8.1 Kordamiseks

KORDAMISEKS

Olemegi jõudnud kursuse viimasesse nädalasse. Programmeerimise mõttes enam uusi kohustuslikke teemasid juurde ei tule - kordame eelnevaid. Kordamine võiks aidata arvestusülesande (8.1a) ja loovtöö (8.1b) edukat sooritamist.

Sel nädalal tulebki lahendada arvestusülesanne või loovtöö. Valige palun teile sobivam variant. (Arvestusülesanne on ka näidiseks nö päris arvestustööks, mille peaksid tegema need tulevased või praegused Tartu Ülikooli üliõpilased, kes tahaksid seda ainet oma õpingute osana arvestada.)

Lisaks arvestusülesande lahendusele või loovtööle tuleb veel esitada vastava ülesande lahendamise protsessi kirjeldus (eelkõige Thonny logifailina) (vastavalt ülesanded 8.2a ja 8.2b). Nimelt salvestab Thonny praktiliselt iga tegevuse, mis lahendamisel tehakse. Logifailide uurimisel saab olulist infot lahendamise protsessi kohta ja see võimaldab edaspidi kursust paremini korraldada ning üldse saada teaduslikku informatsiooni lahendamise kohta.

8. nädalal on ka mitmed silmaringimaterjalid, mille kohta on küsimused nädala testis.

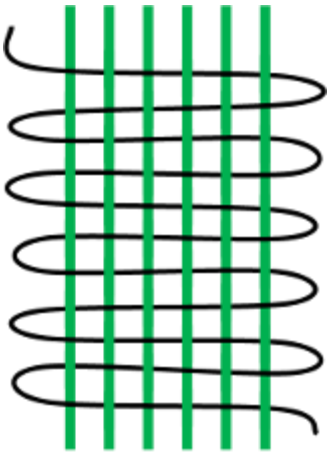
Kui kõik on päris valmis, siis palume vastata lõpuküsitlusele.

8.2 Arvestusülesanne

Kohustuslikult tuleb lahendada (8.1a ja 8.2a) või (8.1b ja 8.2b).

KONTROLLÜLESANNE 8.1a ARVESTUSÜLESANNE

Tõenäoliselt on paljud näinud kaltsuvaipu, kuid võib-olla pole mõelnud sellele, kuidas neid valmistatakse. Kaltsuvaipu kootakse spetsiaalsetel telgedel, millele kinnitatakse hulk tugevast materjalist niite ehk lõimi, mille vahele hakatakse põimima kaltsuribasid. Lõimed on järgneval joonisel kujutatud rohelisena. Meie eesmärk on vajaliku lõimeniidi kogupikkuse leidmine. Kuna lõimed on telgedel pinges, siis pärast valmimist tõmbub vaip mõnevõrra kokku. Kokkutõmbumise kompenseerimiseks võetakse vaiba algpikkus soovitavast lõpp-pikkusest 20% suurem. Samuti arvestatakse, et iga lõime kumbagi otsa tuleb jätta sidumiseks 25 cm varu.



Koostada funktsioon `lõimede_pikkus`, mis

- võtab argumentideks vaiba lõpp-pikkuse (ujukomaarv) ja lõimede arvu (täisarv),
- arvutab ja tagastab vaiba lõimede kogupikkuse ümardatuna sajandikeni.

Koostada programm, mis

- küsib kasutajalt
  - failinime, kus on vaipade lõpp-pikkused ujukomaarvudena meetrites eraldi ridadel,
  - kõrvuti olevate lõimede arvu 5-meetriste ja pikemate vaipade puhul (täisarv),
  - kõrvuti olevate lõimede arvu lühemate vaipade puhul (täisarv);
- loeb failist vaipade pikkused,
- arvutab (funktsiooni `lõimede_pikkus` abil) ja väljastab ekraanile iga vaiba lõimede kogupikkuse,
- arvutab ja väljastab ekraanile, kui palju läheb lõimeniiti vaja kõigi vaipade peale kokku ümardatuna sajandikeni.

Näide arvutuskäigust

Olgu näiteks uues hoones tahtmine kasutada kaltsuvaipu lõpp-pikkustega 7 m, 4,9 m, 3,63 m ja 5 m. Seejuures 5-meetriste või pikemate lõpp-pikkustega vaipade puhul olgu lõimi kõrvuti 120 (vaiba laius umbes 70 cm), lühemate puhul 140 (vaiba laius umbes 80 cm).

Esimese vaiba üks lõim on pikkusega 7 * 1,2 + 0,5 = 8,9 meetrit. Kokku on sellel vaibal lõimi 120, sest vaiba pikkus on 5 meetrit või rohkem. Seega kulub selle vaiba peale kokku 8,9 * 120 = 1068 meetrit lõimeniiti. Teise, kolmanda ja neljanda vaiba peale kulub vastavalt 893,2; 679,84 ja 780 meetrit. Järelikult on kõikide vaipade jaoks vaja kokku 1068 + 893,2 + 679,84 + 780 = 3421,04 meetrit lõimeniiti.

Näide funktsiooni `lõimede_pikkus` rakendamisest

```
>>> lõimede_pikkus(7, 120)
1068.0
>>>
```

Näide programmi tööst

Faili delta_vaibad.txt sisu:

```
7
4.9
3.63
5
```

```
>>> %Run yl8.1a.py
  Sisestage failinimi: delta_vaibad.txt
  Sisestage 5-meetriste ja pikemate vaipade lõimede arv: 120
  Sisestage lühemate vaipade lõimede arv: 140
  1068.0
  893.2
  679.84
  780.0
  Kõigi vaipade peale läheb vaja 3421.04 meetrit lõimeniiti.

>>>
```

Arvestusülesandele sarnane ülesanne koos ühe võimaliku lahendusega võib olla ka abiks mõtete kogumisel. Lahendamise video: http://www.uttv.ee/naita?id=23698

KONTROLLÜLESANNE 8.2a ARVESTUSÜLESANDE LAHENDAMISE PROTSESS

Esitamine *Moodle*'is.

Tekstina esitatakse arvamus arvestusülesande raskusastme ja sobivuse kohta. Thonny kasutajad peavad esitama logifaili. Kes teeb mõne teise vahendiga, see peab põhjalikumalt kirjeldama lahendamise raskusi ja kergusi ning samuti peab andma võimalikult täpse ajalise ülevaate, kui palju ülesande lahendamisele aega kulus.

Logifaili saamise video: https://www.youtube.com/watch?v=UkBo8F9prIc&feature=youtu.be

JUHEND (Thonnyt kasutamata user_logs'ini)

1. Avage Minu Arvuti/My Computer.
2. Valige sealt kaust, kuhu on installeeritud Windows.
3. Kettalt valige Users kaust, kust leidke kasutaja kaust, kes on kasutanud Thonnyt.
4. Kasutaja kaustas peaks asuma .thonny kaust.
5. .thonny kaust sisaldab kausta User_logs, kuhu on salvestatud Thonny logid..
6. Sealt valige logid, mille failinimi on seotud antud ülesandega. (Failinimi on kuupäev, mil antud ülesandeid sooritasite.)
7. Tõstke need logid ühte kausta ning pakkige kokku kas .rar või .zip failiks.
8. Esitage kokkupakitud fail Moodle' kaudu.

JUHEND 2 (Thonnyst user_logs'ini)

1. Avage Thonny.
2. Valige menüüst Tools.
3. Tools'i alt leiate Open Thonny User folder, mis avab kasutaja Thonny kausta.
4. Avage sellest kaustast user_logs.
5. Sealt valige logid, mille failinimi on seotud antud ülesandega. (Failinimi on kuupäev, mil antud ülesandeid sooritasite.)
6. Tõstke need ühte kausta ning pakkige kokku kas .rar või .zip failiks
7. Esitage kokkupakitud fail Moodle'i kaudu.

