5.1 Järjend

JÄRJEND

Seni on meil programmides erinevate andmete hoidmiseks olnud eraldi muutujad. Paljude ülesannete puhul on vägagi mõistlik teatud andmeid koos käsitleda - teatud andmestruktuuridena. Erinevates programmeerimiskeeltes on selle jaoks mõnevõrra erinevad vahendid. Ka konkreetses programmeerimiskeeles võib olla mitmeid võimalusi, on Pythoniski. Pythoni üheks põhiliseks andmestruktuuriks on **järjend** ehk **list** (ingl *list*). Siinkohal tuleb märkida, et erinevate programmeerimiskeelte ja isegi erinevate materjalide puhul võib terminoloogia erineda. Meie peame siin *järjendit* ja *listi* sünonüümideks, mujal ei pruugi see nii olla.

Järjend koosneb elementidest, mis loetletakse nurksulgudes üksteisest komadega eraldatutena. Näiteks

```
nädalapäevad = ["esmaspäev", "teisipäev", "kolmapäev", "neljapäev",
"reede", "laupäev", "pühapäev"]
temperatuurid = [-4, 2, 6, 4, 4]
```

Järjendis võib olla korduvaid elemente, näiteks järjendis `temperatuurid` on `4` kaks korda. Eeltoodud järjendites on esimeses ainult sõned ja teises ainult täisarvud. Tegelikult võivad Pythonis ühe järjendi elemendid olla ka erinevat tüüpi, aga tavaliselt me seda võimalust ei kasuta. Järjendeid võime liigitada nende elementide tüüpide järgi.
Muutujas `nädalapäevad` on meil nüüd siis sõnejärjend ja
muutujas `temperatuurid` täisarvujärjend.

Vaatame, mis tüüpi näitab Python ise muutuja `temperatuurid` puhul.

```
temperatuurid = [-4, 2, 6, 4, 4]
print(type(temperatuurid))
```

Ülesanne

Mis tüüpi on muutuja nädalapäevad?

```
nädalapäevad = ["esmaspäev", "teisipäev", "kolmapäev", "neljapäev", "reede",
"laupäev", "pühapäev"]
```

- [ ] ennik
- [ ] järjend
- [ ] list
- [ ] sõne

MÕNED OPERATSIOONID. INDEKSID

Järgmise programmis on demonstreeritud mõningaid järjendiga seotud operatsioone.

```
# Järjendi loomine
nädalapäevad =
["esmaspäev", "teisipäev", "kolmapäev", "neljapäev", "reede", "laupä
ev", "pühapäev"]

# Järjendi saab ekraanil väljastada
print(nädalapäevad)

# Elementide arvu (järjendi pikkuse) leidmine
nädalapäevi = len(nädalapäevad)
print('Järjendis on ' + str(nädalapäevi) + ' elementi')

# Järjendisse kuulumise kontroll
if "palgapäev" in nädalapäevad:
    print("Palgapäev on järjendis")
else:
    print("Palgapäev ei ole järjendis")

# Konkreetse elemendi väärtus indeksi (järjekorranumbri) abil
print(nädalapäevad[1])
```

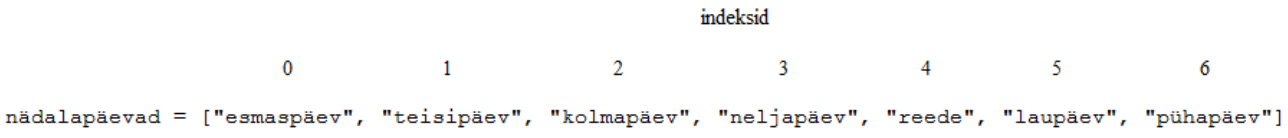
Mõnevõrra üllatav võib tunduda see, et `nädalapäevad[1]` ei anna meile esimest elementi, vaid hoopis teise. Nimelt tavatsetakse programmeerimises nummerdamist alustada nullist. Tõepoolest

```
print(nädalapäevad[0])
```

annab (argimõttes) esimese elemendi. Seega tuleb olla väga tähelepanelik, kas jutt on argine või programmeerimisest. Selguse mõttes räägime siis pigem elemendi indeksist, näiteks element indeksiga 1. Nagu juba nägime märgitakse indeks nurksulgudesse.

Tegelikult ei alga ka tavaelus nummerdamine alati ühest. Näiteks on mõnedes rongijaamades platvorm 0 (Londoni King's Cross, Cardiffi keskjaam). (Eks King's Crossi jaamas ole ka muidugi platvorm 9 3/4.)

Kuna loendamine algab nullist, siis viimase elemendi indeks on ühe võrra väiksem järjendi pikkusest, järjendi `nädalapäevad` puhul siis 6, sest elemente on 7.



Proovigem!

```
nädalapäevad = ["esmaspäev", "teisipäev", "kolmapäev", "neljapäev",
"reede", "laupäev", "pühapäev"]
print(nädalapäevad[6])
print(nädalapäevad[7])
```

Kui minna "üle piiri", kasutades sellist indeksit, millele vastavat elementi pole, siis ilmub veateade `IndexError: list index out of range`.

Ülesanne

Mis ilmub ekraanile?

```
nädalapäevad = ["esmaspäev", "teisipäev", "kolmapäev", "neljapäev", "reede",
"laupäev", "pühapäev"]
print(nädalapäevad[-1])
```

- Vali esmaspäev
- Vali laupäev
- Vali pühapäev
- Vali veateade

Pythonis on tõepoolest võimalus kasutada negatiivseid indekseid. Tegelikult me oleme seda juba kasutanud vihjes Leedu perekonnanimede ülesande juures: *Viimase tähe kontrollimiseks sobib* `nimi[-1] == "e"`.

Tookord oli tegemist sõnega, mida aga saabki käsitleda sümbolite jadana. Nii ongi väga mitmed (siiski mitte kõik) operatsioonid võimalikud nii järjendite kui sõnedega. (Tegelikult veel teistegi andmestruktuuridega, mida selles kursuses üldiselt ei käsitleta.)

VIILUTAMINE

Leedu perenimede ülesande juures oli veel kirjas, et kahe viimase tähe kontrollimiseks saab kasutada näiteks võrdlemist `nimi[-2:] == "ne"`. Tegemist on viilutamisega, mis toimib nii järjendite kui sõnede puhul ja millega saame vastavalt *alamjärjendi* või *alamsõne*. Vaatleme viilutamise võimalusi järgmise programmi abil, kus "noa all" on (antud juhul ühesümboliliste) sõnede järjend. Meelega on sama tulemuse saamiseks kasutatud erinevaid võimalusi.

```
a = ['A', 'B', 'C', 'D', 'E']
print(a[0:2])
print(a[:2])
print(a[2:5])
print(a[2:])
print(a[-2:])
```

Enne koolonit kirjutatakse indeks, millest alates tuleb elemente kopeerida, ja kooloni järele indeks, mille juures tuleb kopeerimine lõpetada (selle indeksiga element jääb tulemusest välja). Kui algav indeks kirjutamata jätta, siis alustatakse järjendi (sõne) algusest. Kui lõpetav indeks kirjutamata jätta, siis kopeeritakse kuni järjendi (sõne) lõpuni (viimane element kaasaarvatud).

Järgmises näites viilutatakse erinevaid sõnesid - nii neid, mis on muutujale väärtuseks antud, kui ka neid, mis pole.

```
sõne1 = 'Tartu'
print(sõne1[1:4])
sõne2 = sõne1
print(sõne2[-2:])
print("Väike-Maarja"[4:9])
```

Ülesanne

Mis ilmub ekraanile?

```
b = [1, 2, 3, 4]
print(b[:])
```

Vali [1, 2, 3, 4]

Vali [2, 3]

Vali Veateade

VEEL OPERATSIOONIDEST

Toome lõpetuseks tabeli, milles on nii juba mainitud kui mainimata võimalusi.

