon

Kirjuta rekursiivne funktsioon fib(n), mis leiab n-nda Fibonacci jada arvu järgmisel viisil:

- n = 1 korral tagastab 0;
- n = 2 korral tagastab 1;

muul juhul aga leiab rekursiivselt eelmise kahe fibonacci arvu summa.

```
def fib(n):
    if _________:
        _________
    elif ________:
        _________
    else:
        _____________
```

Programmeerimise eksami arvutiosa näidis

Eksami sooritamiseks tuleb arvutiosa sooritada vähemalt 50% tulemusele. Lahendada tuleb keskkonnas Thonny. Lisaks ülesannetele tuleb esitada ka lahendamise käigus tekkinud logid.

Ülesanne 1 (30p)

Saja Aakri metsa elaniku karupoeg Puhhi fännidest metsaomanikud on oma metsatükkide pindalad kirjutanud aakrites (1 aaker = 0,4047 hektarit). Igal omanikul on ainult ühe puuliigi metsad. Konkreetsete puuliikide puhul on teada aastane metsa juurdekasv hektari kohta tihumeetrites (tm/ha). Näiteks kase puhul võib see olla 4,8 tm/ha, kuuse puhul 6,6 tm/ha, männi puhul 3,7 tm/ha. Omanik tahab teada, mitu tihumeetrit metsa aastas teatud suurusest suuremates metsatükkides juurde kasvab. **Näide *andmed.txt* sisust:**

```
0.9
3.78
2.05
1.58
```

Kirjuta funktsioon `juurdekasv`, mis

- võtab argumentideks metsatüki pindala (ujukomaarv aakrites) ja metsa aastase juurdekasvu hektari kohta (ujukomaarv),
- tagastab selle pindalaga metsatüki aastase juurdekasvu ümardatuna sajandikeni.

Näide funktsiooni `juurdekasv` tööst:

```
>>> juurdekasv(3.78, 6.6)
10.1
```

Kirjuta programm, mis

- küsib kasutajalt
  - failinime (failis on eraldi ridadel metsatükkide pindalad aakrites);
  - vastava puuliigi aastase juurdekasvu hektari kohta tihumeetrites (ujukomaarv);
  - piiri, mitmest aakrist suuremas metsatükid arvesse võtta (ujukomaarv);
- loeb failist metsatükkide pindalad;
- arvutab (funktsiooni `juurdekasv` abil) ja väljastab metsatüki aastase juurdekasvu, kui selle metsatüki pindala on sisestatud piirist suurem;
- väljastab teate "Metsatükki ei võeta arvesse", kui metsatüki pindala ei ole sisestatud piirist suurem;
- väljastab lõpuks ekraanile, mitme metsatüki juurdekasv arvutati.

Näide programmi võimalikust tööst faili *andmed.txt* korral (kasutaja sisend on paksus kirjas)

```
Sisestage failinimi: andmed.txt
Sisestage aastane juurdekasv hektari kohta tihumeetrites: 6.6
Sisestage piir, mitmest aakrist suuremad metsatükid arvesse võtta: 2
Metsatüki aastane juurdekasv on 10.1
Metsatüki aastane juurdekasv on 5.48
Metsatükki ei võeta arvesse
Arvutati 2 metsatüki juurdekasv
```

Ülesanne 2 (30p)

Koosta programm turniiri läbiviimiseks. Osalejate nimed on failis turniir.txt. Esimesel real on tühikutega eraldatud voorude numbrid (esimese rea alguses on viis tühikut järjest) ja igal järgneval real on osaleja nimi ja voorude punktide arvud, mis on samuti eraldatud tühikutega. Kui osaleja ei ole mõnes voorus veel sooritust teinud, on numbri asemel kriips. Osalejate ja voorude arv ei ole ette teada, s.t programm peab töötama ka siis, kui osalejaid ja/või voore on vähem või rohkem.

Näide faili *turniir.txt* sisust

```
     1 2 3 4 5 6
Mari 2 4 5 - 1 4
Juku 1 3 2 7 8 -
Malle - 2 - 3 2 5
Kalle 4 6 - - 2 -
```

1. Kirjuta funktsioon `loe_seis`, mis võtab argumendiks failinime ja tagastab sõnastiku, mille võtmeteks on turniiril osalejate nimed ja väärtusteks voorude tulemuste järjendid (punktid on järjendites andmetüübilt täisarvud).

Näide funktsiooni `loe_seis` tööst eelnevalt kirjeldatud faili *turniir.txt* korral

```
>>> loe_seis("turniir.txt")
{'Mari': [2, 4, 5, '-', 1, 4], 'Juku': [1, 3, 2, 7, 8, '-'],
'Malle': ['-', 2, '-', 3, 2, 5], 'Kalle': [4, 6, '-', '-', 2, '-
']}
```

2. Koosta funktsioon `lisa_tulemus` kindla vooru punktide salvestamiseks. Funktsioon võtab argumentideks osaleja nime, vooru järjekorranumbri (täisarv), sõnastiku ja lisatava tulemuse (täisarv). Kui sellel osalejal vastavas voorus veel tulemust ei ole, asendab funktsioon `lisa_tulemus` vastava kriipsu sõnastikus tulemusega ning väljastab teate, et tulemus on lisatud. Kui osalejal on selle vooru tulemus juba olemas, väljastab funktsioon selle kohta teate ja ei muuda midagi. Funktsioon tagastab sõnastiku uue seisu.