JUHEND 3 (Thonnyst user_logs'ini)

1. Avage Thonny.
2. Valige menüüst Tools.
3. Tools'i alt leiate Export usage logs, mis pakib logid zip-failiks kokku.
4. Esitage kokkupakitud fail.

8.3 Loovtöö

Kohustuslikult tuleb lahendada (8.1a ja 8.2a) või (8.1b ja 8.2b).

KONTROLLÜLESANNE 8.1b LOOVTÖÖ

Programmide kirjutamine on loominguline tegevus, eriti siis, kui ka ülesanne on vaja ise välja mõelda. Seepärast ongi 8. nädala ülesandeks võimalik valida arvestusülesanne või loovtöö. Kui valite loovtöö, siis palume valida endale südamelähedane temaatika ja jõukohane ülesanne. Kuna lahendus peab demonstreerima teie erinevaid oskusi, siis on toodud vastavad nõuded.

Loovtöö nõuded

- Programmil peab olema kirjeldus, mis tuleb kirjutada programmi algusesse kommentaarina. Kirjelduses peab olema kirjas
  - mis programmiga on tegemist (1-2 lauset);
  - miks kirjutasid selle programmi.
- Programmi koodis peavad olema kommentaarid, mis (eriti keerulisemates kohtades) selgitavad programmi tööd.
- Loovtöös peab kasutama järgmisi programmeerimise kontruktsioone:
  - tingimuslause;
  - juhuslike arvude genereerimine;
  - järjend;
  - tsükkel;
  - funktsioon;
  - failist lugemine ja/või kasutajalt andmete küsimine;
  - faili ja/või ekraanile väljastamine;
  - graafiline liides (Easygui, Tkinter, kilpkonnaga).
- Kuna kõigi nende konstruktsioonide kasutamine ei pruugi olla mõistlik, siis võib neist kaks jääda kasutamata.
- Töö peab olema originaalne ja autori enda tehtud. Kindlasti eelistada lihtsamat programmi, kui plagiaadi esitamist.

Töö tuleb esitada *Moodle*'i keskkonda. Loovtöö esitamiseks peab üles laadima programmi ja ühe ekraanipildi, mis programmitööd demonstreerib.

KONTROLLÜLESANNE 8.2b LOOVTÖÖ TEGEMISE PROTSESS

Esitamine *Moodle*'is.

Tekstina esitatakse arvamus loovtöö raskusastme ja sobivuse kohta. Thonny kasutajad peavad esitama logifaili. Kes teeb mõne teise vahendiga, see peab põhjalikumalt kirjeldama lahendamise raskusi ja kergusi ning samuti peab andma võimalikult täpse ajalise ülevaate, kui palju ülesande lahendamisele aega kulus.

Logifaili saamise video: https://www.youtube.com/watch?v=UkBo8F9prIc&feature=youtu.be

JUHEND (Thonnyt kasutamata user_logs'ini)

1. Avage Minu Arvuti/My Computer.
2. Valige sealt kaust, kuhu on installeeritud Windows.
3. Kettalt valige Users kaust, kust leidke kasutaja kaust, kes on kasutanud Thonnyt.
4. Kasutaja kaustas peaks asuma .thonny kaust.
5. .thonny kaust sisaldab kausta User_logs, kuhu on salvestatud Thonny logid..
6. Sealt valige logid, mille failinimi on seotud antud ülesandega. (Failinimi on kuupäev, mil antud ülesandeid sooritasite.)
7. Tõstke need logid ühte kausta ning pakkige kokku kas .rar või .zip failiks.
8. Esitage kokkupakitud fail Moodle' kaudu.

JUHEND 2 (Thonnyst user_logs'ini)

1. Avage Thonny.
2. Valige menüüst Tools.
3. Tools'i alt leiate Open Thonny User folder, mis avab kasutaja Thonny kausta.
4. Avage sellest kaustast user_logs.
5. Sealt valige logid, mille failinimi on seotud antud ülesandega. (Failinimi on kuupäev, mil antud ülesandeid sooritasite.)
6. Tõstke need ühte kausta ning pakkige kokku kas .rar või .zip failiks
7. Esitage kokkupakitud fail Moodle'i kaudu.

JUHEND 3 (Thonnyst user_logs'ini)

1. Avage Thonny.
2. Valige menüüst Tools.
3. Tools'i alt leiate Export usage logs, mis pakib logid zip-failiks kokku.
4. Esitage kokkupakitud fail.

8.4 Silmaring: Ametid IT sektoris

AMETID IT SEKTORIS

Infotehnoloogia alal on palju erinevaid töökohti. Osades ametites on programmeerimine kesksel kohal, teistes jälle on seos kaudsem. Eesti Infotehnoloogia ja Telekommunikatsiooni Liit on loonud portaali, kus on ka erinevate IT ametite ülevaated: http://startit.ee/karjaar/

IT sektoris töötavate inimeste kohta on selles silmaringi materjalis kaks videot. Esimene on kursuse *Programmeerimise alused* jaoks tehtud video *Inimesed ITs:* http://www.uttv.ee/naita?id=23564 , kus mitmed programmeerimisega seotud inimesed räägivad oma ametist, ITst ja programmeerimisest üldiselt ja selle õppimisest ning arutatakse teemal *IT on poiste mängumaa*.

Teine on kursuse *Programmeerimisest maalähedaselt* jaoks tehtud video *Firmad ja ametid: intervjuud inimestega:* http://www.uttv.ee/naita?id=21173 , kus mitmed programmeerimisega tugevalt seotud inimesed räägivad oma ametist, programmeerimisest üldiselt ja selle õppimisest ning näitavad, millega nad igapäevaselt tegelevad.

8.5 Silmaring: Erinevad programmeerimiskeeled

PROGRAMMEERIMISE PARADIGMAD

Konkreetses teadusharus ja eluvaldkonnas võib kasutusel olla mitu erinevat üldtunnustatud mõistete, seaduste ja meetodite süsteemi (paradigmat), millel rajaneb uurimine ja õpetamine. Ka programmeerimises on võimalikud erinevad paradigmad, mis määravad teatud vaated programmile, algoritmidele jm. Erinevatele paradigmadele vastavad erinevad keeled (või vähemalt erinevad võimalused ühes keeles).

Programmeerimise paradigmasid saab klassifitseerida mitmel moel. Esiteks saab jaotada programmeerimiskeeled kõigepealt kahte gruppi: imperatiivsed ja deklaratiivsed.

Imperatiivsed ehk käskivad keeled on keeled, kus programmi põhiliseks elemendiks on *käsk* ehk *instruktsioon*. Imperatiivses keeles kirjutatud programm kirjeldab üksikasjalikult, mida on vaja teha ja kuidas seda on vaja teha - *programm on käskude jada*. Imperatiivsetes keeltes tuleb täpselt ette "öelda", kuidas lahenduseni jõutakse. Tuntumad imperatiivsed keeled on C, C++, Java, Python.

Deklaratiivsed keeled on keeled, kus kirjeldatakse ehk *deklareeritakse* seosed ja reeglid, mis antud ülesande valdkonda kuuluvad. Deklaratiivsed keeled on keeled, kus programmeerija ei pruugi alati kõiki algoritmi detaile kirja panna, vaid võib esitada otsitava lahenduse kirjelduse, ning juba programmi täitmise käigus otsustab süsteem automaatselt, mis viisil täpselt seda lahendust otsida. Tuntumad deklaratiivsed keeled on: Prolog, Haskell jt.

