

7.1 Lugemine veebist

VEEBIS ON FAIL

Juba mitu nädalat tagasi tutvusime võimalusega oma arvutis olevast failist andmeid kätte saada. Võimalik on aga, et huvitav info on hoopis veebis. Pythoniga ei ole veebist lugemine eriti raske. Kasutada saab käsku *urlopen*, mis on vaja eelnevalt importida moodulist *urllib.request*.

Kui meid huvitav info (antud juhul Juhan Liivi luuletuse "Ta lendab mesipuu poole" 1. salm) on aadressil <https://courses.cs.ut.ee/2015/eprogalused/uploads/Main/mesipuu.txt>, siis saame selle kätte järgmise programmiga.

NB! Kui ülaltoodud aadressilt andmeid ei saa kätte (nt Macide kasutajad), siis palun proovida <http://kodu.ut.ee/~eno/mooc/mesipuu.txt>.

```
from urllib.request import urlopen

failVeebis =
urlopen("https://courses.cs.ut.ee/2015/eprogalused/uploads/Main/mesipuu.txt")
baidid = failVeebis.read() # kogu fail baitidena
tekst = baidid.decode() # baitidest saab sõne
failVeebis.close()
print(tekst)
```

Hetkel käsitlesime kogu teksti tervikuna, algul baitidena, hiljem ühe sõnena. Kuna selles sõnes on ka reavahetused, siis kuvati sõne mitme reana. Vahel on aga vaja ridasid eraldi käsitleda ja siis on abiks funktsioon *splitlines*, mis sõne reavahetuse kohtadelt järjendisse "lõhub".

```
from urllib.request import urlopen

failVeebis =
urlopen("https://courses.cs.ut.ee/2015/eprogalused/uploads/Main/mesipuu.txt")
baidid = failVeebis.read()
tekst = baidid.decode() # baitidest saab sõne
ridadeKaupa = tekst.splitlines() # sõne jaotatakse reavahetuse kohtadelt
failVeebis.close()
print(ridadeKaupa[4]) # rida indeksiga 4
```

Rida indeksiga 4 (`ridadeKaupa[4]`) on siis tavamõistes 5. rida, sest loendamine algab nullist. Kuna tegemist on sõnega, siis saame selle üksiku sümboli ka indeksi abil kätte. Näiteks real

indeksiga 4 saab sümboli indeksiga 7 `ridadeKaupa[4][7]`. Proovige, mis sümbol sellel kohal on. Vastust küsitakse 7. nädala testis.

KINDEL STRUKTUUR. HTML

Eelmises näites oli veebis ilma vorminduseta tekstifail. Suur osa veebis olevatest failidest on keerulisema struktuuriga. Sageli sellisega, kus lehekülje lähtekood on tavalisele vaatajale arusaamatu ja alles nt veebilehitseja kuvab nii, et oleks arusaadav.

Näiteks Tartu Ülikooli arvutiteaduse instituudi veebileht algab nii.

```
<!DOCTYPE html>
<html lang="et" dir="ltr">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
```

Esimese reaga saame teada, et tegemist on HTML-keele tekstiga `!DOCTYPE html`. HTML (*HyperText Markup Language*) on veebilehtede, mis ongi oma olemuselt hüpertekstid, puhul väga levinud. (Vt ka <https://et.wikipedia.org/wiki/HTML>.) Teisel real näidatakse, et keel on eesti keel `lang="et"` ja tekst on vasakult paremale (`dir="ltr"` *Left-to-right*). Edasi on veel näha, et teksti kodeering on UTF-8. HTML-keeles on olulisel kohal märgendid, mida pannakse `< >` vahele ja mille abil oskab veebilehitseja teksti sobivalt kuvada. Märgendikeeli on peale HTML teisi ka. Sageli on nende akronüümide lõpus ML, aga mitte alati. Näiteks on märgendikeel ka LaTeX.