Näide funktsiooni `lisa_tulemus` tööst eelpool toodud sõnastikuga

```
>>> lisa_tulemus("Mari", 3, sõnastik, 0)
Tulemus on juba varem lisatud!
{'Mari': [2, 4, 5, '-', 1, 4], 'Juku': [1, 3, 2, 7, 8, '-'],
'Malle': ['-', 2, '-', 3, 2, 5], 'Kalle': [4, 6, '-', '-', 2, '-
']}
>>> lisa_tulemus("Mari", 4, sõnastik, 2)
Tulemus lisatud!
{'Mari': [2, 4, 5, 2, 1, 4], 'Juku': [1, 3, 2, 7, 8, '-'],
'Malle': ['-', 2, '-', 3, 2, 5], 'Kalle': [4, 6, '-', '-', 2, '-
']}
```

3. Koosta funktsioon `leia_skoor`, mis võtab argumentideks osaleja nime ja sõnastiku ning tagastab selle osaleja punktisumma.

Näide funktsiooni `leia_skoor` tööst eelpool toodud sõnastikuga

```
>>> leia_skoor("Malle", sõnastik)
12
```

4. Koosta põhiprogramm, kus kasutaja saab valida erinevate tegevuste vahel. Pärast iga tegevust küsib programm uuesti, mida kasutaja soovib teha, kuni ta valib programmi töö lõpetamise. Kui kasutaja sisestab

- "1", kuvab programm kõigi osalejate tulemused (ühe inimese tulemused ühel real eraldatuna tühikutega)
- "2", lisab programm osaleja tulemuse järgmiste sammude abi:
    - küsib nime
    - küsib vooru numbrit
    - küsib tulemust
    - lisab tulemuse ja väljastab sellest teate, kasutades funktsiooni `lisa_tulemus`. Kui tulemus on juba olemas, väljastab funktsioon selle kohta teate.
- "3", küsib programm osaleha nime ja leiab funktsiooni `leia_skoor` abil selle osaleja skoori (punktisumma) ning väljastab tulemuse sobival kujul ekraanile.
- "4", leiab programm suurima skooriga osaleha ja väljastab tema nime ning punktisumma.
- "5", teeb programm järgmised sammud:
    - salvestab turniiritabeli uue seisu esialgse failiga samal kujul faili *trniir_uus.txt*
    - väljastab lõpetamise kohta teate,
    - lõpetab töö.

Näide programmi tööst (kasutaja sisend on paksus kirjas)

```
1 - Vaata punktitabelit
2 - Lisa tulemus
3 - Vaata skoori
4 - Leia võitja
5 - Lõpeta programmi töö
Vali tegevus: 1
Mari 2 4 5 - 1 4
Juku 1 3 2 7 8 -
Malle - 2 - 3 2 5
Kalle 4 6 - - 2 –
Vali tegevus: 2
Sisesta nimi: Mari
Sisesta voor: 4
Sisesta punktid: 2
Tulemus lisatud!
Vali tegevus: 1
Mari 2 4 5 2 1 4
Juku 1 3 2 7 8 -
Malle - 2 - 3 2 5
Kalle 4 6 - - 2 -
Vali tegevus: 3
Sisesta nimi: Mari
Mari skoor on 18.
Vali tegevus: 4
Suurima skooriga on Juku (21 punkti).
Vali tegevus: 5
Programm lõpetas töö.
```

Pärast näiteprogrammi töö lõppu peaks faili *turniir_uus.txt* sisu olema:

```
     1 2 3 4 5 6
Mari 2 4 5 2 1 4
Juku 1 3 2 7 8 -
Malle - 2 - 3 2 5
Kalle 4 6 - - 2 -
```

III Eksamikomplekti paberosa uus näidis

NIMI:

Programmeerimise eksami paberosa

Kontrollitud oludes kontrolltöö koosneb paberosast ja arvutiosast. Kokku on aega 180 minutit. Arvestuseks on vaja saada arvestatud nii paberosa kui ka arvutiosa (kumbki vähemalt 50% punktidest).
Paberosa ülesanded lahendatakse

- ilma arvutita,
- ilma materjalide abita.

Paberosa lahenduste esitamise aja otsustate ise. Soovitatav lahendusaeg on 25-30 minutit. Paberosas on kokku 5 ülesannet, iga ülesanne 8 punkti (kokku 40 punkti). Selle osa arvestuseks on vaja koguda 50% võimalikest punktidest ehk vähemalt 20 punkti.