Imperatiivseid keeli saab edasi jaotada struktuurseks ja objektorienteeritud keelteks.

Deklaratiivsed keeled jaotatakse funktsionaalse programmeerimise keelteks ja loogilise programmeerimise keelteks.

STRUKTUURSED KEELED

Struktuursete keelte põhiideeks on lähenemine probleemile kui tervikule ja seejärel probleemi tükeldamine väiksemateks osadeks. Struktuurprogrammeerimise käigus vaadeldakse esmajoones funktsioone ja protseduure. Kui on välja selgitatud vajalikud tegevused ja algoritmid, siis teises järjekorras vaadeldakse nende tegevuste teostamiseks vajaminevaid andmeid. Tuntumad struktuursed keeled on Pascal, C.

Struktuursete keeltega tutvumiseks vaatleme näiteks keele C programme.

Klassikaline *Hello world* programm on tavaliselt esimene programm, mida kirjutatakse, kui õpitakse uut programmeerimiskeelt. Üks tuntud kollektsioon sisaldab 512 *Hello world* programmi rohkem või vähem tuntud programmeerimiskeeltes ning sama fraasi tõlget 76 erinevas inimkeeles.

Esimene C-programm

Klassikaline *Hello world* programm C keeles:

```
// Programm, mis kirjutab ekraanile tervituse.
 #include <stdio.h> // sisendi/väljundi päis
 main () { // peafunktsioon
    printf ("Hello, world!\n"); // väljund (reavahetusega)
 }
```

Teine C-programm

```
// Programm, mis väljastab ekraanile arvude 1, 2, ..., 9, 10 ruudud.
 #include <stdio.h>
 main () {
    int i; // muutuja deklareerimine
    for (i = 1; i<11; i++) { // tsükkel
       printf("%d ",i*i); // väljund
    }
    printf("\n"); // reavahetus
 }
```

OBJEKTORIENTEERITUD KEELED

Erinevalt struktuursest lähenemisest suunab objektorienteeritud lähenemise kasutamine esmase tähelepanu andmetele ja alles seejärel tõstatatakse küsimus, mida nende andmetega teha saab. OO lähenemine püüab esitada programmis esinevaid andmeid ja nendega teostatavaid tegevusi sarnastena meie igapäevases elus ettetulevatele objektidele, kusjuures objekte kujutatakse 'intelligentsetena'. Näiteks on meil objekt KALENDER. Küsides nüüd endalt, mida KALENDER teha OSKAB, leiamegi selle objektiga seotud tegevused: soovitud kalendrilehe näitamine ja lehekeeramine. Mitte kalendri VAATAMINE vaid NÄITAMINE, sest kalender ise ei vaata. Ja lehekeeramise laseme tal endal teha, seda võiks ta ju osata. Tuntumad objektorienteeritud keeled Java, C++ jne.

Vaatame näiteks keele Java programme.

Esimene Java-programm

```
// Programm, mis kirjutab ekraanile tervituse.
 class Hello { // programmi algus
    public static void main(String[] args) { // peafunktsioon
       System.out.println("Hello, world!");  // väljund
(reavahetusega)
    }
 }
```

Teine Java-programm

```
// Programm, mis väljastab ekraanile arvude 1, 2, ..., 9, 10 ruudud.
 class Ruudud { // programmi algus
    public static void main(String[] args) { // peafunktsioon
       for (int i = 1; i < 11; i++) { // tsükkel ühest kuni kümneni
          System.out.print(i*i+" ");  // väljund
       }
       System.out.println(); // reavahetus
    }
 }
```

FUNKTSIONAALSED KEELED

Funktsionaalse programmeerimise keeled on keeled, kus lahendus kirjeldatakse funktsioonide kogu abil. Funktsionaalsete keelte idee on programmide kirjutamine (matemaatiliste) funktsioonide defineerimise teel, määramata seejuures täpselt ära, mis strateegia järgi funktsiooni resultaati tuleb arvutada. Funktsionaalsete keelte omapära seisneb selles, et erinevalt imperatiivse keele programmidest ei ole üheselt ära määratud funktsioonide täitmise järjekord. Neid kasutatakse väga laialdaselt tehisintellekti probleemide lahendamisel. Tuntumad funktsionaalsed keeled on: Lisp, Scheme, Haskell jt.

Funktsionaalsed programmeerimiskeeled on oma ajaloo vältel pälvinud rohkem teoreetikute kui praktikute tähelepanu. Funktsionaalsest programmeerimisest pärinevad mitmed programmeerimise kontseptsioonid, nt kõrgemat järku funktsioonid, rekursioon, laisk arvutus ja palju muud. Ajapikku on paljud neist leidnud rakendamist paljude praktilisema suunitlusega keelte poolt.

Esimene Haskell-programm

```
-- Programm, mis kirjutab ekraanile tervituse.
 module Hello (main) where     -- peamoodul, mooduli päis
 main = putStr "Hello, world!\n"   -- väljastamine (reavahetusega)
```

Teine Haskell-programm

```
-- Programm, mis väljastab ekraanile arvude 1, 2, ..., 9, 10 ruudud.
 module Squares (main) where           -- mooduli päis
 squares n m = [ i^2 | i <- [n..m] ]   -- funktsiooni defineerimine
 main = print $ squares 1 10           -- väljastamine
```

LOOGILISED KEELED

Loogiliste keelte omapära seisneb selles, et nendes kirjutatud programm kirjeldab ülesandes kasutatavate objektide seosed loogiliste avaldistena, mille väärtused saavad olla kas tõesed või väärad. Ka programmi töö tulemuseks on esitatud küsimuse tõeväärtus. Tuntumaks loogiliseks keeleks on Prolog.

Esimene Prolog-programm

```
% Programm, mis kirjutab ekraanile tervituse.
 ?- write('Hello, world!'), nl. % väljastamine (reavahetusega)
```

Teine Prolog-programm

```
% Programm, mis väljastab ekraanile arvude 1, 2, ..., 9, 10 ruudud.
 squares(11). % töö lõpetamine
 squares(M) :- K is M*M, write(K), write(' '), N is M+1, squares(N).
% reegel
 ?- squares(1). % töö alustamine
```

SKRIPTIMISE KEELED

Skriptimise keeled ehk skriptikeeled on interpreteerivad programmeerimise keeled. Skript on interpreteeritavas keeles kirjutatud programm. Tavalised programmid enne käivitamist peavad olema muudetud binaarfailideks ehk kompileeritud. Skriptid aga jäävad inimesele loetavad ja need on interpreteeritud ehk tõlgitud arvutile mõistetavasse keelde ridade kaupa iga käivitamise korral.

On olemas erinevad skriptikeeled, näiteks teksti töötlemise skriptikeeled (Perl, Bash jt.), veebiprogrammeerimise skriptikeeled (PHP, JavaScript jt.) jt.

Esimene Perl-programm

```
# Programm, mis kirjutab ekraanile tervituse.
 print "Hello, world!\n"; # väljastamine (reavahetusega)
```

Teine Perl-programm

```
# Programm, mis väljastab ekraanile arvude 1, 2, ..., 9, 10 ruudud.
 for($i = 1; $i <= 10; ++$i) { # tsükkel
    print $i*$i, ' '; # väljastamine
 }
 print "\n"; # reavahetus
```

Esimene PHP-programm

```
<?php // php skripti algus; skript, mis kirjutab ekraanile tervituse
 echo "Hello, world!\n"; // väljastamine
 ?>
```

Teine PHP-programm

```
<?php // php skripti algus; skript, mis väljastab ekraanile arvude
1, 2, ..., 9, 10 ruudud
   for($i = 1; $i <= 10; ++$i){ // tsükkel
     echo $i*$i; // väljastamine
     echo " ";
   }
   echo "\n"; // reavahetus
 ?>
```

PHP-programmi on võimalik kirjutada ka HTML-koodi sisse ja vaadata veebibrauserist.