Avaldis	Väärtus	Kommentaar
len([2, 1, 4, 3, -5])	5	Elementide arv
min([2, 1, 4, 3])	1	Minimaalne element
max([2, 1, 4, 3])	4	Maksimaalne element
sum([2, 1, 4, 3])	10	Elementide summa
sorted([2, 1, 4, 3])	[1, 2, 3, 4]	Tagastab järjestatud järjendi
3 in [2, 1, 4, 3]	True	Kas on selline element?
[2, 1] + [3, 1]	[2, 1, 3, 1]	Järjendite liitmine
[2, 1, 4, 1].count(1)	2	Elemendi esinemiste arv
[1, 2, 3] == [2, 1, 3]	False	Järjekord on oluline

Katsetage neid julgesti!

Kontrollülesande 5.1 lahendamiseks pole tegelikult kogu seda arsenali vaja. Edasipidi aga võivad erinevad võimalused kasuks tulla küll.

Tabelis on järjendite liitmise tehe, mille abil saame järjendile ka vaid ühe elemendi lisada. Ühe elemendi lisamiseks saab samuti kasutada funktsiooni *append*. Järgmises näites ongi kummalegi järjendile üks element lisatud.

```
a = [5, 8]
a.append(7)
print(a)
b = [5, 8]
b += [7]
print(b)
```

Vaata ka kokkuvõtvat videot: http://www.uttv.ee/naita?id=24026

5.2 *For*-tsükkel. Funktsioon *range*

ÜKSIK ELEMENT TEISTE SEAS

Jätkame järjendi teemaga vaadeldes nüüd järjendi ühte konkreetset elementi. Võtame näiteks järjendi `a = [12, 23, 34, 45, 56]` elemendi, mille indeks on 3. Näeme, et `a[3]` (või teisiti `a[-2]`) väärtus on 45. Tegelikult võime `a[3]` (ja muidugi ka teisi elemente) küllaltki iseseisvatena käsitleda. Näiteks saame seda kasutada avaldistes või anda sellele uue väärtuse.

```
a = [12, 23, 34, 45, 56]
print(a[3])
print(type(a[3]))
print(a[3] - 32)
a[3] = 1
print(a[3])
```

Ülesanne

Mis ilmub ekraanile?

```
a = [12, 23, 34, 45, 56]
a[3] = 1
print(a[a[3]])
```

Vali 1

Vali 23

Vali Veateade

Nagu näeme, siis on indeksit võimalik arvutades leida. Oluline on vaid, et indeks oleks lubatud vahemikus.

Järjendi idee on siiski kõiki (või vähemalt mingeid) elemente koos käsitleda. Proovime järjendi elemendid järjestikustel ridadel ekraanile saada. Tegemist on selgelt tsüklilise tegevusega, mida saamegi *while*-tsüklina realiseerida.

```
a = [12, 23, 34, 45, 56]
i = 0
while i < len(a):
    print(a[i])
    i += 1
```

Kuna viimase elemendi indeks on ühe võrra väiksem kui järjendi pikkus, siis tõesti selline tsükli tingimus on sobiv. Väljastamise asemel võime ka näiteks kõik need elemendid, mille väärtus on suurem kui 30, nullida.

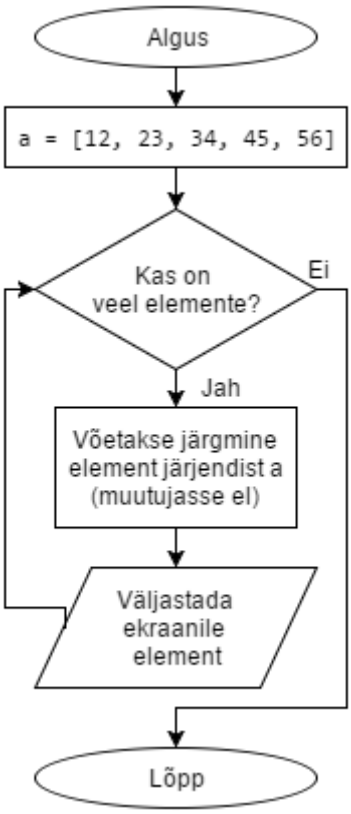
```
a = [12, 23, 34, 45, 56]
i = 0
while i < len(a):
    if a[i] > 30:
        a[i] = 0
    i += 1
print(a)
```

FOR-TSÜKKEL

Kuna väga sageli on vaja järjendiga just elementhaaval midagi ette võtta, on kasutusele võetud eriline tsükkel. *For*-tsükkel võtabki järjest igal sammul fookusesse järjendi ühe elemendi.

Elementide väljastamise saame realiseerida nüüd lühemalt.

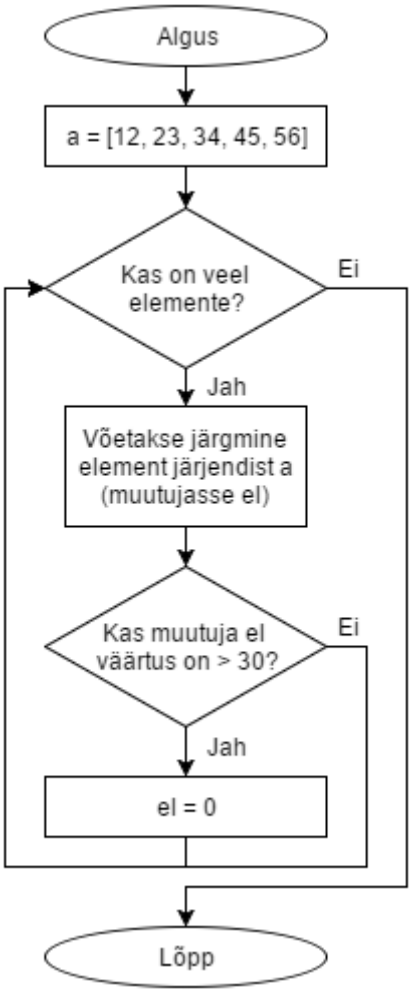
```
a = [12, 23, 34, 45, 56]
for el in a: #el asemel võib ka mingi teine nimi olla
    print(el)
```



Muutuja `el` saab igal tsükli sammul väärtuseks järjendi `a` järgmise elemendi. Laias laastus on *for*-tsükkel *while*-tsükliga sarnane. Esimesel real on päis, mis määrab, millal tsükkel lõpetatakse. Järgneb taandreaga määratud keha (ehk sisu), mida igal sammul täidetakse. Tsüklit korratakse niipalju kordi, kui järjendis on elemente.

Muutuja `el` kaudu ei saa muuta järjendi elementide väärtusi. Kui püüda teha ülaltoodud teisele *while*-tsüklile analoogi, siis nii jääb `a` muutmata.

```
a = [12, 23, 34, 45, 56]
for el in a:
    if el > 30:
        el = 0
print(a)
```



Mitmetes programmeerimiskeeltes on *for*-tsükkel teistsugune ja Pythoni *for*-tsüklile sarnast nimetatakse hoopis *for-each*-tsükliks.

RANGE

Üsna sageli on vaja selliseid tsükleid, kus tsükli muutuja väärtus teatud reeglite järgi muutub. Meil endalgi oli vaja, et näiteks `i` omandaks järjest väärtusi 0, 1, 2, 3 jne. Siin tuleb appi funktsioon `range`, mis genereerib järjendit meenutava väärtuse. Näiteks `range(4)` annab vahemiku, milles on 0, 1, 2 ja 3. Vajadusel võime ka vastava järjendi saada `list(range(4))`, aga *for*-tsükli jaoks seda vaja pole.

```
for i in range(4):
    print(i)
```

Muutuja `i` saab järjest väärtused 0, 1, 2 ja 3.