Näiteülesanne 1. Tingimuslause

```
sisend = int(input("Palun sisestage täisarv: "))
if sisend > 0:
    if sisend > 5:
        print("Haru 1")
    if sisend > 10:
        print("Haru 2")
    else:
        print("Haru 3")
    print("Haru 4")
else:
    print("Haru 5")
```

Mis väljastatakse ekraanile, kui kasutaja sisestab

- 0
- 5
- 10
- 20

Näiteülesanne 2. Järjend

```
järjend = [1, 2, 3, 4, 5, 6, 7, 8]
a = järjend[1] + järjend[2]
print(järjend[a])

print("----")

for i in range(5, len(järjend)):
    print(järjend[i])

print("----")

for arv in järjend:
    if arv % 2 == 0:
        print(arv)
```

Mis väljastatakse ekraanile?

Näiteülesanne 3. Funktsioon

```
def kolmeks(arv):
    return arv / 3

n = 27.0
print(kolmeks(n))
print("---")

print(kolmeks(kolmeks(n)))
print("---")

while n > 1:
    print(n)
    n = kolmeks(kolmeks(n))
```

Mis väljastatakse ekraanile?

Näiteülesanne 4. Kahemõõtmeline järjend ja kahekordne tsükkel

```
a = [[1, 7, 6], [2, 4, 5], [8, 3, 1], [10, 9, 8]]

s = a[0][0]
for rida in a:
    for el in rida:
        if el > s:
            print(el)
            s = el
print(s)
```

Mis väljastatakse ekraanile?

Näiteülesanne 5a. Sõnastik ja funktsioon

```
def fun(son):
    for võti in son.keys():
        if võti <= 8:
            print(son[võti])
fun({1: 2, 3: 4, 5: 6, 7: 8, 9: 10, 11: 12})
```

Mida väljastatakse ekraanile?

Näiteülesanne 5b. Rekursioon

```
def dekaade(arv):
    print(arv)
    if arv == 0:
        return 0
    else:
        return 1 + dekaade(arv-10)


print(dekaade(20))
```

Mida väljastatakse ekraanile?

IV Eksamikomplekt 1 (Tartu 25.03)

NIMI:

Programmeerimise eksami paberosa

Kontrollitud oludes kontrolltöö koosneb paberosast ja arvutiosast. Kokku on aega 180 minutit. Arvestuseks on vaja saada arvestatud nii paberosa kui ka arvutiosa (kumbki vähemalt 50% punktidest).
Paberosa ülesanded lahendatakse

- ilma arvutita,
- ilma materjalide abita.

Paberosa lahenduste esitamise aja otsustate ise. Soovitatav lahendusaeg on 25-30 minutit. Paberosas on kokku 5 ülesannet, iga ülesanne 8 punkti (kokku 40 punkti). Selle osa arvestuseks on vaja koguda 50% võimalikest punktidest ehk vähemalt 20 punkti.

Ülesanne 1. Tingimuslause

```
sisend = int(input("Palun sisestage täisarv: "))
if sisend <= 2:
    if sisend == 2:
        print("Haru 1")
    if sisend < 0:
        print("Haru 2")
    else:
        print("Haru 3")
    print("Haru 4")
else:
    print("Haru 5")
```

Mis väljastatakse ekraanile, kui kasutaja sisestab

- -1
- 0
- 2
- 3

Ülesanne 2. Järjend

```
järjend = [1, 2, 3, 4, 5, 6, 7, 8]
a = järjend[-1] // järjend[1]
print(järjend[a])

print("----")

for i in range(len(järjend[:2])):
    print(järjend[i])

print("----")

for arv in järjend:
    if arv >= 4:
        print(arv)
```

Mis väljastatakse ekraanile?

Ülesanne 3. Funktsioon

```
def liida(arv1, arv2):
    return arv1 + arv2

a = 3
b = 4
print(liida(a, b))
print("---")

print(liida(liida(a, b), b))
print("---")

while a < 15:
    print(a)
    a = liida(a, liida(a, b))
```

Mis väljastatakse ekraanile?

Ülesanne 4. Kahemõõtmeline järjend ja kahekordne tsükkel

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

summa = 0
for rida in a:
    for i in range(2, len(rida)):
        print(rida[i])
        summa += rida[i]
print(summa)
```

Mis väljastatakse ekraanile?

Ülesanne 5. Rekursioon