Järgmine programm loeb meie instituudi veebilehelt 121 esimest märki. Nimelt saab funktsioonile `read()` ka argumendi ette anda. Kui argumenti pole, siis loetakse terve fail.

```
from urllib.request import urlopen
failVeebis = urlopen("http://www.cs.ut.ee")
baidid = failVeebis.read(121) # 121 esimest
tekst = baidid.decode() # baitidest saab sõne
print(tekst)
failVeebis.close()
```

Kui lehekülg on teatud struktuuriga, siis saab kirjutada programmi, mis otsib sealt huvipakkuvaid andmeid. Nt järgmine programm loeb aadressilt <http://meteo.physic.ut.ee/> ühe valitud kuupäeva andmed ja leiab sealt selle päeva keskmise temperatuuri.

```
from urllib.request import urlopen

#valitud kuupäev
päev = 24
kuu = 12
aasta = 2015

#paneme lingi kokku
vastus =
urlopen("http://meteo.physic.ut.ee/et/showperiod.php?type=setday&year="+str(aasta)+"&month="+str(kuu)+"&day="+str(päev))
#loeme terve faili
baidid = vastus.read()
tekst = baidid.decode()

#otsime failis sõna 'keskmine', mis on ümbritsetud HTML-märgenditega
otsitav = "<SMALL>keskmine</SMALL><BR><B>"
algus = tekst.index(otsitav) #meetod index tagastab otsitava sõna
positsiooni alguse
temp_algus = algus + len(otsitav) #indeks, kust tekstis võib leida
temperatuuri
deg = tekst.index(" &deg;") #peale temperatuuri on tühik ja sümbol
temp = tekst[temp_algus:deg] #lõikame temperatuuri välja

#väljastame kuupäeva ja keskmise temperatuuri
print(str(päev)+"."+str(kuu)+"."+str(aasta)+": "+str(temp))
```

Uurige erinevate päevade keskmisi temperatuure. Nt kui palju erineb 2013. aasta jõulupäeva keskmine 2014. aasta jaanipäeva keskmisest.

Ülesanne

Millised järgnevatest on märgendikeeled?

- ☐ ETML
- ☐ HTML
- ☐ LaTeX
- ☐ SGML
- ☐ XML

AVAME VEEBILEHITSEJA

Pythoni programmi saab panna veebilehitsejat avama.

```
from webbrowser import *  
open("www.cs.ut.ee")
```

Samuti saab programselt avada selle lehekülje, mis näitab, mis on õigekeelsussõnastikus kirjas sõna "programmeerimine" kohta.

```
from webbrowser import *  
aadress = "http://www.eki.ee/dict/qs/index.cgi?Q=programmeerimine"  
open(aadress)
```

Võime ka otsisõna programmi kasutajalt küsida.

```
from webbrowser import *  
sona = input("Sisestage sõna: ")  
aadress = "http://www.eki.ee/dict/qs/index.cgi?Q=" + sona  
open(aadress)
```

7.2 Lugemine failist

LUGEMINE FAILIST. TÕDE JA ÕIGUS

Juba siis, kui tutvusime *for*-tsükliga, tõime välja võimaluse selle abil failist lugeda. Oleme seda võimalust ka korduvalt kasutanud.

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

Pikema teksti lugemise ja analüüsimise näitena võtame käsile Anton Hansen Tammsaare "Tõe ja õiguse" 1. köite. Tegemist on meie rahva ühe tüvitekstiga, mida siin omalaadselt käsitleme. Valdur Mikita on [Sirbis](#) kirjutanud: "Rahvas, kelle kõige kuulsam kirjandusteos on viieköiteline sookuivendamise käsiraamat, ei saa olla normaalne – ja ei peagi. Võimalik, et meil on ilmaruumis hoopis üks teine asi ajada."

Kõigepealt tuleb tekst arvutisse laadida [siit](#). (Kes tahab, võib proovida ka programselt veebist lugemist.)

Järgmine programm loendab, mitu korda on tekstis sõna "tõde" ja mitu korda "õigus".

```
f = open("anton_hansen_tammsaare_tode_ja_oigus_i.txt",
encoding='UTF-8')

tõde = 0 # loendaja
õigus = 0 # loendaja

for rida in f: # ridade kaupa
    sõnad = rida.split() # rea sõnad järjendisse
    for s in sõnad: # sõnade kaupa
        if s == "tõde":
            tõde += 1
        if s == "õigus":
            õigus += 1

print("Failis sõna 'tõde' on", tõde, "korda.")
print("Failis sõna 'õigus' on", õigus, "korda.")

f.close()
```

Ülesanne

Kas tekstis `anton_hansen_tammsaare_tode_ja_oigus_i.txt` on rohkem kasutatud sõna "tõde" või sõna "õigus"?