Kui argumendiks on mittepositiivne arv, siis ühtegi arvu vahemikus pole.

```
for i in range(-4):
    print(i)
```

Vahel meil tegelikult tsükli muutuja väärtust teada pole vajagi, huvitab vaid korduste arv. Paneme näiteks kilpkonna jälle ruutu joonistama

```
from turtle import *
for i in range(4):
    forward(100)
    left(90)
exitonclick()
```

Kuigi muutuja i väärtust vaja ei olnud, pidi see muutuja ikka kirjas olema.

Funktsiooni `range` saab kasutada ka 2 või 3 argumendiga. Kui on kaks argumenti, siis esimene näitab, millisest arvust alustatakse ja teine, millise arvu juures lõpetatakse (seda ennast mitte kaasates).

```
for i in range(1, 4):
    print(i)
```

Ülesanne

Millised variandid annavad 0, 1, 2, 3, 4, 5?

- ☐ range(5)
- ☐ range(6)
- ☐ range(0, 5)
- ☐ range(0, 6)
- ☐ range(0, 6, 1)

Kui teine argument ei ole esimesest suurem, siis ühtegi arvu vahemikus pole.

Kui on kolmaski argument, siis see näitab sammu, mis võib olla ka negatiivne. Nii annab `range(0, 13, 2)` arvud 0, 2, 4, 6, 8, 10, 12 ja `range(4, -5, -3)` arvud 4, 1, -2.

Muidugi saab ka erinevates kohtades muutujaid kasutada. Näiteks teeme programmi, mis küsib kasutajalt kaks täisarvu. Seejärel kirjutab ekraanile alates esimesest arvust üle ühe kõik täisarvud kuni teise arvuni (seda mitte kaasates). Ühtlasi leiab ka kõigi väljastatud arvude korrutise.

Näiteks, kui sisestatakse 4 ja 12, siis ekraanile väljastatakse: 4 6 8 10 Korrutis on 1920

Üks võimalik lahendus on selline. Korrutise puhul ei sobi algväärtuseks 0, sest korrutamine seda ei muudaks,

```
algus = int(input("Sisestage esimene arv "))
lõpp = int(input("Sisestage teine arv "))
korrutis = 1
for i in range(algus, lõpp, 2):
    print(i)
    korrutis *= i
print("Korrutis on " + str(korrutis))
```

Kontrollülesande 5.2 lahendamisel võib sellest näitest kasu olla.

5.3 Andmed failist

ANDMED FAILIST

For-tsüklit saame kasutada ka failist andmete lugemiseks. Selline võimalus annab programmidele uue mõõtme - enam me ei pea andmeid programmi sisse kirjutama või kasutajalt sisestamist küsima. Failis võivad andmed olla erineva struktuuriga, näiteks krüpteeritult. Meie piirdume, vähemalt esialgu, selliste failidega, mis on ka inimesele loetavad ja arusaadavad. Räägime lihtsatest tekstifailidest, mis koosnevad ridadest. Failide sisselugemiseks saab kasutada järgmist programmi.

```
fail = open("andmed.txt", encoding="UTF-8")
for rida in fail:
    print("Lugesin sellise rea: " + rida)
fail.close()
```

Andmed loetakse failist *andmed.txt*, mis on programmiga samas kaustas. Faili avamisel oleks võinud kasutada ka ühe argumendiga varianti `open("andmed.txt")`, aga kuna meil võib tulla tegemist täpitähtedega, täpsustame kodeeringut.

Failist saame read *for*-tsükliga järjest kätte. Muutujas `fail` pole siiski järjend vaid mõnevõrra keerukam struktuur, millega aga *for*-tsükkel kenasti hakkama saab. Tsükli kehas saame konkreetse reaga midagi peale hakata, antud juhul see lihtsalt väljastati ekraanile. Tegelikult aga saab olenevalt rea tähendusest väga huvitavaid ja kasulikke asju teha.

Olgu meil näiteks fail jooks.txt, kus on kirjas, mitu kilomeetrit tervisesportlane erinevatel kordadel jooksis. Teeme programmi, mis kõik need arvud kokku liidab.

```
fail = open("jooks.txt", encoding="UTF-8")
kokku = 0
for rida in fail:
    kokku += float(rida)
fail.close()
print("Kokku joosti " + str(kokku) + " kilomeetrit.")
```

Muidugi võib tsükli kehas ka rohkem ridu olla. Näiteks liidame kokku ainult need jooksud, mil joosti üle 10 kilomeetri.

```
fail = open("jooks.txt", encoding="UTF-8")
kokku = 0
for rida in fail:
    kilomeetreid = float(rida)
    if kilomeetreid > 10:
        kokku += kilomeetreid
fail.close()
print("Kokku joosti " + str(kokku) + " kilomeetrit.")
```

Kui tahame andmetega midagi edasi teha, siis võib olla mõistlik need järjendisse panna.

```
fail = open("jooks.txt", encoding="UTF-8")
kokku = 0
jooksud = []
for rida in fail:
    jooksud.append(float(rida))
    # 2. võimalus jooksud += [float(rida)]
fail.close()
print(jooksud[4])
```

Enne tsüklit on tehtud tühi järjend: `jooksud = []`. Tsükli igal sammul järjend pikeneb: `jooksud.append(float(rida))` või `jooksud += [float(rida)]`.

Pärast tsüklit väljastatakse järjendi element, mille indeks on 4.

Kontrollülesande 5.3 peaks saama lahendatud juba eeltoodud materjali põhjal.

KUIDAS FAILID TEKIVAD?

Eespool olevates näidetes oli fail, kust andmeid võtta, juba olemas. Selline fail võib olla tekkinud väga erineval moel. Seda, kuidas Pythoni programmiga saab andmeid faili kirjutada, vaatame edaspidi. Tekstiredaktoriga faili koostades peab jälgima, et tegemist oleks lihtsa nö *plain*-tekstiga. Salvestamisel võiks valikute korral eelistada UTF-8 kodeeringut. Muude korral võib vaja minna programmis kodeeringu muutmist.

Meil on võimalik tekstiredaktorina kasutada Thonnyt ennast. Salvestamisel tuleb lihtsalt salvestustüübina *text files* valida. Vaikimisi lähevad need failid samasse kausta, kus programmidki ja see meile sobib.

Tohutult palju võimalusi pakuvad avalikud andmebaasid, kust saame kõigepealt sobiva tabeli alla laadida ja seda siis oma programmi abil analüüsida. Vastav näide on selle materjali lõpus. Siinkohal lihtsalt mõned
lingid: http://www.stat.ee/, http://haridussilm.ee/, http://ec.europa.eu/eurostat.

PANGAAUTOMAAT RAHVUSVAHELISEKS

Failidest andmete saamisega saame täiendada pangaautomaadi programmi nii, et saaks keelt valida. Selle programmi loomisest on ka video.