```
def summa(n):
    print(n)
    if n <= 0:
        return 0
    elif n % 2 == 0:
        return n + summa(n-2)
    else:
        return summa(n-1)

print(summa(11))
```

Mida väljastatakse ekraanile?

Programmeerimise eksam 25. märtsil 2022

- Arvutiosa ülesanne lahendatakse
    - pärast pabertesti esitamist;
    - arvutiga;
    - kasutades programmeerimiskeskkonda Thonny;
    - vajadusel materjale (nt kursuse materjalid, enda varasemad tööd, Google...) kasutades.
- Kindlasti tuleb esitada Thonny logid - ilma tervet lahendamisaega hõlmavate Thonny logideta esitust arvesse ei võeta.
- Lahendusprogramm (py-fail) ja Thonny logifailid (zip või txt fail) esitatakse Moodle'is ja igaks juhuks saadetakse ka aadressile tauno.palts@ut.ee
- Logi saamiseks sulge töötav Thonny versioon ja käivita Thonny kontrolltöö arvutiosa alguses ning pärast lahendamise lõpetamist.

Programmide koostamiseks tuleb kasutada keskkonda Thonny. Veendu enne ülesandega alustamist, et logimine on sisse lülitatud (*Tools → Options → General → Log program usage events*). Selleks, et logimine sujuks õigesti, sulge enne ülesandega alustamist ja pärast ülesande lahendamist Thonny ning käivita uuesti. See tagab, et logisse ei jää üleliigset infot.

Ülesanne 1 (30p)

Raba Gümnaasiumis õpib vähe õpilasi. Juku kirjutas faili *vanused.txt* kõigi Raba Gümnaasiumis õppivate laste vanused, igal real üks vanus. **Näide *vanused.txt* sisust:**

```
16
16
16
17
17
18
18
18
18
19
19
```

Kirjuta funktsioon `leia_protsent`, mis

- saab argumendiks kaks täisarvu ja tagastab arvuna, mitu protsenti esimene moodustab teisest ümardatuna kahe komakohani.

Kirjuta programm, mis

- küsib kasutajalt failinime
- loeb kokku kogu failis olevate inimeste arvu ja inimeste arvud antud vanusevahemikkudes (10., 11., 12. klass), eeldusel, et
    - 16 või 17 → 10 klass
    - 18 →11. klass
    - 19 → 12. klass
- väljastab kõikides klassis käivate õpilaste maksimaalse võimaliku arvu ja mitu protsenti on see kõikidest õpilastest (kasutades funktsiooni `leia_protsent`).

Näide programmi võimalikust tööst faili *vanused.txt* korral (kasutaja sisend on paksus kirjas)

```
Sisesta faili nimi: vanused.txt
10. klassis õpib maksimaalselt 5 õpilast, mis moodustab 45.45%
õpilastest.
11. klassis õpib maksimaalselt 4 õpilast, mis moodustab 36.36%
õpilastest.
12. klassis õpib maksimaalselt 2 õpilast, mis moodustab 18.18%
õpilastest.
```

Ülesanne 2 (30p)

Jukul on suur kollektsioon puslesid ja palju sõpru, kes temalt puslesid laenata armastavad. Kirjuta programm, millega on Jukul võimalik lihtsalt arvet pidada, kelle käes pusled on.
Juku kasutab puslede kategoriseerimiseks süsteemi, kus tähed (A, B, C, D, E, F, G, H, I, J) vastavad pusle tükkide arvule nii, et A = 500, B = 1000, C = 1500, D = 2000, …. Kuna puslesid on tal palju, siis lisaks on ta kasutusele võtnud numbrid, ehk iga pusle on tuvastatav tähe- ja numbrikombinatsiooniga (A1, A2, B1, jne). Näiteks A1 on esimene 500-tükiga pusle ja A2 on teine 500-tükiga pusle jne. Võib eeldada, et kui Jukul on mingi tükkide arvuga pusle, siis on tal ka kõiki sellest väiksema arvuga puslesid.
Juku hoiab oma puslede infot failis *pusled.txt*, kus read ja veerud on vastavate puslede kombinatsioonid. Iga kombinatsiooni all on *, kui pusle on Juku enda käes, ja muudel juhtudel selle inimese nimi, kelle käes pusle on. Kui sellele kombinatsioonile vastavat puslet ei eksisteeri, on selle koha peal kriips "-". Info on samuti eraldatud tühikutega.

Näide faili *pusled.txt* sisust

```
  1 2 3 4
A * * * Mari
B * - - -
C * * Kalle -
```

1. Kirjuta funktsioon `loe_kollektsioon`, mis võtab parameetriks faili nime ja tagastab sõnastiku, mille võtmeks on pusle identifitseerimise kombinatsioon ja väärtuseks isik, kelle käes vastav pusle on. Kui seda puslet ei eksisteeri (failis on selle koha peal "-"), siis vastavat kirjet sõnastikku ei lisata.

Näide funktsiooni `loe_kollektsioon` tööst eelnevalt kirjeldatud faili *pusled.txt* korral

```
>>> loe_kollektsioon("pusled.txt")
{'A1': '*', 'A2': '*', 'A3': '*', 'A4': 'Mari', 'B1': '*', 'C1':
'*', 'C2': '*', 'C3': 'Kalle'}
```

2. Kirjuta funktsioon `vaata_puslesid`, mis võtab parameetriks sõnastiku ja väljastab eraldi ridadel kõikide eksisteerivate puslede koodid ja isiku, kelle käes vastav pusle on (Juku puhul võib olla *).

Näide funktsiooni `vaata_puslesid` tööst eelpool toodud sõnastikuga