Muudame natuke esimest PHP-programmi ja paneme tulemuse HTML-koodi sisse. Oletame, et meil on olemas kaks sõne ja me tahame neid kokku panna ja väljastada.

```
<html>
  <head>
   <title>PHP Test</title>
  </head>
  <body>
   <?php // php skripti algus; skript, mis kirjutab ekraanile
tervituse
    $a = "Hello"; // muutuja deklareerimine
    $b = "world"; // muutuja deklareerimine
    echo $a . ", " . $b . "!\n"; // väljastamine
   ?> <!-- php skripti lõpp. Kuna asub juba HTML koodi sees, siis on
vaja teistmoodi kommenteerida -->
  </body>
 </html>
```

ERINEVAD PROGRAMMEERIMISKEELED JA PARADIGMAD

Mõned programmeerimiskeeled toetavad konkreetset paradigmat (Haskell funktsionaalset, Prolog loogilist), teised toetavad aga mitmeid erinevaid. Nii näiteks võivad keeltes C++ ja JavaScript kirjutatud programmid olla puhtalt struktuursed või puhtalt objektorienteeritud või sisaldada mõlema stiili elemente. Millist stiili kasutada, on programmeerijate otsustada. Ka Python võimaldab mitut programmeerimisstiili, näiteks objektorienteeritud, struktuurset või funktsionaalset programmeerimist. Järgmises tabelis (https://en.wikipedia.org/wiki/Comparison_of_programming_languages) võrreldakse levinumaid programmeerimiskeeli.

PROGRAMMEERIMISKEELTE KASUTUSALAD

Erinevate keelte kasutuslevik on pidevas muutumises ja seda on huvitav jälgida näiteks siin: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Sageli kasutatakse suurte programmide loomisel mitu erinevat programmeerimiskeelt (programmi erinevate osade jaoks). Nt järgmises tabelis (https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites) on näidatud, milliseid programmeerimiskeeli on kasutatud kõige populaarsemate veebilehtede puhul.

Pythoni juures võib esile tuua arendamise kiirust, samas programmid võivad jääda aeglaseks. Samuti on Pythonis olemas ka kõik objektorienteeritud programmeerimise vahendid. Python on hea keel prototüüpimiseks: tihtipeale luuakse mingi arvutiprogrammi esialgne kavand selles keeles ning hiljem realiseeritakse see mõnes kiiremas kõrgkeeles. Vahel kirjutatakse ainult programmi aeglasemad osad C-s või C++-is.

Python on olnud oluline Google'i programmeerimisel algusest peale kuni tänaseni. Kümned Google'i insenerid kasutavad Pythonit. Samuti on Facebook, NASA, Yahoo! Groups ja YouTube osaliselt kirjutatud Python-is. Ka Thonny on kirjutatud keeles Python. Siin on välja toodud veel mõned organisatsioonid, mis kasutavad Pythonit.

Kuna C on riistvarale lähedane keel, siis kasutatakse seda operatsioonisüsteemide draiverite ning operatsioonisüsteemide tuumade kirjutamiseks. C++ kasutatakse tihti suuremate ja veakindlate süsteemide loomisel, mille puhul on kiirus oluline. Enamik operatsioonisüsteeme on kirjutatud C ja C++ keeles. Nende hulka kuuluvad Windows 95, 98, 2000, XP, Vista, Windows 7, Windows 8, Apple Mac OS X, Symbian OS ja Be-OS, Google Chrome OS, RIM BlackBerry OS 4.x, Apple iPhone iPod Touch ja iPad OS jne. Samuti on mitmed veebilehitsejad kirjutatud C ja C++ keeles, nt Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, Safari, Netscape Navigator, Opera ja Opera Mini. Ka Microsoft Office tooted (nagu Word, Excel, PowerPoint, Outlook jt) on kirjutatud C++ keeles. Järgmiste veebilehtede programmeerimisel on samuti kasutatud C ja C++: Facebook, YouTube, Amazon, Paypal. Veel üks tuntud programm, mis on kirjutatud C ja C++ keeles, on Winamp.

Hetkel on Java üks populaarsemaid programmeerimiskeeli, mida kasutab umbes 9 miljonit arendajat, eriti klient-server veebirakendustes. Tuntumad rakendused on Twitter, Amazon, Google, Facebook ja Youtube. Samuti Google ja Android, Inc. on otsustanud kasutada Javat Android operatsioonisüsteemi loomisel.

PHP on üldotstarbeline skriptimise keel, mis on eriti sobilik veebiserverite jaoks. Veebruaris 2014 82% veebilehtedel kasutati PHP'd serveripoolse programmeerimiskeelena. Ka nt Moodle ja Facebook kasutavad PHP keelt.

Aga nt Skype on kirjutatud hoopis keeles Embarcadero Delphi (ehk objektorienteeritud Pascal).

8.6 Silmaring: Õppimisvõimalused

Võib-olla tahate programmeerimist veel edasi õppida. Me mõtleme kursusele **"Programmeerimise alused II"** (3 EAP), aga selle ettevalmistamisega läheb veel aega ja tõenäoliselt toimub see 2017. aasta kevadel.

Mitmetes Eesti kõrgkoolides on informaatika ja infotehnoloogiaga seotud õppekavasid. Kõige lähemalt teame muidugi Tartu Ülikooli arvutiteaduse instituudis toimuvat.

Esimest korda võetakse sel suvel vastu üliõpilasi magistriõppekavale **Infotehnoloogia mitteinformaatikutele**. Sinna on oodatud inimesed, kel on eelnev magistrikraad (või rakenduskõrgharidus või bakalaureusekraad koos 2-aastase töökogemusega) mõnelt muult erialalt.

Tartu Ülikoolis saab veel õppida ka järgmistel õppekavadel

- bakalaureuseõppes
  - Informaatika
  - Arvutitehnika
- magistriõppes
  - Matemaatika- ja informaatikaõpetaja
  - Informaatika (inglise keeles)
  - Arvutitehnika ja robootika (inglise keeles)

Huvitavad õppekavad on ka Tallinna Tehnikaülikoolis, Eesti Infotehnoloogia Kolledžis, Tallinna Ülikoolis ja teisteski kõrgkoolides. (Loetelu õppekavadest.) Informaatika ja infotehnoloogia õppekavagrupi õppekavadel õpib kõrgkoolides umbes 4630 üliõpilast.

Infotehnoloogiaga seotud õppekavasid on ka kutseõppeasutustes.

Ka täienduskoolitusena on erinevaid infotehnoloogiaga seotud kursusi, nt siin ja siin ning siin.

Silmad tasub lahti hoida ka majandus- ja kommunikatsiooniministeeriumi initsiatiivi uute IT-spetsialistide koolitamise osasuhtes. Sellest kirjutab ka ajaleht "Postimees".

8.7 Silmaring: Informaatikaolümpiaadist

Mõnes mõttes ei ole programmeerimine tavaline õppeaine. Näiteks paljudes koolides ei õpetata programmeerimist üldse ja seal, kus õpetatakse, tehakse seda väga erinevatel viisidel. Samas olümpiaadide mõttes on programmeerimine täiesti võrdväärselt matemaatika, füüsika, keemia, bioloogia ja paljude teiste ainete kõrval. Siiski ei leia olümpiaadide nimekirjast programmeerimist, vaid hoopis informaatika. Informaatikaolümpiaadi ülesanded on siiski põhiliselt programmeerimise ülesanded.