Programmi tööks vajalikud failid on est.txt, eng.txt ja ger.txt.

```
from time import sleep

keel = int(input("1) Eesti 2) English 3) Deutsche "))

if keel == 2:
    failinimi = "eng.txt"
elif keel == 3:
    failinimi = "ger.txt"
else:
    failinimi = "est.txt"

readfailist = open(failinimi, encoding = "UTF-8")
read = [] #ilma reavahetusteta ridade list
for rida in readfailist: #reakaupa listist readfailist
        read = read + [rida.strip("\n")] #ilma reavahetuseta rida
lisatakse listi
        # või read.append(rida.strip("\n"))
readfailist.close() # faili ei lähe enam vaja

sisestatud_pin = ""
katseid = 3
while sisestatud_pin != "1234" and katseid > 0:
    print(read[0])
    print(read[1] + str(katseid) + read[2])
    katseid -= 1
    sisestatud_pin = input()
if sisestatud_pin == "1234":
    print(read[3])
else:
    print(read[4])
    i = 10
    while i > 0:
        print(i)
        i -= 1
        sleep(1)
```

KILPKONNA TRANSLAATOR

Failis võib olla hoopis programmitekst. Tegelikult ju meie programmide tekstid ongi lihtsalt tekstifailid. Olgu meil näiteks failis kirjas, kuidas kilpkonn peaks liikuma

```
edasi 40
paremale 90
edasi 50
tagasi 30
vasakule 90
edasi 30
```

Kirjutame sellise programmi, mis selle teksti kilpkonnale arusaadavaks tõlgib.

```
from turtle import *

# faili avamine
fail = open("kilpkonn.txt")
# faili töötlemine ja kilpkonnaga joonistamine
for rida in fail:
    osad = rida.split()  # tühikute kohal osadeks
    liikumine = osad[0] # kuidas liikuda
    arv = int(osad[1]) # kui palju
    if liikumine == "vasakule":
        left(arv)
    elif liikumine == "paremale":
        right(arv)
    elif liikumine == "edasi":
        forward(arv)
    elif liikumine == "tagasi":
        backward(arv)
    else:
        print("Ei saa sellest käsust aru!")

fail.close()
exitonclick()
```

Näeme, et kilpkonn saigi niimoodi eestikeelsetest käskudest aru. Programm oskab tõlkida (ingl *translating*)! Tegelikult nimetataksegi selliseid programme, mis programmiteksti arvutile arusaadavaks teevad *translaatoriteks*.

AVALIKUST ANDMEBAASIST

Eespool juba mainisime avalikke andmebaase. Näiteks statistikaameti kodulehel https://www.stat.ee/ on väga palju huvitavat informatsiooni. Siin saab ise andmeid valida ja sobivas formaadis salvestada. Kuigi pakutakse ka *tekstifaili*, on meile siin sobivam *tabeldieraldusega pealkirjata tekst (.csv)* või *semikooloneraldusega pealkirjata tekst (.csv)*. Mõlemal juhul tekib *csv*-fail. Formaadinimi csv on lühend sõnadest *comma-separated values* (komaga eraldatud väärtused). Kuigi vahel kasutataksegi väärtuste eraldajana tõesti koma, siis paljudel juhtudel on eraldajaks hoopis tabeldusmärk (Pythonis "\t") või semikoolon.

Näiteks võib rida failis RV031sm.csv olla selline (loomulik iive 1923. aastal). `"1923";3636`

Nüüd on vaja see rida semikooloni kohalt osadeks jaotada ja osad täisarvudena esitada. Osadeks saab jaotada funktsiooni `split` abil, mis tekitab antud juhul kahe elemendiga sõnejärjendi. Üks element on sõne `'"1923"'` ja teine `'3636'`. Tegelikult tahame mõlemal juhul saada täisarve. Kuna esimesel juhul on sõnes endas jutumärgid, siis tuleb need eemaldada näiteks funktsiooni `strip` abil.

```
readfailist = open("RV031sm.csv")
aastad = [] # järjend aastate jaoks
iibed = [] # järjend iivete jaoks
for rida in readfailist: # reakaupa listist readfailist
    realt = rida.split(';') # jaotada semikooloni kohalt
    aastad += [int(realt[0].strip('"'))] # aastate järjendisse
juurde
    # või aastad.append(int(realt[0].strip('"')))
    iibed += [int(realt[1])] # iivete järjendisse juurde
    # või iibed.append(int(realt[1]))
readfailist.close() # faili ei lähe enam vaja
print(aastad)
print(iibed)
```

Järgnevas videos on seda temaatikat põhjalikumalt näidatud:
https://youtu.be/IrY19oHMyjM

5.4 Silmaring: krüpteerimine

Autor: Kristjan Krips

SISSEJUHATUS

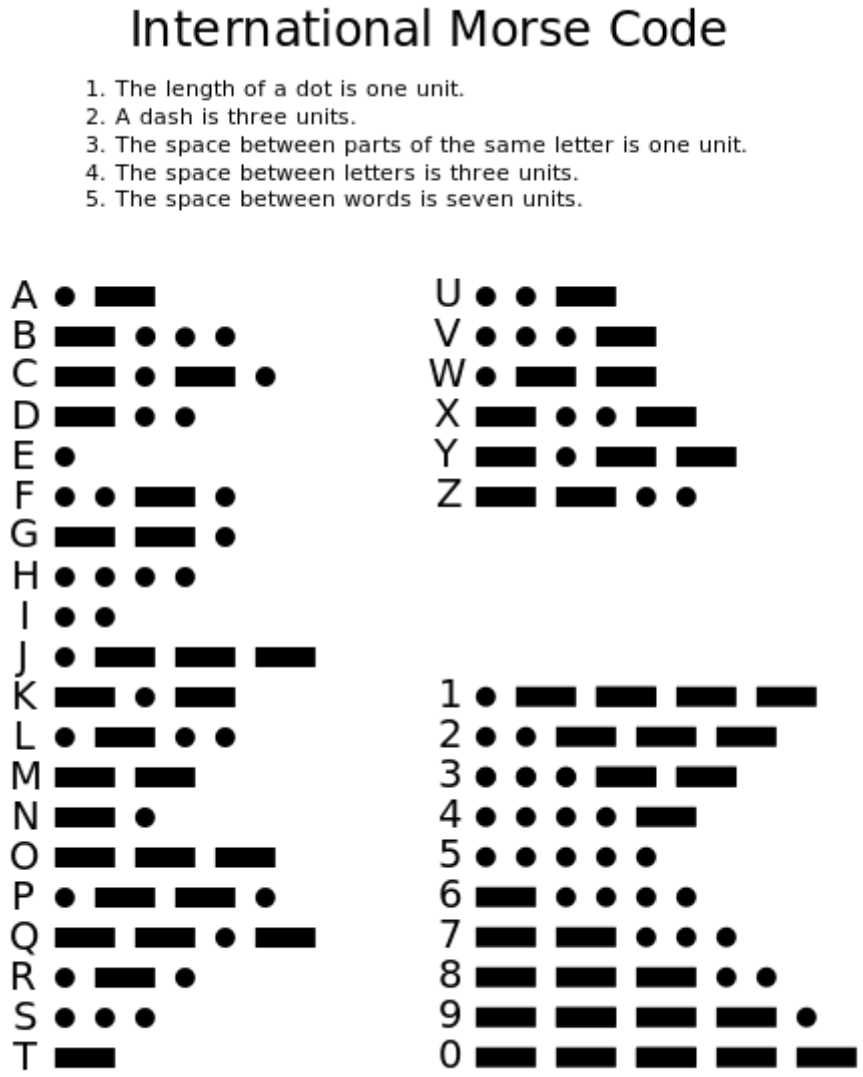
Selle nädala silmaringi materjalis teeme lühikokkuvõtte sellest, mis on **krüpteerimine** ja miks seda vaja läheb. Ajaloo osas mainisime masinat nimega Enigma, mida kasutati Teise maailmasõja ajal militaarsõnumite kaitsmiseks pealtkuulajate eest. Enigma oli masin, mis võimaldas andmeid krüpteerida ning seega neid kaitsta kolmandate osapoolte eest (näiteks sakslaste andmete kaitsmist inglaste eest). Enigma muutis masinasse sisestatud andmeid nii, et masinast väljuvad andmed olid kolmandale osapoolele loetamatud.

Enigma. Allikas

KODEERIMINE

On oluline tähele panna, et Enigma puhul polnud tegu andmete lihtsa **kodeerimisega**, sest kodeeritud andmeid on kerge taastada. Lihtne kodeerimise näide on **morsekood**, mis muudab kindla eeskirja alusel andmeid ning võimaldab neid selle sama eeskirja alusel taastada.