```
>>> vaata_puslesid(sõnastik)
A1 *
A2 *
A3 *
A4 Mari
B1 *
C1 *
C2 *
C3 Kalle
```

3. Koosta funktsioon `mitu_puslet`, mis võtab argumendiks sõnastiku ja kategooria (tähe). Funktsioon tagastab täisarvu, mis näitab, mitu puslet selles kategoorias on (pole oluline kas on välja laenutatud või Juku käes). *Vihje. Loe kokku sõnastiku võtmete arv, mis algab vastava tähega.*

Näide funktsiooni `mitu_puslet` tööst eelpool toodud sõnastikuga

```
>>> mitu_puslet(sõnastik, "A")
4
>>> mitu_puslet(sõnastik, "I")
0
```

4. Koosta funktsioon `laena_pusle`, mis võtab argumendiks sõnastiku ning tagastab selle muudetud kujul. Funktsioon küsib kasutajalt pusle koodi ja inimese nime, ning märgib vastava pusle antud inimese käes olevaks (võib eeldada, et vastava koodiga pusle eksisteerib ja et kasutaja sisestab korrektse koodi). Kui vastav pusle juba oli välja laenutatud, siis teavitab funktsioon sellest kasutajat.

Näide funktsiooni `laena_pusle` tööst eelpool toodud sõnastikuga

```
>>> laena_pusle(sõnastik)
Sisesta pusle kood: A2
Sisesta laenaja nimi: Malle
{'A1': '*', 'A2': 'Malle', 'A3': '*', 'A4': 'Mari', 'B1': '*',
'C1': '*', 'C2': '*', 'C3': 'Kalle'}
>>> laena_pusle(sõnastik)
Sisesta pusle kood: A8
Vastava koodiga pusle ei ole Juku käes.
{'A1': '*', 'A2': '*', 'A3': '*', 'A4': 'Mari', 'B1': '*', 'C1':
'*', 'C2': '*', 'C3': 'Kalle'}
```

5. Koosta funktsioon `tagasta_pusle`, mis võtab argumendiks sõnastiku ning tagastab selle muudetud kujul. Funktsioon küsib kasutajalt pusle koodi, ning märgib vastava pusle uuesti Juku käes olevaks (võib eeldada, et vastava koodiga pusle eksisteerib ja et kasutaja sisestab korrektse koodi). Kui vastav pusle juba oli Juku käes, siis teavitab funktsioon sellest kasutajat.

Näide funktsiooni `laena_pusle` tööst originaalis toodud sõnastikuga

```
>>> tagasta_pusle(sõnastik)
Sisesta pusle kood: A2
Vastava koodiga pusle juba on Juku käes.
{'A1': '*', 'A2': '*', 'A3': '*', 'A4': 'Mari', 'B1': '*', 'C1':
'*', 'C2': '*', 'C3': 'Kalle'}
>>> tagasta_pusle(sõnastik)
Sisesta pusle kood: A4
{'A1': '*', 'A2': '*', 'A3': '*', 'A4': '*', 'B1': '*', 'C1':
'*', 'C2': '*', 'C3': 'Kalle'}
```

6. Koosta funktsioon `kirjuta_faili`, mis võtab argumendiks sõnastiku ning faili nime ja kirjutab vastava nimega faili sõnastikus oleva seisu samal kujul esialgse puslede failiga (info eraldatud tühikutega). Pärast faili kirjutamist teavitab funktsioon sellest kasutajat.
Vihje1. Esimese rea kirjutamiseks saad leida suurima puslede arvu kasutades funktsiooni mitu_puslet.
*Vihje2. Liigu järjest tähtede ja arvudega maksimaalseteni, mis kataloogis eksisteerivad (A1, A2, A3, A4, B1, B2, …), kontrolli, kas vastav kombinatsioon eksisteerib sõnastikus, ning vastavalt sellele lisa faili väärtus (-, *, nimi).*

Näide funktsiooni `kirjuta_faili` tööst (algse sõnastiku puhul peab faili sisu olema sama nagu näide)