☐ Vali tõde

☐ Vali õigus

☐ Vali mõlemaid on võrdselt

See programm leidis ainult täpselt sõnade "tõde" ja "õigus" arvud. Kui tahame, et arvesse tuleksid ka need juhud, kus vahetult sõna järel on mingi kindel kirjavahemärk, siis saame kasutada järgmisi võrdlusi. Antud juhul siis võetakse arvesse need juhud, kus sõna järel on punkt, koma või küsimärk.

```
if s.strip(".",?) == "tõde":  
    tõde += 1  
if s.strip(".",?) == "õigus":  
    õigus += 1
```

Nüüd vaatame varianti, kus kontrollitakse, kas sõna "tõde" ("õigus") sisaldub vaadeldavas sõnes.

```
if "tõde" in s:  
    tõde += 1  
if "õigus" in s:  
    õigus += 1
```

Kui tahame suurte tähtedega variante arvesse võtta, siis saame seda teha nii.

```
if "tõde" in s.lower():  
    tõde += 1  
if "õigus" in s.lower():  
    õigus += 1
```

Selle variandi vastust küsitakse ka 6. nädala testis.

VEEL FAILIST LUGEMISEST. FUNKTSIOON `readline`

Vaatleme veel mõningaid võimalusi, kuidas failidest andmeid kätte saada. Alustame funktsiooniga *readline*, mis võtab failist järgmise rea. Esialgu aga teeme Thonny endaga või mõne tekstiredaktoriga faili *andmed2.txt*, kust andmeid võtame. Esimesel real on inimese nimi, teisel real vanus (täisarvuna) ning kolmandal real meiliaadress.

Fail peab olema *plain-text* kujul (laiendiga *.txt*), näiteks Wordi fail (laiendiga *.doc* vms) ei sobi.

```
f = open("andmed2.txt", encoding="UTF-8")

nimi = f.readline()
vanus = f.readline()
aadress = f.readline()

print("Nimi:", nimi)
print("Vanus:", vanus, "aastat")
print("Aadress:", aadress)

f.close()
```

Vaatame lähemalt, mis toimub.

```
f = open("andmed2.txt", encoding="UTF-8")
```

Käsk *open* otsib failisüsteemist üles soovitud faili ja tagastab viite sellele (antud näites salvestasime selle viite muutujasse *f*, mis on levinud nimi failide tähistamiseks). Kui tahame avada faili samast kaustast, kus asub programm, siis piisab vaid failinimest koos laiendiga: `f = open('andmed2.txt')`. Selleks, et täpitähed õigesti paistaksid, täpsustame kodeeringut `encoding="UTF-8"`.

```
nimi = f.readline()
```

Rida `nimi = f.readline()` loeb failist ühe rea, milles on meie näites isiku nimi, ning annab selle väärtuseks muutujale *nimi*. Järgmisel korral sama käsku kasutades loetakse juba järgmine rida - meie näites vanus.

```
f.close()
```

Käsk `f.close()` paneb faili kinni.

Programmi tööle pannes näeme, et väljundis tekib üleliigseid tühje ridu. Nimelt jäetakse iga rea lõppu alles ka failist pärinev reavahetuse sümbol. Tekstifailides tähistatakse reavahetust sarnaselt Pythoniga: kui Pythoni sõne sisse kirjutame sümboli `\n`, siis tähendab see Pythoni jaoks reavahetust. Iga kord kui tekstiredaktoris järgmise rea ette võtate, sisestab programm teie eest faili reavahetuse märgi. See reavahetuse märk jäetakse alles, kui nüüd failist teksti loeme.

FUNKTSIOONID `readlines` JA `read`

Lisaks funktsioonile `readline` saab failist info kätte ka funktsiooniga `readlines`.

```
f = open("andmed2.txt", encoding="UTF-8")
loetud = f.readlines()
f.close() # faili ei lähe enam vaja
print(loetud)
```

Näeme, et muutujas `loetud` on list, mille iga element on üks rida näidatud failist.

Veebist lugemisel kasutasime funktsiooni `read`. Seda saame kasutada ka failist lugemise korral. Nii saame kogu faili sisu ühe sõnena.

```
f = open("andmed2.txt", encoding="UTF-8")
loetud = f.read()
f.close() # faili ei lähe enam vaja
print(loetud)
```

Võime aga märkide arvu (antud juhul 5) ka ette anda.

```
f = open("andmed2.txt", encoding="UTF-8")
loetud = f.read(5)
f.close() # faili ei lähe enam vaja
print(loetud)
```