Krüpteerimine erineb kodeerimisest selle poolest, et andmete töötlemisel kasutatakse **võtit**, mistõttu kolmas osapool ei saa ilma võtit omamata krüpteeritud andmetest esialgset taastada.



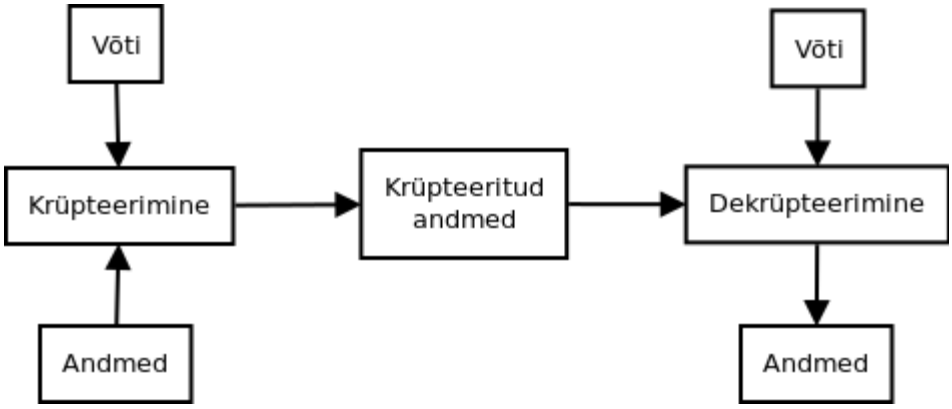
Morse on näide kodeerimisest. Allikas

KRÜPTEERIMINE

Tänapäeval on krüpteerimine põhiline viis digitaalsete andmete kaitsmiseks. Krüpteerimise abil kaitstakse telefonikõnesid pealtkuulamise eest ning meditsiiniandmeid ja ärisaladusi lekkimise eest. Muuhulgas kasutatakse krüpteerimist tasuliste telekanalite edastamisel, wifi võrgu kaitsmisel, HTTPS protokolli kasutavate veebilehtedega suhtlemisel, digiallkirjastamisel ja e-hääletamisel. Krüpteeritud ühenduse kasutamine on üks oluline meetod turvalise internetipanga teenuse pakkumisel. Kuna krüpteerimine mõjutab meid iga päev, siis võib tekkida küsimus, kuidas ikkagi krüpteerimine toimib ning kas ründajal on võimalik krüpteeritud sõnumeid lahti murda.

SÜMMEETRILINE KRÜPTEERIMINE

Kõige lihtsam on krüpteerimist ette kujutada nii, et on olemas **krüpteerimisalgoritm** (eeskiri), mis saab sisendiks **andmed** ja **krüpteerimisvõtme** ning annab väljundiks **krüpteeritud andmed**. Nende krüpteeritud andmete taastamiseks on vaja kasutada **dekrüpteerimise** (lahtikrüpteerimise) **võtit** ning ilma vastava võtmeta pole võimalik andmeid esialgsel kujul taastada. Juhul kui krüpteerimisvõti ja dekrüpteerimisvõti on samad, siis on tegu sümmeetrilise krüpteerimisalgoritmiga. Niisuguseid algoritme kasutatakse suurte andmehulkade krüpteerimiseks, hea näide on kõvakettal või mälupulgal olevate andmete kaitsmine.



Krüpteerimine ja dekrüpteerimine

Nüüd võib tekkida küsimus, et mis on see **võti**, mida krüpteerimisel ja dekrüpteerimisel kasutatakse. Vastav võti on mõnes mõttes sarnane tavalise lukuvõtmega — võtit tuleb valvata, et sellest koopiat ei tehtaks, ja võtit tuleb kaitsta kadumise eest, et võõrad võtit kasutada ei saaks. Lisaks sellele tuleb võtit enda jaoks säilitada, et tagada juurdepääs võtmega suletud hoiukohtadele või ruumidele. Oluline on märkida, et ka krüpteeritud andmetele on võimalik juurde pääseda ainult vastavat võtit kasutades. Võti on kahendsüsteemis arv ehk siis ühtedest ja nullidest koosnev jada. Tänapäevaste sümmeetriliste krüpteerimissüsteemide korral on võtmed enamasti 128-bitised või 256-bitised ehk siis ühtede ja nullide jadad pikkusega 128 või 256 märki. On väga oluline, et

krüpteerimisel kasutatav võti oleks juhuslikult genereeritud, sest krüpteerimise tugevus sõltub nii krüpteerimisalgoritmi kui ka võtme tugevusest.

Samas toob juhuslike bitijadade kasutamise nõue endaga kaasa kaks probleemi:

1. Esimene probleem seisneb selles, et juhuslike võtmete genereerimine ei ole lihtne, sest arvutid ei suuda päris juhuslikke arve genereerida.
2. Teine probleem on seotud sellega, et inimesed ei ole võimelised meelde jätma pikki nende jaoks seosetuid bitijadasid.

Seetõttu krüpteeritakse võti enamasti **parooliga**, mida on inimesel lihtsam meelde jätta, ning krüpteeritud võti salvestatakse kas eraldi failina, lisatakse krüpteeritud faili päisesse või hoitakse eraldiseisval kiibil.

Andmete krüpteerimiseks on vaja valida sobiv **krüptosüsteem**. Krüptosüsteem koosneb:

- võtme genereerimise algoritmist,
- andmete kaitsmiseks mõeldud krüpteerimisalgoritmist ja
- andmete taastamiseks mõeldud dekrüpteerimisalgoritmist.

Krüptosüsteemi valikul tuleks lähtuda selle **turvalisusest**. Tänapäeval on sümmeetrilise krüpteerimise korral standardlahenduseks AES-i (Advanced Encryption Standard) kasutamine. AES krüpteerib andmeid 128-bitiste plokkide kaupa, täpsemat infot AES-i toimimise kohta leiate vastavalt Wikipedia lehelt: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

AES-i pakutavat krüpteeringut peetakse murdmatuks. See tähendab seda, et isegi superarvutid ei suuda AES-i abil krüpteeritud andmeid lahti murda, sest leitud pole ühtegi olulist krüptograafilist viga ning kõikide erinevate võtmete proovimiseks kuluks miljoneid aastaid. See kehtib ainult vastava süsteemi korrektsel kasutamisel, mis tähendab seda, et võti on genereeritud juhuslikult ning see ei leki. Juhul kui AES võtme kaitsmiseks kasutatav **parool on lühike**, näiteks kaheksa sümboli pikkune, siis on võimalik kõikvõimalike lühikeste paroolide proovimise teel vastav parool üsna kiiresti üles leida ning seega andmed dekrüpteerida. Samuti on vaikimisi eelduseks, et krüpteerimistarkvara on kirjutatud **spetsialistide** poolt, et see ei lekitaks infot krüpteeritavate andmete või võtmete kohta. Krüptoalgoritmide programmeerimine ei ole lihtne, sest info võib lekkida väga erinevatest kohtadest, näiteks protsessori energiatarbimise kaudu.

SÜMMEETRILISE KRÜPTEERIMISEGA SEOTUD PROBLEEMID

Sümmeetriline krüpteerimine võimaldab andmeid kaitsta, kuid probleemiks osutub krüpteeritud andmete **jagamine teiste osapooltega**. Kuna sümmeetrilise krüpteerimise korral kasutatakse krüpteerimiseks ja dekrüpteerimiseks sama võtit, siis oleks krüpteeritud andmete jagamisel **vaja edastada ka võti**.

Nüüd tekib küsimus, et kui turvaline on võtme jagamine ja edastamine. On oluline tähele panna, et kui võtit ja krüpteeritud andmeid edastatakse koos, siis saab pealtkuulaja juurdepääsu krüpteeritud andmetele ning seega poleks andmed kaitstud. Seega oleks võtme edastamiseks vaja **turvatud sidekanalit**. Samas, kui suhtluspartnerite vahel on juba olemas turvatud sidekanal, siis ei olegi enam vajadust andmeid eraldi krüpteerida.