```
>>> kirjuta_faili(sõnastik, "pusled_uus.txt")
Andmed faili salvestatud.
```

7. Koosta põhiprogramm, kus kasutaja saab valida tegevuste vahel. Programm salvestab esmalt puslede seisu funktsiooni loe_kollektsioon kasutades sõnastikku. Pärast iga tegevust küsib programm uuesti, mida kasutaja soovib teha, kuni ta valib programmi töö lõpetamise. Kui kasutaja sisestab

- "1", kuvab programm puslede kataloogi funktsiooni `vaata_puslesid` abil,
- "2", kuvab kasutajale kategoorias olevate puslede arvu Juku kataloogis. Selleks:
    - küsib pusletükkide arvu, ja kui sisestatakse sobiv arv (peab jaguma 500ga) väljastab funktsiooni `mitu_puslet` abil ekraanile vastava suurusega puslede arvu.
- "3", märgib pusle laenutatuks funktsiooni `laena_pusle` abil.
- "4", märgib pusle tagastatuks funktsiooni `tagasta_pusle` abil.
- "5", teeb programm järgmised sammud:
    - salvestab puslede tabeli uue seisu faili *pusled_uus.txt* funktsiooni `kirjuta_faili` abil,
    - väljastab lõpetamise kohta teate,
    - lõpetab töö.

Näide programmi tööst (kasutaja sisend on paksus kirjas)

```
1 - Vaata pusle kataloogi
2 - Leia puslede arv
3 - Laena pusle
4 - Tagasta pusle
5 - Lõpeta programmi töö
Vali tegevus: 1
A1 *
A2 *
A3 *
A4 Mari
B1 *
C1 *
C2 *
```

```
C3 Kalle
Vali tegevus: 2
Sisesta pusle suurus: 700
Palun sisesta sobiv suurus.
Sisesta pusle suurus: 500
500 jupiga puslesid on 4700
Vali tegevus: 3
Sisesta pusle kood: A2
Sisesta laenaja nimi: Malle
Vali tegevus: 4
Sisesta pusle kood: C3
Vali tegevus: 5
Andmed faili salvestatud.
Programmi töö lõpetatud.
```

Pärast näiteprogrammi töö lõppu peaks faili *pusled_uus.txt* sisu olema:

```
  1 2 3 4
A * Malle * Mari
B * - - -
C * * * -
```

V Eksamikomplekt 2 (Tallinn 26.03)

NIMI:

Programmeerimise eksami paberosa

Kontrollitud oludes kontrolltöö koosneb paberosast ja arvutiosast. Kokku on aega 180 minutit. Arvestuseks on vaja saada arvestatud nii paberosa kui ka arvutiosa (kumbki vähemalt 50% punktidest).
Paberosa ülesanded lahendatakse

- ilma arvutita,
- ilma materjalide abita.

Paberosa lahenduste esitamise aja otsustate ise. Soovitatav lahendusaeg on 25-30 minutit. Paberosas on kokku 5 ülesannet, iga ülesanne 8 punkti (kokku 40 punkti). Selle osa arvestuseks on vaja koguda 50% võimalikest punktidest ehk vähemalt 20 punkti.

Ülesanne 1. Tingimuslause

```
sisend = int(input("Palun sisestage täisarv: "))
if sisend >= 0:
    if sisend == 0:
        print("Haru 1")
    if sisend % 3 == 0:
        print("Haru 2")
    else:
        print("Haru 3")
    print("Haru 4")
else:
    print("Haru 5")
```

Mis väljastatakse ekraanile, kui kasutaja sisestab

- -1
- 0
- 1
- 3

Ülesanne 2. Järjend

```
järjend = [1, 2, 3, 4, 5, 6, 7, 8]
a = järjend[len(järjend) // 4]
print(a)
print(järjend[a])

print("----")

for i in range(4, len(järjend)):
    print(järjend[i])

print("----")

for arv in järjend:
    if arv % 3 == 0:
        print(arv)
```

Mis väljastatakse ekraanile?

Ülesanne 3. Funktsioon

```
def korruta(arv):
    return arv * 2

n = 3
print(korruta(n))
print("---")

print(korruta(korruta(n)))
print("---")

while n < 15:
    print(n)
    n = korruta(korruta(n))
```

Mis väljastatakse ekraanile?

Ülesanne 4. Kahemõõtmeline järjend ja kahekordne tsükkel

```
a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

summa = 0
for i in range(len(a)):
    for j in range(len(a)):
        if i > j:
            print(a[i][j])
            summa += a[i][j]
print(summa)
```

Mis väljastatakse ekraanile?

Ülesanne 5. Sõnastik ja funktsioon