Etteantud märkide arv võib olla ka üks, siis võetakse märke ühekaupa. Järgmine programm on inspireeritud asjaolust, et SMSi saatmisel on õ-tähte sisaldavad sõnumite maksimaalne pikkus märgatavalt väiksem. Sageli asendatakse täht õ numbriga 6. Teeme vastava programmi

```
f = open("sms.txt", encoding="UTF-8")

while True: # lõpmatu tsükkel, kui ei katkestata
    sümbol = f.read(1) # loetakse üks sümbol
    if sümbol == "": # kui enam pole
        break # tsükkel katkestatakse
    if sümbol == "õ":
        print("6", end = "") # reavahetust ei tule
    else:
        print(sümbol, end = "") # reavahetust ei tule

f.close() # faili ei lähe enam vaja
```


Sama ülesande saab lahendada ka kahe *for*-tsükliga. Välimine tsükkel tegutseb ridade kaupa. Sisemisel võetakse vastavast reast sümboleid.

```
f = open("sms.txt", encoding="UTF-8")

for rida in f: # ridade kaupa
    for sümbol in rida: # sümboleite kaupa
        if sümbol == "õ":
            print("6", end = "") # reavahetust ei tule
        else:
            print(sümbol, end = "") # reavahetust ei tule

f.close() # faili ei lähe enam vaja
```

Toome ka kolmanda variandi, kus funktsiooni `read()` saame faili kogu sisu ühe sõnena käsitleda. Selles sõnes muudame funktsiooni `replace` abil kõik tähed õ numbriteks 6.

```
f = open("sms.txt", encoding="UTF-8")

failisisu = f.read()

asendatudõ = failisisu.replace("õ", "6")

print(asendatudõ)

f.close()
```

Ülesanne

Olgu failis arvud.txt

2
2000
4
210

Mis ilmub järgmise programmi mõjul ekraanile?

```
failist = open("arvud.txt")  
summa = 0  
for rida in failist:  
    summa = summa + rida  
print(summa)
```

Vali 0

Vali

02
2000
4
210

Vali 2216

Vali Veateade

7.3 Faili kirjutamine

TÜHJA FAILI

Eespool oleme tutvunud mitmete võimalustega, kuidas failist andmeid lugeda. Failid ise on meil varem olemas olnud või oleme need teinud tekstiredaktoriga (ka Thonnyga kui tekstiredaktoriga). Nüüd vaatame, kuidas isetehtud programmiga andmeid faili kirjutada. Selleks pakub võimalusi funktsioon *write* (ingl *kirjuta*).

Järgmine programm kirjutab kasutajalt küsitud andmed faili.

```
nimi = input("Palun sisesta oma nimi: ")
vanus = input("vanus: ")
aadress = input("aadress: ")

f = open("andmed3.txt", "w")
f.write(nimi + "\n")
f.write(vanus + "\n")
f.write(aadress + "\n")
f.close()
```

Faili kirjutamiseks tuleb funktsioonile *open* anda ka teine argument väärtusega "w" (nagu *write*). Kui sellise nimega fail juba eksisteerib, siis *open*(..., "w") teeb selle tühjaks. Erinevalt funktsioonist *print* ei tekita funktsioon *write* automaatselt reavahetusi. Selleks, et saada eri andmeid eri ridadele, lisasime reavahetuse sümboli käsitsi.

Proovime kirjutada programmi, mis küsib kasutajalt kaks failinime. Esimene fail peaks olema juba olemas ja seal peaks olema mingi tekst. Teine fail võiks olla uus. Programmi ülesanne on võtta esimese faili sisu, teisendada see suurtähtedesse ning kirjutada teise faili.

```
doonorNimi = input("Mis failist info võtta? ")
aktseptorNimi = input("Millisesse faili info pannakse? ")
doonorFail = open(doonorNimi, encoding="UTF-8")
aktseptorFail = open(aktseptorNimi, "w")
for rida in doonorFail:
    aktseptorFail.write(rida.upper())
doonorFail.close()
aktseptorFail.close()
```

Programm töötab nii, et ekraanile ei väljastata midagi. Ometi on tegemist võimsa vahendiga - me saame failides olevad andmed teatud muudatustega teise faili kanda. Tähtede suureks muutmine ei ole muidugi midagi eriti võimsat, aga põhimõtteliselt saab nii palju keerulisemaid ja ka vajalikumaid asju teha. Võib-olla mingeid andmeid üldse mitte teise faili lubada!?

Ka faili kirjutamiseks saab määrata kodeeringu, näiteks `open(aktseptorNimi, "w", encoding="UTF-8")`.