Mida teha olukorras, kus suhtluspartnerite vahel ei ole turvatud sidekanalit ning nad peavad edastama konfidentsiaalseid andmeid? Võimalik on kasutada kahte **erinevat ebaturvalist sidekanalit** (näiteks edastatakse parool käsitsi ümbrikus), lootuses, et pealtkuulaja ei saa mõlemat kanalit pealt kuulata ning seetõttu ei saa juurdepääsu kas krüpteeritud andmetele või siis võtmele. Samas ei paku niisugune süsteem suhtluspartneritele kindlust, et nende saadetud konfidentsiaalsed andmed jäävad salajaseks.

Isegi kui õnnestub leida turvaline viis võtme edastamiseks, siis tekib teine probleem. Nimelt on vaja iga sõnumi jaoks genereerida uus võti, sest juhul kui suhtluspartner lekitab võtme (näiteks portaal ei turva paroole piisavalt hästi), siis saaksid kolmandad osapooled järgmisi sõnumeid lugeda. See omakorda tekitab vajaduse **turvalise võtmehalduse** jaoks.

Neid probleeme saab lahendada **avaliku võtme krüptograafia ehk asümmeetrilise krüptograafia** abil.

Allikas

ASÜMMEETRILINE KRÜPTEERIMINE

Asümmeetrilise krüpteerimise ehk **avaliku võtme krüpteerimise** korral kasutatakse kahte võtit — **avalikku ja salajast võtit**. Need võtmed genereeritakse korraga ning nad on omavahel seotud nii, et avalik võti ei lekita informatsiooni salajase võtme kohta. Võtmete genereerimine peab olema **juhuslik**.

Nii nagu nimigi ütleb, on üks võti avalik ning seda võib kõigiga jagada. Samas tuleb salajast võtit kaitsta lekkimise eest. Asümmeetrilise krüpteerimise korral on kõigil osapooltel enda **võtmepaarid**. Näiteks on kõigil **Eesti ID-kaardi** omanikel kiibis võtmepaar. Samuti on kõigil HTTPS pakkuvatel veebilehtedel oma võtmepaar.

Avalikku võtit kasutatakse andmete krüpteerimiseks ning vastavat salajast võtit kasutatakse andmete dekrüpteerimiseks. Seega võimaldab asümmeetrilise krüptograafia kasutamine lihtsamat võtmevahetust, sest avalikku võtit saab edastada ebaturvalise sidekanali abil. Kui on vaja suhtluspartnerile saata konfidentsiaalseid andmeid, siis kõigepealt on vaja küsida suhtluspartneri avalikku võtit ning siis saab seda kasutada, et

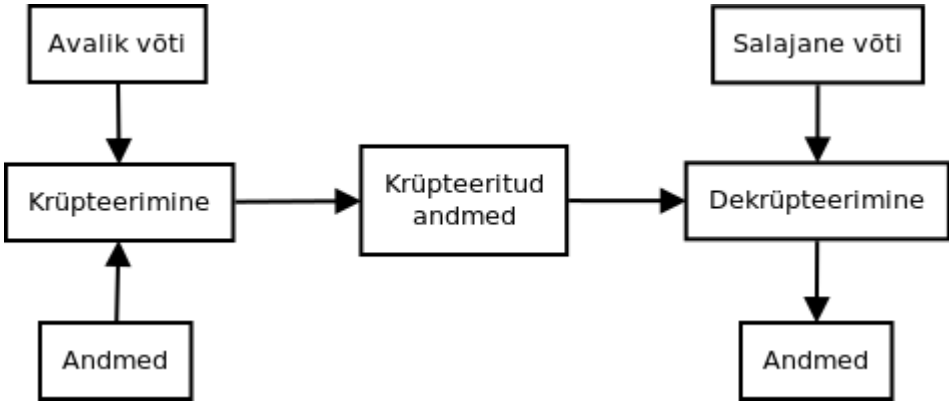
konfidentsiaalseid andmeid krüpteerida. Kuna krüpteerimiseks kasutati suhtluspartneri avalikku võtit, siis saab vastavalt krüpteeritud andmeid dekrüpteerida ainult see sama suhtluspartner, sest ainult temal on juurdepääs sobivale salajasele võtmele.

Selgituseks võtame näite, kus Kristjan tahab Taunole saata krüpteeritult salajast faili. Selleks peab:

1. Tauno jagama Kristjaniga enda avalikku võtit. Näiteks on ID-kaardi kiibis võtmepaar (avalik ja privaatne võti) ja Tauno jagab Kristjaniga ID-kaardi avalikku võtit. See pole probleem, sest ID-kaardi avalikud võtmed ongi vabalt jagatavad.
2. Seejärel Kristjan krüpteerib faili kasutades Tauno ID-kaardi avalikku võtit ning saadab krüpteeritud faili Taunole.
3. Nüüd ei jää Taunol muud üle, kui kasutada ID-kaardi kiibil olevat privaatset võtit ja saabki esialgsele failile kenasti ligi.

Tegelikult küll kasutatakse suuremate andmete krüpteerimiseks alati sümmeetrilist krüptograafiat ning seega selles olukorras **hübriidsüsteemi**, kus genereeritakse sümmeetriline võti, sellega krüpteeritakse fail ja võti ise krüpteeritakse siis vastava avaliku võtmega.

Juhul kui aga võtmeid kasutatakse vastupidi, siis on tegemist **digiallkirjastamisega**. Sellisel juhul krüpteeritakse andmed salajase võtmega ning seda saab teha ainult salajase võtme omanik. Digiallkirja kontrollimiseks kasutatakse vastavat avalikku võtit ning seda saavad teha kõik, kellel on juurdepääs vastavale avalikule võtmele.



Krüpteerimine ja dekrüpteerimine salajase ja avaliku võtmega

5.5 Lugu: alkeemia IV

LUGU: ALKEEMIA IV

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

Ra jälgis tumedaid pilvi, mis tema silme eest mööda triivisid, ning mõtles ähvardavale ilmale. Ta ei jõudnud kaasa võtta veekindlat mantlit, rääkimata toidumoonast. Noormehe mõttekäiku segas karjatus, mis sundis teda pead pöörama. Tema sarviline savist kaaslane vaatas alla Tea poole, kes lamas kivisel mäeserval ning üritas härga eemale peletada.

"Kõrval," pobises Ra ning loom sammus aeglaselt tema juurde, mille peale mees otsustas sõna otseses mõttes härjal sarvist haarata ning jalule tõusta. Jalule saamine toimus lihtsalt, kuid jalul püsimine oli märksa raskem. Õhk oli hõre, pea käis ringi ning hetkeks tundus nagu tuul üritaks teda uuesti maha suruda.

Tea oli selleks ajaks jõudnud enda jaki liivast puhtaks kloppida ning ümbrust uurida. Nähes, et tema määratud teekaaslane suutis samuti püsti seista, astus neiu lähemale ning tutvustas ennast. "Tundub, et edasi lähme me koos. Mina olen Tea."

"Ra," pobises näost kahvatu mees enda liikuvale savikamakale toetudes. Ta kogus end hetkeks ning ajas selja sirgu. "Me peaksime liikuma hakkama. Taevas lubab tormi ning meil ei ole peavarju ega korralike reisiriideid. Arvan, et kui kohe liikuma hakkame, siis jõuame õhtuks mäe jalamile."

Tea nõustus ning koos hakati teed mööda mäeküljelt laskuma. Tea ja tema kotkas läksid ees ning Ra üritas enda härga üle kivide talutada.