```
def fun(son):
    for võti, väärtus in son.items():
        if väärtus <= 8:
            print(võti)
fun({1: 2, 3: 4, 5: 6, 7: 8, 9: 10, 11: 12})
```

Mis väljastatakse ekraanile?

Programmeerimise eksam 26. märtsil 2022

- Arvutiosa ülesanne lahendatakse
    - pärast testi esitamist;
    - arvutiga;
    - kasutades programmeerimiskeskkonda Thonny;
    - vajadusel materjale (nt kursuse materjalid, enda varasemad tööd, Google...) kasutades.
- Kindlasti tuleb esitada Thonny logid - ilma tervet lahendamisaega hõlmavate Thonny logideta esitust arvesse ei võeta.
- Lahendusprogramm (.py fail) ja Thonny logifailid (.zip või .txt fail) esitatakse Moodle'is ja saadetakse ka aadressile tauno.palts@ut.ee
- Logi saamiseks sulge töötav Thonny versioon ja käivita Thonny kontrolltöö arvutiosa alguses ning pärast lahendamise lõpetamist.

Programmide koostamiseks tuleb kasutada keskkonda Thonny. Veendu enne ülesandega alustamist, et logimine on sisse lülitatud (*Tools → Options → General → Log program usage events*). Selleks, et logimine sujuks õigesti, sulge enne ülesandega alustamist ja pärast ülesande lahendamist Thonny ning käivita uuesti. See tagab, et logisse ei jää üleliigset infot.

Ülesanne 1 (30p)

Juku armastab teha kolme sorti moosi - maasikatest, vaarikatest ja mustikatest. Iga kord, kui Juku tegi oma tavapärase koguse mingit moosi, kirjutas ta faili *moosid.txt* vastavad marjad, igal real üks kirje. **Näide faili *moosid.txt* sisust:**

```
maasikad
mustikad
mustikad
vaarikad
maasikad
vaarikad
mustikad
```

Kirjuta funktsioon `leia_protsent`, mis

- saab argumendiks kaks täisarvu ja tagastab ujukomaarvuna arvuna, mitu protsenti esimene arv moodustab teisest arvust ümardatuna kahe komakohani.

Kirjuta programm, mis

- küsib kasutajalt failinime,
- loeb kokku kogu failis olevate erinevate mooside valmistamise korrad,
- väljastab kõikide mooside kohta mitu korda Juku neid valmistas ja mitu protsenti on see kõikidest moositegudest (kasutades funktsiooni `leia_protsent`).

Näide programmi võimalik väljund antud faili korral (kasutaja sisend on paksus kirjas)

```
Sisesta faili nimi: moosid.txt
Maasikatest tehti moosi 2 korda, mis moodustab 28.57% moositegudest.
Vaarikatest tehti moosi 2 korda, mis moodustab 28.57% moositegudest.
Mustikatest tehti moosi 3 korda, mis moodustab 42.86% moositegudest.
```

Ülesanne 2 (30p)

Juku kasvatab erinevaid aiasaadusi, mida sõpradele müüa. Kirjuta programm, et aidata Jukul olemasolevate asjade üle arvet pidada.

Juku hoiab oma aiasaaduste infot failis *tooted.txt*, kus igal real on tühikuga eraldatud ühe aiasaaduse info. Iga rea alguses on toote nimi, selle järel kogus (täisarvuna) ning hind (ujukomaarvuna).

Näide faili *tooted.txt* sisust

```
kurk 5 0.75
tomat 7 1.25
kapsas 3 0.5
```

1. Kirjuta funktsioon `loe_saadused`, mis võtab argumendiks faili nime ja tagastab maatriksi, kus on iga toode eraldi järjendis ja iga järjendi element vastavalt teisendatud.

Näide funktsiooni `loe_saadused` tööst faili *tooted.txt* korral

```
>>> loe_saadused("tooted.txt")
[['kurk', 5, 0.75], ['tomat', 7, 1.25], ['kapsas', 3, 0.5]]
```

2. Kirjuta funktsioon `vaata_saaduseid`, mis võtab argumendiks maatriksi ja väljastab eraldi ridadel kõikide eksisteerivate aiasaaduste nimed.

Näide funktsiooni `vaata_saaduseid` tööst eelpool toodud maatriksiga

```
>>> vaata_saaduseid(maatriks)
kurk
tomat
kapsas
```

3. Koosta funktsioon `lisa_saadus`, mis võtab argumendiks maatriksi. Funktsioon küsib kasutajalt aiasaaduse nime ja koguse. Kui vastav saadus on olemas, uuendatakse maatriksis vastava elemendi kogust. Kui ei ole, siis küsib funktsioon kasutajalt ka hinna ja lisab maatriksisse uue järjendi vastava toote infoga. Võib eeldada, et kasutaja sisestab sobiva info. Funktsioon tagastab muudetud kujul maatriksi.

Näide funktsiooni `lisa_saadus` eelpool näidatud maatriksiga (kasutaja sisend paksu kirjaga)

```
>>> lisa_saadus(maatriks)
Sisesta aiasaadus: kaalikas
Sisesta kogus: 3
Sisesta hind: 1.5
[['kurk', 5, 0.75], ['tomat', 7, 1.25], ['kapsas', 3, 0.5],
['kaalikas', 3, 1.5]]
>>> lisa_saadus(maatriks)
Sisesta aiasaadus: kurk
Sisesta kogus: 2
[['kurk', 7, 0.75], ['tomat', 7, 1.25], ['kapsas', 3, 0.5]]
```

4. Koosta funktsioon `müü_saadus`, mis võtab argumendiks maatriksi ning tagastab selle muudetud kujul. Funktsioon küsib kasutajalt saaduse nime ja koguse ning vähendab maatriksis vastavalt saaduse kogust ja teavitab kasutajat, palju selle müümisest teeniti. Kui antud aiasaadust nii palju (või üldse) ei ole, teavitab funktsioon sellest kasutajat. Kui mõne saaduse uueks koguseks on 0, siis eemaldatakse see maatriksist.

Näide funktsiooni `müü_saadus` tööst eelpool toodud maatriksiga