Matemaatikas ja mitmetes teistes ainetes jõutakse olümpiaadi juurde tavaliselt koolipoolse initsiatiiviga. Näiteks korraldatakse koolis eelvoor, mille parimad suunatakse piirkonna vooru ja sealt omakorda valitakse vabariiklikku vooru pääsenud. Informaatikaolümpiaadi puhul on õpilase enda initsiatiiv märksa olulisem.

Õpilane saab ise registreeruda eelvooru, mis toimub novembris kindlates kohtades üle Eesti. Omal initsiatiivil saab õpilane osaleda ka lahtisel võistlusel, mis toimub oktoobris. Lahtisel võistlusel esitatakse lahendused veebipõhiselt ja kuhugi kohale tulema ei pea. Umbes nagu meie e-kursusel! Nii eelvoorust kui lahtiselt võistluselt valitakse lõppvooru osalejad. Lõppvooru põhjal aga valitakse rahvusvahelisele võistlusele osalejad.

Nüüd siis on muidugi küsimus, kas meie kursuse “Programmeerimise alused” lõpetanu võiks olümpiaadil osaleda? Võib öelda, et teatud oskused on tal selleks tõesti juba olemas. Viimastel on olümpiaadi ametlikeks keelteks olnud C++, Pascal, Java ja Python. Seega on Python osalemiseks sobiv. Sageli antakse ülesande sisend ette tekstifailina ja ka väljund tuleb esitada tekstifailina. Nendega toimetamiseks peaks meie kursusest piisama.

Põhiline on probleem muidugi selles, et programm selle sisendiga täpselt seda teeks, mida ülesandes nõutud. Ülesanded võivad olla erineva raskusastmega - põhikoolis ilmselt mõnevõrra lihtsamad kui gümnaasiumis. Eraldi on veel edasijõudnute rühm. Hulk varasemaid ülesandeid on kättesaadavad aadressil http://eio.ut.ee/.

Soovitame julgesti neid ülesandeid vaadata ja proovida lahendada. Kui ülesanded sobivad, siis võib proovida olümpiaadidel osaleda.

Aegajalt korraldab Tartu Ülikooli teaduskool ka kursusi olümpiaadide ettevalmistuseks .

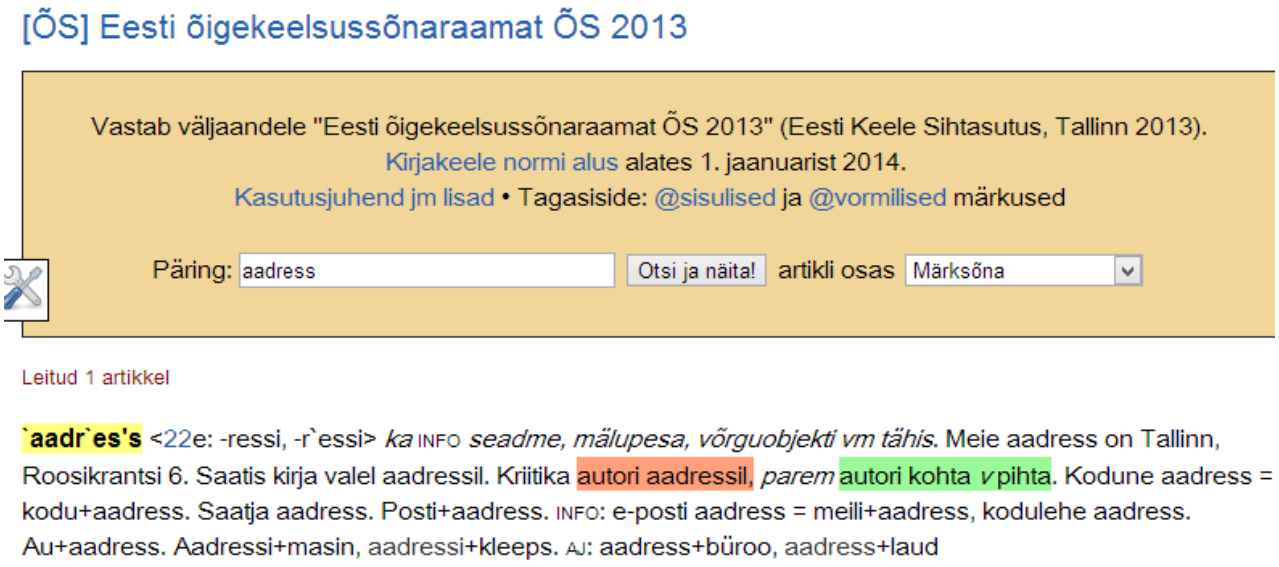
Eesti Informaatikaolümpiaadi žürii esimees Targo Tennisberg (Bondora AS, süsteemiarhitekt) kirjutab raamatut võistlusprogrammeerimisest. See pole küll veel päris valmis, aga autori lahkel loal võib juba paljut huvitavat lugeda aadressil http://targotennisberg.com/eio/VP/.

8.8 Silmaring: Regulaaravaldis

Päris paljud programmeerijatel ette tulevad ülesanded on seotud sõnade ja teksti analüüsiga. Seesugusest analüüsist on sageli abi ka ülesannete juures, kus seda esmapilgul ehk ei ootakski.

ÕIGEKEELSUSSÕNARAAMAT. METAMÄRK

Kui soovime teada saada mingi eestikeelse sõna õigekirja või tähendust, siis võime abi otsida "Eesti õigekeelsussõnaraamatust ÕS 2013". Näiteks kui tahame teada, kuidas käändub sõna "aadress", siis sisestame veebilehel http://www.eki.ee/dict/qs/ olevasse päringuvormi selle sõna ja saamegi tema kohta info kätte.



Mis saab aga siis, kui me ei tea täpselt, kuidas otsitavat sõna kirjutada? Selle peale on mõeldud! Probleemsete tähtede asemele saame panna küsimärgi. Näiteks kui me ei tea, kas õige on "kandidaat" või "kanditaat", võime sisestada "kandi?aat". Küsimärki nimetatakse selles kontekstis **metamärgiks** ehk metasümboliks. Metamärk ei tähista iseennast, vaid viitab mingile reeglile, mille põhjal saame meie (ja ka arvuti) aru, kuidas antud kohta sõnas mõista. Seega metamärgina tähistab küsimärk ühte suvalist sümbolit. Otsingusõna "kandi?aat" puhul leitakse selliseid sõnu sõnaraamatust ainult üks. Sobivaid vasteid võib-olla ka rohkem, näiteks "?ell" puhul on neid lausa kuus. Kui asendada sõna esimene täht (või mitu tähte) küsimärgiga, siis see annab juba päris hea meetodi, kuidas riimuvaid sõnu otsida!

Otsingute puhul on teiseks levinud metamärgiks tärn *, mis tähistab suvalist arvu mingeid märke (seejuures ka märkide puudumist). Näiteks päring "meh*aanika" leiab kõik sõnad, mis algavad tähtedega "meh" ja lõppevad tähtedega "aanika". Nüüd võib sobivate vastete hulgas olla ka lühemaid ja pikemaid sõnu, sest tärni asemel võib olla null, üks või rohkem märki. Nii saame "ka*ak" korral vasteteks teiste hulgas "kaak", "kajak" kui ka "kaelkirjak".