Poole päeva jooksul ei jõudnud surmväsinud ja läbimärjad noored kuigi palju enda eesmärgile lähemale. Viimases hädas leidis Tea enda kotka abil mõõduka koopasuu, mis tundus kutsuvam, kui kaljuserval tormi trotsimine. Käänulisest käigust paistis õrna valgust. Tea ei kahelnud hetkegi ning jooksis julgelt uudistama.

"Kes seal on!?" hüüdis hirmunud hääl kaugemalt käigust. Hääle omaniku tõrvikutuli kumas seintele ning praksus kutsuvalt.

"Oleme väsinud rändurid, kes on lootusetult eksinud! Ehk lubate meil endaga liituda?" vastas Tea enne, kui Ra pähe torkasid mõtted röövlitest, kes mäe kõhus võivad redutada, või mäekollidest, kes koobastes saaki varitsevad.

"Rändurid?" üllatus võõras, "Neid ei käi siin just palju. Hea küll, astuge aga valguse kätte. Varjudes pole just kõige ohutum."

Ra üritas paari kiire käeviipe ja näoilmega Tead veenda ka enda loomale ootamiskäsku andma, sest mõistagi ei ole kõige lihtsam seletada, miks täiesti tavalistel ränduritel savist kaaslased on. Tüdruk oli talurahva ebausuga alkeemikutesse mitmel korral kokku puutunud ning nõustus.

Tunneli lõpus seisis mõlkis kiivriga nahkturvises mees, kelle selja taga kõrgus suurtest kividest laotud müür. "Tulge aga. Aega ei tasu meil kaotada. Varsti on õhtu käes ja siis ei tasu pimedas ringi luusida."

Valvur avas pehkinud puitukse ning lasi teekäijad läbi müüri. Nende ees avanes kummaline vaatepilt. Avaras koopas peitus terve linn: Paarsada kivist majakest, kividega piiratud mägikitsede tarandikud, inimesed, kes majade vahel sagisid, ja lugematu arv kustutatud laternaid ning pooleldi põlenud küünlaid. Turvises mees juhatas Ra ja Tea linnavanema majani, kes noori lahkelt sisse kutsus. Ka tema ja ta naine kinnitasid, et väljas ei ole sel kellaajal enam ohutu ning tasub sees püsida. Tea küsimustele vastuseks ütles ta: "Varjud."

"Mis mõttes varjud?" päris ka Ra tõstes pilku seene- ja vetikahautiselt, mida neile kivist kaussides pakuti.

"Nii me neid siin kutsume. Ööajal ronivad koopaurgudest välja jubedad elajad, kes varjudes konutavad ja loomi karjast ära viivad. Ainus asi, mis neid eemal hoiab, on tuleleek." "Miks kõik küünlad ja laternad siis praegu kustutatud on?" tegi Tea kummalise tähelepaneku.

Hallipäine mees kratsis kukalt: "Meil on küll väga head küünlad, mis öö läbi põlevad, kuid me ei suuda neid piisavalt kiiresti valmistada, et neid kogu aeg saaks põletada. Lisaks on meil abimees." Ta naeratas nukralt ja näitas aknast välja.

Linna keskel seisis puidust ehitatud hernehirmutise sarnane tegelane. Valvur kallas tema küljel rippuvasse lähkrisse lambiõli ning pistis tema õlgedest juustesse tõrvikuleegi. Kuiv hein lõi lõõmama, kuid edasi ei levinud. Vaikselt hakkasid puust mehe jalad ja käed liikuma, ning ta asus laternaid süütama. Võttis okstest sõrmedega pealaelt tuld ning pistis klaasist kastidesse.

Tea surus enda näo vastu aknaklaasi ja jälgis imestusega huvitava alkeemiaseadeldise tööd ning üritas seda mõtetes lahti võtta. Ra uudistas üle Tea õla ning märkas midagi huvitavat. Hernehirmutis süütas esimese maja juures kõik laternad ning läks siis pikkade aeglaste sammudega järgmise juurde. Küünalde poole ta isegi ei vaadanud.

"Miks ta küünlaid ei süüta?" küsis Ra.

Külavanema naine vastas naeratusega: "Ta süütab alati kõige esimesena laternad ning alles siis jõuab küünalde ja tõrvikuteni."

Midagi oli selle meistriteose juures valesti. Kui puitmees annab tuld laternatele esimesel ringil ning tuleb küünalde jaoks tagasi, siis kaotab ta väärtuslikku aega puhtalt majade vahel liikumisel. Tundus, et Ral oli õigus, sest Tea tardus paigale ning jälgis kaugemal seisvate

majade katuseid, kus laternate asemel olid vaid süütamata küünlad. Kõrgel tänava kohal sagisid väiksed elajad, kes lõhkusid katusekive ja ringi hüppasid. Ra tõmbas Tea kõrvale ning rääkis talle kiirelt, miks hernehirmutise tööring ei tööta ning tüdruku silmis lõi põlema mõte. „Muidugi!“ hõikas ta, „See on vale tsükkel! Selline käsuring töötaks ühe maja puhul, kuid terve linna jaoks on see tunduvalt aeglasem.“ Tea lõpetas lause ning sööstis uksest välja. Linnavanem jõudis vaid kraaksatada midagi ohu kohta enne, kui Ra enda kaaslasele järgnes.

Nad jooksid puitmehe poole ning Tea hüüdis: "Lõpeta!" Leegid hernehirmutise pealael kustusid ning ta jäi seisma. Tea haaras õlgedega täidetud pea seest pabersõõri ning tegi söetükiga kiired parandused. Ta muutis algse tsükli olemust nii, et ta võtaks arvesse iga maja ning süütaks neis kõik laternad ja küünlad enne, kui edasi läheb. Naine asetas paberilipiku tagasi oma kohale ning mõistis, et neil ei ole tuld, millega kummaline kratt uuesti tööle panna, ja katuräästa küünaldeni ei ole enam aega ronida, sest märkamatult olid tema ja Ra sisse piiranud sajad varjuolendid, kes lõksutasid teravate hammastega täidetud lõugu. Ring nende ümber tõmbus aina kokku ja silme eest hakkas mustaks minema. Viimases hädas mõistis Ra, et tule tegemiseks pole alati taelakarpi tarvis ning karjus täiest kõrist: "SIIA!"

Pisikesed olendid peatusid hetkeks kohkunult, kui koopaseintelt hakkas kajama metallist sõrgade plaginat. Savist härg tormas raksatusega läbi pehkinud ukse ning ajas laiali näljaste varjuelajate armee. Ra naeris kergendusest ning andis enda lemmiku käsu: "Tuld!"

5.6 Viienda nädala kontrollülesanded 5.1, 5.2, 5.3

Viiendal nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks vähemalt üks järgmistest ülesannetest, kas 5.4a, 5.4b või 5.4c (võib ka kaks või kolm lahendada). Lahendused tuleb esitada Moodle'is, kus need kontrollitakse automaatselt. Moodle'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

Kui teile tundub, et automaatkontroll töötab ebakorrektselt, siis palun kirjutage aadressil proge@ut.ee.

Kontrollülesanne 5.1. Mootorrattad

2016. aastal registreeriti Eestis 1145 uut mootorratast (http://www.stat.ee/34654). Registeeritud mootorrataste andmed on kuude kaupa failis mootorrattad.txt, kus esimesel real on jaanuaris registreeritud tsiklite arv, teisel real veebruaris registreeritud tsiklite arv jne. Faili võite salvestada siit või koostada ise mõne tekstiredaktoriga.

Koostada programm, mis

- loeb failist registreeritud mootorrataste andmed kuude järgi järjendisse;
  - Failist järjendisse saab lugeda järgmise programmijupi abil:

```
fail = open("mootorrattad.txt", encoding="UTF-8")

mootorrattad = []
for rida in fail:
    mootorrattad.append(int(rida)) # 2. võimalus mootorrattad +=
[int(rida)]
```

Viide: https://courses.cs.ut.ee/2016/eprogalkool/fall/Main/Andmedfailist

- küsib kasutajalt täisarvu, mis tähistab ühe kuu järjekorranumbrit (jaanuar 1, veebruar 2 jne);
- väljastab mitu uut mootorratast sel kuul registreeriti.