FAILI JUURDE. TEMPERATUURID VEEBIST FAILI

Eelmistes näidetes avasime selle faili, kuhu kirjutada tahtsime nii, et funktsiooni `open` teine argument oli `"w"`. Kui fail oli juba olemas, siis tehti see fail enne kirjutamist tühjaks. Kui teine argument on `"a"`, siis kirjutatakse olemasoleva faili lõppu. Kui sellist faili pole, siis see luuakse. Argument `"a"` tuleb ingliskeelsest sõnast *append* (*lisa*).

Järgmises programmis ongi faili avamisel teine argument `"a"` - kui fail on enne olemas, siis lisatakse ridu selle lõppu. Samuti saab ka siin määrata kodeeringu, näiteks `open(failiNimi, "a", encoding="UTF-8")`.

Programmi loeb aadressilt <http://meteo.physic.ut.ee/> aasta 2015 jaoks iga päeva keskmise temperatuuri ja kirjutab selle faili. Programm töötab umbes pool minutit. Kui võtta vähem kui terve aasta andmed, siis töötaks kiiremini. Programmis kasutatakse kahekordset tsüklit, mille põhjalikum käsitus jääb sellest kursusest välja.

```
from urllib.request import urlopen

fail = open("andmed.txt", "a") # fail avatakse juurde kirjutamiseks
päevad = [31,28,31,30,31,30,31,31,30,31,30,31]
aasta = 2015

for kuu in range(1,13): # välimine tsükkel kuude järjenumbrite kaupa
    for päev in range(1, päevad[kuu-1]+1): # sisemine tsükkel
        päevade kaupa
            # konkreetse päeva andmed
            vastus =
urlopen("http://meteo.physic.ut.ee/et/showperiod.php?type=setday&year="+str(aasta)+"&month="+str(kuu)+"&day="+str(päev))
            baidid = vastus.read()
            tekst = baidid.decode()

            otsitav = "<SMALL>keskmine</SMALL><BR><B>"
            algus = tekst.index(otsitav) # leitakse õige koht
            temp_algus = algus + len(otsitav) # siit algab temperatuur
            deg = tekst.index(" &deg;") # siin lõpeb
            temp = tekst[temp_algus:deg] # viilutades leitakse
temperatuur
            # funktsioon rjust paneb ühekohalisele päeva ja kuu
            järjenumbrile 0 ette, et oleks kahekohaline.
            fail.write(str(päev).rjust(2,"0")+str(kuu).rjust(2,"0")+str(aasta)+": "+str(temp)+"\n")

vastus.close()
fail.close()
```

Ülesanne

Kui faili `andmed.txt` tahetakse avada lugemiseks, siis millised järgmistest variantidest selleks sobivad on.

- ☐ `open("andmed.txt")`
- ☐ `open("andmed.txt", "a")`
- ☐ `open("andmed.txt", "r")`
- ☐ `open("andmed.txt", "w")`
- ☐ `open("andmed.txt", "r", encoding="UTF-8")`

ŠIFREERIMINE

Tegeleme nüüd natuke salakirjadega. Nimelt ei kirjuta me faili mitte sümboli enda, vaid hoopis teise sümboli. Näiteks *A* asemel kirjutame tähestikus 4 kohta tagapool oleva tähe *E*. Näiteks on *AEG* on siis hoopis *EIK*. Tegemist on nihkešifriga, mis on ajalooliselt tuntud šifreerimisviis. Kui nihe on teada, siis on teksti lihtne käsitsigi dešifreerida, rääkimata arvutiga. Isegi siis, kui nihe pole teada, saab dešifreerimisega suhteliselt hõlpsasti hakkama.

Ühe tähe teiseks nihutamisel oleks mõistlik tähed nummerdada ja siis vastavalt nihke järjekorranumbrile juurde liita või sellest (dešifreerimise korral) lahutada. Väga loomulik oleks näiteks *A* järjekorranumbriks võtta 1, *B* korral 2 jne. Seda me siiski ei tee, sest tähtedel ja tegelikult paljudel teistelgi (sealhulgas nähtamatutel) märkidel on sellised järjekorranumbrid juba olemas. Vastava järjekorranumbri saame teada funktsiooniga *ord*. Katsetage järgmist programmi.

```
print(ord("A"))
print(ord("a"))
print(ord("ä"))
print(ord("ö"))
print(ord(" "))
```

Näeme, et täpitähed on meie loomulikust tähestikust hoopis eemal.

Vastupidi jälle funktsiooniga *chr* saame teada arvule vastava märgi.

```
print(chr(65))
```

Proovige ka päris suurte arvudega. Räägime neist pikemalt selle nädala silmaringimaterjalis.