Otsingus võib korraga olla mitu metamärki, nt *us?epp. Otsingusõna võib koosneda ka ainult metamärkidest, näiteks "??????????????????????".

Ülesanne

Proovige veebilehel http://www.eki.ee/dict/qs/ erinevaid päringuid, näiteks neid:

- kandi?aat
- *sepp
- *us?epp
- ??????????????????????

REGULAARAVALDIS

Metamärkide süsteeme on mitmeid ja olenevalt süsteemist võivad metamärkide tähendused ka erineda. Järgnevalt vaatleme süsteemi, mis on paljudes programmeerimiskeeltes levinud. Selles süsteemis ongi märkide ? ja * tähendus teistsugune kui ülaltoodud päringutes.

Käesolevas tekstis on kasutatud termineid "sõna" ja "sõne". "Sõna" on siin tavalises tähenduses, näiteks "armastus" ja "lapsepõlv" on sõnad. Mõiste "sõne" on laiem, sest sõne võib sisaldada ka tühikuid, kirjavahemärke, isegi reavahetusi jms. Sõnest võib mõelda kui suvalisest sümbolitejadast. Mingi sõne alamsõneks nimetatakse sõnet, mis sisaldub esialgses sõnes. Sõne on näiteks “Ma tahaksin kodus olla”, selle alamsõned on teiste hulgas näitekst "Ma", "kodu", "sin ko" ja " ".

Eeltoodud ÕSi päringute puhulgi anti välja need sõnad, mis vastasid päringus määratud mustrile. Mustrit saab kirjeldada **regulaaravaldise** abil. Iga konkreetse sõne puhul saab siis öelda, kas see sõne vastab antud regulaaravaldisele või mitte. Regulaaravaldistes on tähtis roll metamärkidel, siin materjalis piirdume ainult mõne olulisemaga. Need on toodud järgnevas tabelis.

Metamärk	Kirjeldus	Näide
.	üks suvaline sümbol	"a.i"
		Sümboli "a" järel on suvaline sümbol ja seejärel "i".
		Näiteks "abi", aga mitte "appi".
?	0 või 1 kord eelnevat sümbolit või regulaaravaldist	"ai?"
		Sümboli "a" järel on 0 või 1 sümbolit "i".
		Ainsad näited on "a", "ai".
*	suvaline arv kordi (ka 0 korda) eelnevat sümbolit või regulaaravaldist	"ai*"
		Sümboli "a" järel on suvaline arv

		korda sümbolit "i".
		Näiteks "a", "ai", "aii", "aiiii".
{n}	n korda eelnevat sümbolit või regulaaravaldist	"ai{2}"
		Sümboli "a" järel 2 sümbolit "i".
		Ainus näide on "aii".
[ ]	võimalike sümbolite hulga märkimine	"a[ij]"
		Sümboli "a" järel on sümbol "i" või sümbol "j".
		Ainsad näited on "ai", "aj".
[-]	võimalike sümbolite vahemiku märkimine	"a[0-9]"
		Sümboli "a" järel on suvaline number.
		Näiteks "a0", "a5".
( )	rühmitamine	"a(ij)*"
		Sümboli "a" järel on suvaline arv korda sümbolipaari "ij".
		Näiteks "a", "aij", "aijij", "aijijij", aga mitte "aji".
^	sõne algus	"^ai"
		Sõned, mis algavad sümbolipaariga "ai".
		Näiteks "ai", "aine", "aiandus".
$	sõne lõpp	"ai$"
		Sõned, mis lõpevad sümbolipaariga "ai".
		Näiteks "ai", "hai", "samurai".

Metamärkide abil saame koostada lihtsamaid ja keerulisemaid regulaaravaldisi. Ühes regulaaravaldises võib olla ka mitu metamärki. Näiteks ".a*" tähistab, et suvalisele sümbolile järgneb suvaline arv korda sümbol "a". Kuna suvaline arv võib ka 0 olla, siis "a" võib ka puududa.

Sõne vastavust regulaaravaldisele saab Pythonis kontrollida funktsioonidega `re.match` ja `re.search`. Kõigepealt tuleb importida moodul *re*. Nimi *re* tuleb ingliskeelsest mõistest *regular expression*, mis tähendabki tõlkes regulaaravaldist. Seega kirjutame programmi algusesse `import re`.

FUNKTSIOON re.match

Vaatleme esialgu funktsiooni `re.match`, millega saab kontrollida, kas sõne algus vastab regulaaravaldisele. Esimeseks argumendiks tuleb sellele funktsioonile anda regulaaravaldis, teiseks argumendiks sõne. Funktsiooni `re.match` saame kasutada valikulauses.

Järgnev programm kontrollib, kas sõne "abi" algab regulaaravaldisele "a.i" vastavalt. Punkt (".") tähistab üht suvalist sümbolit ning antud hetkel sobib suvalise sümboli rolli täht "b".

```
import re
if re.match("a.i", "abi"):
    print("vastab")
else:
    print("ei vasta")
```

Funktsioon `re.match` otsib vastavust sõne algusest. Seega `re.match("a.i", "Nabi")` puhul vastavust pole. Küll aga ei pea regulaaravaldisele vastava osaga sõne ära lõppema, nt `re.match("a.i", "abiks")` näitab vastavust.

Mitme metamärgiga regulaaravaldised avavad uusi (ka keerulisi) võimalusi. Näiteks ".*" tähendab, et suvalisi sümboleid on suvaline arv kordi (võib olla ka null korda). Regulaaravaldisega ".*a.i" on seega vastavuses “Nabi”, “esmaabi”, “aabits”, “aminohape on oluline” ja veel lõpmatu arv teisi sõnesid. Metamärgiga $ on võimalik kontrollida, kas regulaaravaldisele vastav alamsõne lõpetab sõne. Näiteks regulaaravaldisega “a.i$” on vastavuses “abi”, aga “abiks” ei ole.

Ülesanne

Selle ja mitmete edasiste ülesannete puhul antakse *Kontrolli*-nuppu vajutades reaktsioon iga vastusevariandi jaoks eraldi. Reaktsioon oleneb sellest, kas vastusevariant on õigesti märgitud/märkimata või mitte.

Millised järgmistest sõnedest on `re.match` mõttes vastavuses regulaaravaldisega `".a?t"`?

- [ ] `"ot"`
- [ ] `"aat"`
- [ ] `"2at"`
- [ ] `"matt"`
- [ ] `".a?t"`

Järgmises ülesandes on regulaaravaldistes kasutusel mitmed erinevad metamärgid.

Ülesanne

Millistega järgmistest regulaaravaldistest on sõna `"ABBA"` funktsiooni `re.match` mõttes vastavuses?

- [ ] `"AB{2}A"`
- [ ] `"AB{2}$"`
- [ ] `"AB{2}"`
- [ ] `"AB{2}[AB]$"`
- [ ] `"[AB]{3}A"`
- [ ] `"B{2}A"`

FUNKTSIOON re.search

Funktsiooni `re.match` kõrval kasutatakse ka üsna sarnaselt toimivat funktsiooni `re.search`, mis otsib vastavust mitte ainult sõne algusest, vaid üle kogu sõne. Nii saame näiteks leida, kas sõnes leidub kolm kõrvuti olevat numbrit. Seda, kas tegemist on numbriga saab kontrollida "[0-9]" abil (vt metamärkide tabelit). Loogelistes sulgudes saame ette anda, mitu korda eelnevat sümbolit või avaldist peab olema.

```
import re
if re.search("[0-9]{3}", "Auto registreerimisnumbriga 007PYT
lähenes"):
    print("sisaldub")
else:
    print("ei sisaldu")
```

Tegelikult võiksime "[0-9]" asemel kirjutada "[0123456789]", sest seda me ju otsime: kas sõnes leidub selline koht, kus on järjest kirjutatud kolm numbrit ehk sümbolit, millest igaüks võib olla kas 0, 1, 2, 3, 4, 5, 6, 7, 8 või 9.