```
>>> müü_saadus(maatriks)
Sisesta aiasaadus: porgand
Vastavat aiasaadust ei ole.
[['kurk', 5, 0.75], ['tomat', 7, 1.25], ['kapsas', 3, 0.5]]
>>> müü_saadus(maatriks)
Sisesta aiasaadus: kurk
Sisesta kogus: 8
```

```
Vastavat aiasaadust ei ole nii palju.
[['kurk', 5, 0.75], ['tomat', 7, 1.25], ['kapsas', 3, 0.5]]
>>> müü_saadus(maatriks)
Sisesta aiasaadus: kapsas
Sisesta kogus: 3
Teeniti 1.5 eurot.
Toode sai otsa.
[['kurk', 5, 0.75], ['tomat', 7, 1.25]]
>>> müü_saadus(maatriks)
Sisesta aiasaadus: kurk
Sisesta kogus: 2
Teeniti 1.5 eurot.
[['kurk', 3, 0.75], ['tomat', 7, 1.25], ['kapsas', 3, 0.5]]
```

5. Koosta funktsioon `kogu_väärtus`, mis võtab argumendiks maatriksi ning tagastab ujukomaaarvu, mis vastab sellele summale, mille Juku teeniks kui ta kõik maatriksis olevad aiasaadused maha müüks.

Näide funktsiooni `kogu_väärtus` tööst originaalis toodud maatriksiga

```
>>> kogu_väärtus(maatriks)
14.0
```

6. Koosta funktsioon `kirjuta_faili`, mis võtab argumendiks maatriksi ning faili nime ja kirjutab vastava nimega faili maatriksis oleva seisu samal kujul esialgse aiasaaduste failiga. Pärast faili kirjutamist teavitab funktsioon sellest kasutajat.

Näide funktsiooni `kirjuta_faili` tööst. (PS! maatriksit muutmata, peab uues faili *tooted_uus.txt* sisu olema sama, mis esialgses failis *tooted.txt*).

```
>>> kirjuta_faili(maatriks, "tooted_uus.txt")
Andmed faili salvestatud.
```

7. Koosta põhiprogramm, kus saab valida erinevate tegevuste vahel. Programm salvestab esmalt aiasaaduste seisu funktsiooni `loe_saadused` abil maatriksisse. Pärast igat tegevust küsib programm uuesti, mida kasutaja soovib teha, kuni ta valib programmi töö lõpetamise. Kui kasutaja sisestab

- "1", kuvab programm aiasaaduste kataloogi funktsiooni `vaata_saaduseid` abil,
- "2", lisab kataloogi uued aiasaadused funktsiooni `lisa_saadus` abil,
- "3", eemaldab kataloogist mingi arvu aiasaaduseid funktsiooni `müü_saadus` abil,
- "4", leiab ja väljastab müüdavate aiasaaduste väärtuse funktsiooni `kogu_väärtus` abil,
- "5", teeb programm järgmised sammud:
    - salvestab aiasaaduste maatriksi seisu faili *tooted_uus.txt* funktsiooni `kirjuta_faili` abil,
    - väljastab lõpetamise kohta teate,
    - lõpetab töö.

Näide programmi tööst (kasutaja sisend on paksus kirjas)

```
1 - Vaata aiasaaduste kataloogi
2 - Lisa aiasaadusi
3 - Müü aiasaadusi
4 - Leia võimalik tulu
5 - Lõpeta programmi töö
Vali tegevus: 1
kurk
tomat
kapsas
Vali tegevus: 2
Sisesta aiasaadus: kaalikas
Sisesta kogus: 3
Sisesta hind: 2
Vali tegevus: 2
Sisesta aiasaadus: kurk
Sisesta kogus: 1
Vali tegevus: 3
Sisesta aiasaadus: tomat
Sisesta kogus: 5
Teeniti 6.25 eurot.
Vali tegevus: 3
Sisesta aiasaadus: kurk
Sisesta kogus: 6
Teeniti 4.5 eurot.
Toode sai otsa.
Vali tegevus: 4
Maksimaalne võimalik tulu on 10.0 eurot.
Vali tegevus: 5
Andmed faili salvestatud.
Programmi töö lõpetatud.
```

Pärast näiteprogrammi töö lõppu peaks faili *tooted_uus.txt* sisu olema:

```
tomat 2 1.25
kapsas 3 0.5
kaalikas 3 2.0
```

VI Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kadi Sammul**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Gümnaasiumi programmeerimise eksami koostamine ja katsetamine**, mille juhendaja on **Tauno Palts,** reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.
2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Kadi Sammul

06.05.2022