Näited programmi tööst:

```
>>> %Run yl5.1.py

  Palun sisestage mitmes kuu: 3
  3. kuul registreeriti 107 uut mootorratast.

>>> %Run yl5.1.py

  Palun sisestage mitmes kuu: 7
  7. kuul registreeriti 104 uut mootorratast.

>>>
```

Kontrollülesanne 5.2. Laikimine (for-tsükliga)

Kaisa tegi Facebooki postituse. Ta märkas, et iga minuti järel laikisid ta sõbrad tema postitust nii, et esimesel minutil kogus postitus 1 laigi, teisel minutil ei laikinud keegi, kolmandal minutil 3 laiki, neljandal minutil mitte ühtegi laiki, viiendal minutil 5 laiki jne.

Koostada programm, mis

- küsib kasutajalt minutite arvu (mittenegatiivne täisarv);
- arvutab *for*-tsükli ja funktsiooni *range()* abil postituse laikide koguarvu;
- väljastab saadud laikide arvu ekraanile.

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

Näited programmi tööst:

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

  Sisestage minutite arv: 7
  Laikide koguarv on 16.

>>>

>>> %Run yl3.2.py

  Sisestage minutite arv: 8
  Laikide koguarv on 16.

>>>
```

Tegemist on ülesande 3.2 variandiga for-tsükli jaoks.

Kontrollülesanne 5.3. Sissetulekud

Failis *konto.txt* on kirjas ujukomaarvudena pangakonto tehingud (kus positiivsed arvud on sissetulekud ja negatiivsed arvud on väljaminekud). Iga arv on eraldi real. Näitefaili võite salvestada siit või koostada ise mõne tekstiredaktoriga (kasvõi Thonnyga). Tekstifaili kasutamiseks programmi sees peab fail asuma programmifailiga samas kaustas.

Koostada programm, mis

- loeb failist nimega *konto.txt* andmed;
- väljastab ekraanile kõik sissetulekud ehk failist leitud positiivsed arvud. Iga arv peab olema eraldi real ja positiivsete arvude omavaheline järjekord peab jääma samaks nagu failis.

Näide programmi tööst:

Näiteks antud näitefaili *konto.txt* puhul peab ekraanile ilmuma

```
>>> %Run yl5.3.py
  100
  18.67
  86.23
  0.03
  531.67
>>>
```

5.7 Viienda nädala kontrollülesanded 5.4abc

Viiendal nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks vähemalt üks järgmistest ülesannetest, kas 5.4a, 5.4b või 5.4c (võib ka kaks või kolm lahendada). Lahendused tuleb esitada Moodle'is, kus need kontrollitakse automaatselt. Moodle'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

Kui teile tundub, et automaatkontroll töötab ebakorrektselt, siis palun kirjutage aadressil proge@ut.ee.

Järgmisest kolmest ülesandest (5.4a, 5.4b, 5.4c) tuleb lahendada vähemalt üks.

Kontrollülesanne 5.4a Jukebox

Ada tahab valida plaadiautomaadist laulu ja uurib, milliseid laule masin mängib. Muusikapalad on kirjas failis, kus iga laul on eraldi real.

Programmi testimiseks kasutatakse järgmisi faile, mida võite salvestada või koostada ise mõne tekstiredaktoriga (nt Notepad):

- jukebox.txt;
- 80ndad.txt;
- eesti_muusika.txt;
- edm.txt.

Koostada programm, mis

- küsib kasutajalt failinime (kasutaja sisestab failinime koos laiendiga, nt *jukebox.txt*);
- loeb sisestatud nimega failist andmed;
- näitab kõiki laule koos järjekorranumbritega (alates 1);
- küsib kasutajalt, mitmendat laulu ta soovib (kasutaja sisestab alati täisarvu);
- väljastab ekraanile vastavalt valitud arvule muusikapala

Näide programmi tööst:

Näiteks antud näitefaili *jukebox.txt* puhul peab ekraanile ilmuma

```
>>> %Run yl5.4a.py

  Palun sisestage failinimi: jukebox.txt
  Muusikapalade valik:
  1. Journey - Don't Stop Believin'
  2. Judas Priest - Living After Midnight
  3. Aerosmith - Dream On
  4. Deep Purple - Smoke On The Water
  5. Free - All Right Now
  6. Black Sabbath - Paranoid
  7. Van Morrison - Brown Eyed Girl
  8. AC/DC - Back In Black
  9. Led Zeppelin - Stairway To Heaven
  10. ZZ Top - Sharp Dressed Man

  Valige laulu järjekorranumber: 7
  Mängitav muusikapala on Van Morrison - Brown Eyed Girl.

>>>
```

Kontrollülesanne 5.4b Loomulik iive

Loomulik iive on elussündide arvu ja surmajuhtude arvu vahe. Failis *synnid.txt* on esitatud Eesti eelmise aasta sündide registreerimisandmed kuude lõikes (http://www.stat.ee/34270). Failis *surmad.txt* on kirjas Eesti eelmise aasta surmajuhtude registreerimise andmed (http://www.stat.ee/34271).

Programmi testimiseks kasutatakse lisaks ka järgmisi faile, mida võite salvestada või koostada ise mõne tekstiredaktoriga (nt Notepad):

- synnid_1.txt ja surmad_1.txt;
- synnid_2.txt ja surmad_2.txt;
- synnid_3.txt ja surmad_3.txt;
- synnid_4.txt ja surmad_4.txt.

Kirjutada programm, mis

- loeb failist synnid.txt sündide arvud kuude kaupa järjendisse nii, et esimene element on jaanuari kuu sündide arv, teine element on veebruari sündide arv jne;
- loeb failist surmad.txt surmade arvud kuude kaupa järjendisse nii, et esimene element on jaanuari kuu surmade arv, teine element on veebruari surmade arv jne;
- koostab loodud järjendite põhjal järjendi, kus elementideks on vastava kuu loomulik iive;
- väljastab ekraanile loomuliku iibe järjendi;
- väljastab kuu numbrid (jaanuar 1, veebruar 2 jne), mille korral oli iive positiivne.

Näide programmi tööst:

```
>>> %Run yl5.4b.py
  [-319.0, -264.0, -83.0, -102.0, -17.0, 65.0, 81.0, 76.0, -58.0, -222.0, -217.0, -311.0]
  2016. aasta positiivse iibega kuu numbrid:
  6
  7
  8
>>>
```

Kontrollülesanne 5.4c Tahvli juurde

Mõned õpetajad on tavatsenud õpilasi tahvli juurde vastama kutsuda kuupäeva järgi vastavalt õpilaste nimekirjale. Näiteks 4. kuupäeval tuleb esimesena vastama nimekirjas 4. olev õpilane. Failis *nimekiri.txt* on õpilaste nimed, igaüks eraldi real. Üks selline, mis on genereeritud leheküljel http://random-name-generator.info/, on siin. Võite ise koostada ka teistsuguse nimekirja.

Koostada programm, mis

- küsib failinime (eeldame, et kasutaja sisestatud nimega fail leidub ja seal on vähemalt 31 nime);
- loeb sisestatud nimega failist andmed;
- väljastab vastavalt tänasele kuupäevale õpilase nime, kes peab vastama tulema.

Programm peab tänase kuupäeva leidma automaatselt, aluseks saab võtta järgmise näite:

```
from datetime import *
print(datetime.now().day)
```

Näide programmi tööst:

```
>>> %Run yl5.4c.py
  Sisestage failinimi: nimekiri.txt
  Vastama tuleb: Joosep Toots
>>>
```