Ülesanne

Millised järgnevatest variantidest tuvastavad vastavuse?

- [ ] `re.search("^b", "abcd")`
- [ ] `re.match("b", "abcd")`
- [ ] `re.search("^a", "abcd")`
- [ ] `re.match("a", "abcd")`

Järgmises näites püüame leida, kas teates sisaldub korvpalli seis. Esialgu eeldame lihtsustatult, et kummalgi võistkonnal on kahekohaline skoor.

```
import re

korvpalli_seis = "[0-9]{2}:[0-9]{2}"

teade = "Esimene korvpallimäng olümpiamängudel toimus 1. augustil
1936. aastal. Eesti võitis Prantsusmaa 34:29."

if re.search(korvpalli_seis, teade):
    print("sisaldab ilmselt korvpalliseisu")
else:
    print("ei sisalda ilmselt korvpalliseisu")
```

Tegelikult võib korvpallis skoor olla ka kolmekohaline ja mängu algusjärgus muidugi ka ühekohaline. Seda kirjeldab paremini regulaaravaldis `"[0-9][0-9]?[0-9]?:[0-9][0-9]?[0-9]?"`.

VEEL (JA PÄRIS ELULISI) VARIANTE

Nüüd vaatame elulist, kuid pisut keerulisemat näidet geneetika valdkonnast. See on eelkõige mõeldud just huvitavaks ja harivaks lugemiseks - ülesannetes neid teadmisi ei nõuta.

DNA koosneb nukleotiididest adeniin (A), guaniin (G), tsütosiin (C) ja tümiin (T). Lühendite abil saame DNA ahela lõigu kirja panna näiteks sõnena, mis koosneb tähtedest A, G, C ja T. DNA ahelas paiknevad geenid, millest transkripteeritakse RNA molekule ning millest omakorda moodustatakse aminohapetest koosnevaid proteiine, mis on vajalikud meie elutegevuseks ja panevad paika, kuidas meie keha töötab. Mitte kogu DNA ahel ei koosne geenidest, vaid geenid asuvad teatud positsioonidel DNA ahelas. Selleks, et teada saada, millal mõni geen algab või lõppeb, on olemas algus- ja lõpukoodonid - kindlad järjestused, mis annavad teada, et selle koha pealt tuleb geeni lugema hakata või lugemine lõpetada. Alguskoodoniks on järjestus ATG ning lõpukoodoneid on lausa mitu: TGA, TAG ja TAA.

Kui tahame näiteks leida, kas mõnel DNA lõigul asub potentsiaalne geen, peaksime kontrollima, kas leidub selline järjestus, mis algab kolmikuga ATG ning lõppeb kolmikuga TGA, TAG või TAA. Lisaks peab nende kahe kolmiku vahele jääma kolmega jaguv arv nukleotiide. Kolmega jaguvust on vaja kontrollida seetõttu, et ühele aminohappele vastab nukleotiidide kolmik ning kui nukleotiidide arv ei jagu kolmega, siis ei saa sellisest järjestusest teha proteiini.

Järgmine näide demonstreerib, kuidas teha kindlaks, kas antud DNA lõigus on mõni potentsiaalne geen:

```
import re

dna_lõik
= "AAGGCTAGACTGGCTTAGCCTCCCAGCCTACATCTTTCTCCCATGCTGGATGCTTCCTGCCCTCA
AACATCAGGCTCCATGTTTTTCAGCTTTGGGACTCACAGTGGCTTCCTTGCTCCTCAGTTTACAGACA
GCCTATTGTGGGACCTTGTGATTGTGTGAGTTAATACTACTTAATAAACTCCCATATGTAGGTGTGTG
TATATGTCTTATTAGTTCTATCCCTCTAGAGAACCCTGACTGATACACCTGCTCTAACAGGCTGTTGA
ATGCCCATGGCTTTTCCAGGTGCAGGGCATAAGCTGCCAGTGGATATACTATTATTGGGTCTACAAGG
TGATGGCCCACTTCTCACAGCTCCATTAGGCAGTGTTCCGGTGAAGGCTCTGTGTGGGGACTTGAACC
CTTCATTTTCCCTTGATACTACCCTAGAAGTTATCTGTGGGGGTTCTGTTTCTGAAGTAGGCTTCTGC
CTGGACACCCAGGCTTTTCATACATCCTTGGAAATCTACGCAGAAGGTGCCAAGACTTATTCATTCTT
ACACTCTGTGCACTTGCAGGCTTAACAGTACATAGAAGACACCAAGGCTTATGGCAGTTTGTACTTTC
CTGAGCAGCAACATGAAGAGTATGTGGGAGGTTTTGAGCCAAGGCTGGAGCCGGAGTTGCCTAGACGT
GGGGAACAGTATTTCAAGACTACACAGGGCAGCAAAGCCCTGGGCCTGGCCATGGAAACCATTGTTTC
CTACTAGGCCTCAGGACTTGTCATGGAAGGGGCTGCCATGAAGTCTCTGAAATGCTTTCCAAGCATCC
TCCTCATTGTCTTGGCTATCAGCACTTGGCTCCCTTTTTAGTCATGCATACTTCTCTAGCAAGTCGTT
TCTTCACAGCCTGCCTGAATTCCTCTCCCCAGAAAGGTTTTTTTCTTCCTCTGCCACATGGCCAGGCT
GCAAATTTTGAAAACTTTTGTTGTCTGCTTCCCTTTTAAATATAGGTTCTAACTTTAAGTCATTCCTT
TGTTCCTAAATCTGAGCATAGGTTGTTAGAAGCTGCCAGGCCACATTTTGAATGCTTTTCTGCTTAGA
AATTTCCTCCACCAGGTACCCTAAATCATCCCCATGAAGTTCAAACTTCCAAAGATCCTCAGGGCATG
AACAGAATGCATCCAGGGTTTTTACTAAGGTATAACACACATGACTTTTGCTCCAGCTCTCAATAAGT
TATTTCTATCTGAGACCTCAGCAGCCTGGAATTCGGTCTCCGTATCACTATAAGTATTTTGATCACAA
CCATTTAACCAGTCTCTAAGACATTTCTAACTTTTCCTCATCTTCCTGTCTTCTTCAAAGCTCTCCAA
ACTCTTTCAAACATAGCTCATTACCCAGTTCCAAAGCTGCTTCCACATTTTCAGATATCTTTATAGCA
AAGGCTCGATCCTCAATACCAATTTTCTTTACTAGACTGCTCTTGTATTTCTATGAAGAAATATCTGA
GAGAGGGTAATTTATAAAGAAAAGAGGTTTAATTGGCTCCCAGTACTGCAGGATTTACAGAAATAATG
ATACTGGCATCTGCCTGGCTTCTTGGGAGGTCTCAGGCAACTTACAATCACGGTAGAAGGTGAAGAGA
GAGCAGGCATGCCACTTGGTGAAAGCAGGAGCGTGAGAGAGACAGTGGGTG"

leidub = re.search("ATG(...)+T((GA)|(AG)|(AA))", dna_lõik)

if leidub:
    print("Antud DNA lõigus asub vähemalt üks potentsiaalne geen.")
else:
    print("Antud DNA lõigus ei asu ühtegi potentsiaalset geeni.")
```

Regulaaravaldiste abil saab näiteks üles leida kõik potentsiaalsed geenid ja nende asukohad DNA ahelal ning edasi saavad geenitehnoloogid ja bioinformaatikud uurida, millise funktsiooni eest erinevad geenid vastutavad.

Püüame nüüd ka kasutatud regulaaravaldisest `ATG(...)+T((GA)|(AG)|(AA))` detailsemalt aru saada.

- "ATG" regulaaravaldise alguses näitab, et otsitakse alamsõne, mis algab tähtedega "ATG".
- (...) tähistab kolme suvalist märki. Pluss ("+") pärast seda ütleb, et eelnevat (antud juhul siis kolme märki) peab olema 1 või rohkem korda. See tagab, et algus- ja lõpukoodoni vahele jääb kolmega jaguv arv nukleotiide.

- Otsitav alamsõne peab lõppema kolmikuga "TGA", "TAG" või "TAA". See tagatakse regulaaravaldise lõpuga `T((GA)|(AG)|(AA))`, kus märk | tähendab "või". Seega võib lõpukoodon olla kas "TGA", "TAG" või "TAA".

Ülesanne

Millistele regulaaravaldistele vastavad alamsõnad on sõnes "Ta jõudis vaevu-vaevu kohale."? Teisiti sõnastades, mida saab kirjutada lünka _____, et ekraanile väljastataks "sisaldub".

```
import re
if re.search(_____, "Ta jõudis vaevu-vaevu kohale."):
    print("sisaldub")
else:
    print("ei sisaldu")
```

- [ ] `"(vaevu){2}"`
- [ ] `"(vaevu.){2}"`
- [ ] `"ko[a-z]al"`
- [ ] `"j[a-zõ]u"`
- [ ] `"..."`

Lõpetuseks võib veel öelda seda, et regulaaravaldist algkursustes tavaliselt ei käsitleta, aga loodetavasti olete nõus, et tegemist on huvitava ja kasuliku teemaga.

8.9 Lugu: alkeemia VII

LUGU: ALKEEMIA VII

Selle loo on kirjutanud Tartu Ülikooli informaatika eriala esmakursuslane Ander Peedumäe.

"Ma tean väga hästi, mida teiesugused meist arvavad." Rongaisand kirtsutas nina ja taarus edasi-tagasi. "Võlurid on laisad. Võlurid ei suuda luua midagi uut. Võlurid on lihtsalt raipekotkad, kes teiste loitsude korjustest parima välja nokivad. Ometi olete teie need, kes minu asjades sorivad.“ Ta pööras selja ning vaatas taevasse, nagu üritaks midagi leida. Ra heitis pilgu maha ning köhatas närviliselt. „Ma tean, et see ei olnud õige, kuid ma üritasin vaid koduteed leida. Ma ei soovinud sellega halba. Teal ei olnud sellega midagi pistmist.“

„Ka mina ei soovi teile halba. Me oleme kõik need, kelleks me loodud oleme. Teie olete alkeemikud. Ka mina olin kunagi alkeemik. Me alustasime samast kohast, aga elu on nõudnud erinevaid oskusi. Katlad, pudelid ja savikamakad on vaid pindmine kiht. Rännukivist ja mägikoobastest ei tasu rääkidagi.“ Tundus, nagu pikk mustapäine mees oleks hetkelisest unest ärganud. Ta vaatas alkeemikute poole ning ulatas neile enda hõlma alt pärgamendirulli. „Loodan, et te leiate tee tagasi. Teil on vaid üks katse. Ärge seda mõtlematult raisake.“

Teale tundus, et tumedates riietes võlur naeratas hetkeks. Järgmisel kadus mees aga mustade sulgede keerisesse. Võib-olla kujutas tüdruk seda vaid ette.

Tea avas rulli ning hakkas seda usinalt lugema. Tundus, et loitsus ei olnudki midagi keerulist. Seisa voolava vee kohal ning lausu õiged sõnad. Ra jaoks tundus nende seisukord imelik. Miks peaks mees abi pakkuma neile, kes üritasid talt varastada? Kuidas ta üldse teadis, et nad mägedest tulevad?

Tea ei raisanud aega: „Tule, lähme sillale. Meil on voolavat vett tarvis, et seda lausuda.“ Ta vibutas pärgameti Ra näo ees ning hakkas liikuma.

Teel sillani ei olnud ühtegi jälge võitlusest: mitte ühtegi kõrbenud rohukõrt ega murdunud puud. Vesi vulises vaikselt ning Tea hakkas ürikust ette lugema.

Ra peatas loitsu lugemise. „Kuhu me minna tahame?“

„Ülikooli alkeemiatorni?“

„Ma kahtlen, kas me sinna jõuame.“

Taevas tõmbus tumedaks ning puulehed keerlesid tuules. Ra oli hakanud mõtlema viimastele nädalatele, mille jooksul nad olid reisinud mägedest kaugemale. Kõik tundus tühi ja

kummaline. Kõik algas väikestest tähelepanekutest nagu tundmatud tähtkujud taeva all ja veidrad kuufaasid.

„Kas sa ei ole märganud midagi veidrat," küsis ta Tealt ja rääkis talle asjadest, mis ei tundunud õiged.

Tea turtsatas naerda, kuid jäi äkitselt vait.

„Rännukivi tõi meid mäeküljele, kus oli linnake ja ülesanded, mis olid nagu meile loodud. Mitme päevatee kaugusel ei ole midagi muud peale inimeste, kes tahavad meid minema saata. See ei olegi päris?"

Pilved kiskusid nüüd päris mustaks ja Ra märkas, et seal, kus pidi seisma veski, oli tühjus ning ka puuvõradest piilunud onnike oli läinud.

Tea hakkas loitsu lugema. Noormees üritas talle öelda, et see ei tööta, kuid naine ei teinud väljagi. Piksenooled lõid puulatvadesse. Tuul vihises üle lagendiku ja sild hakkas murenema. Naine jõudis loitsu lõppu. Ta kahtles hetkeks ja ütles sihtkohaks: „Välja!"

Torm aina tõusis, sinine sähvatus pimestas ning maa kadus jalge alt. „Tuttav tunne," mõtles Tea, „Hea seegi." Kõuekõminast kasvas välja vaikus. Kõrgete löövide alt oli kuulda tuule ulgumist.

„Tere tulemast tagasi."

Rännukivi sinine valgus kumas savikatedraali seintelt vastu. Tea ja Ra ees seisis nende õpetaja. Alkeemik naeris ja plaksutas.

„Ma ei ole veel sellist põgenemist õpilaste poolt näinud!"

Tea tundis end petetuna. Ta surus maha enda viha ning esitas küsimuse: „Kus me olime?"

„Te olite rännukivi sees. Ammusel ajal, kui mina veel noor olin, ehitasime alkeemikute nõukogu helgemate peadega ideaalse eksamineerimissüsteemi. Maailm, kus inimesed saavad riskivabalt lahendada reaalseid probleeme ja meie näeme täpselt, kuidas nad enda teadmisi kasutavad. Mõni maagiline võte tekitab helisid, teised näitavad meile midagi, mida reaalselt ei ole. Kui me kõigi aistingute jaoks eraldi maagiat kasutame, siis saame ideaalse kasutajaliidese. See on vaid illusioon, kuid see töötab nagu maailm omaette - võltsreaalsus. Te olete ainsad mitmete aastate jooksul, kes said sellest mängust jagu mängimata. Õnnitlen teid, verivärsked alkeemikud."

Ra tundis tuikavat valu enda käel ning märkas, et ka Tea käsivarrele oli tekkinud sinine märk – tõelise alkeemiku tähis.