Nüüd aga siis programm, mis kirjutab teksti šifreeritult faili.

```
def kodeeri(sümbol, nihe):  
    return chr(ord(sümbol) + nihe)  
  
def šifreeri(failinimi, nihe):  
    f = open(failinimi, "w")  
    kiri = input("Sisesta lause ")  
    for sümbol in kiri:  
        f.write(kodeeri(sümbol, nihe))  
    f.close()  
  
failinimi = input("Faili nimi? ")  
nihe = int(input("Nihe? "))  
šifreeri(failinimi, nihe)
```

Katsetage šifreerimise programmi.

Nüüd on aga küsimus, kuidas salakiri dešifreerida. Toome siin mõned ideed, mis võivad selles aidata.

- Funktsioon *dekodeeri* on funktsioonile *kodeeri* üsna sarnane. Kui šifreerimisel tuli nihe liita, siis dešifreerimisel tuleb see lahutada.
- Kui šifreerimisel avati fail kirjutamiseks, siis dešifreerimisel tuleb see avada lugemiseks.
- Sümbolite ühekaupa käsitlemiseks saab kasutada sellist tsüklit

```
while True:  
    sümbol = f.read(1) # loetakse üks sümbol  
    if sümbol == "": # kui enam pole  
        break # tsükel katkestatakse  
    print(dekodeeri(????????), end = "") # reavahetust ei tule
```

Küsimärgid tuleb asendada vastavalt sellele, kuidas funktsioon *dekodeeri* tehtud on.

7.4 Lihtne kasutajaliides *EasyGuiga*

LIHTNE KASUTAJALIIDES. MOODUL EasyGui

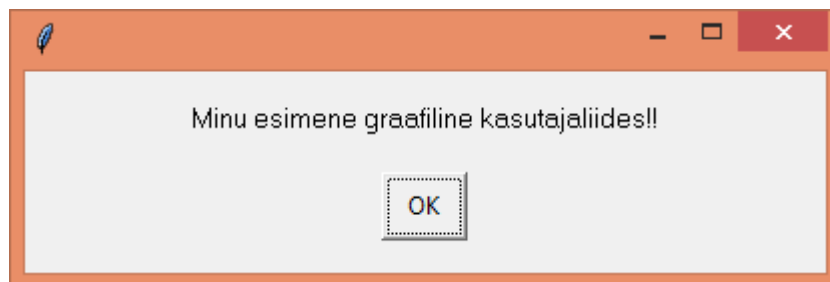
Seni oleme teinud selliseid programme, kus kasutaja on vastused sisestanud tekstina käsureaaknas. Kui vaatame igapäevaselt kasutatavaid programme, siis toimub kasutajalt info saamine nuppude, hiire liikumise ja nupuvajutuste ja mitmete muude mugavate vahenditega. Kuna nende puhul on tegemist graafilise lahendusega, siis öeldakse, et programm on **graafilise kasutajaliidesega** (ingl *graphical user interface*, GUI).

Meie alustame graafiliste kasutajaliideste tegemist mooduli [EasyGUI](#) abil. Mooduli kasutamiseks tuleb programmiga samasse kausta paigutada fail [easygui.py](#).

Esimese näitena teeme teatega dialoogiakna.

```
from easygui import * # EasyGui kasutamiseks importida
msgbox("Minu esimene graafiline kasutajaliides!!") # teateaken
```

Tulemus peaks välja nägema selline.

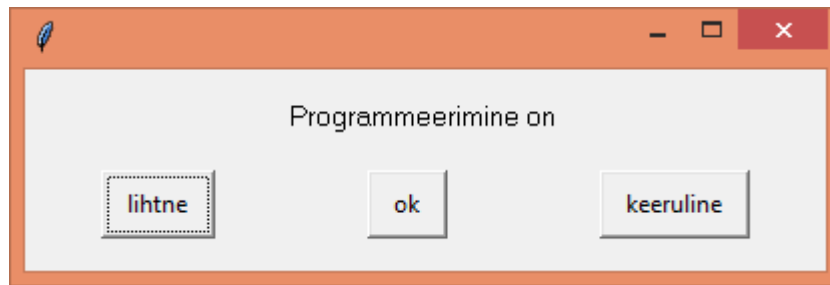


See on lihtsaim dialoogiaken. Sageli on aga vaja kasutajale rohkem võimalusi anda, kui ainult OK vajutada. Järgmises näites on juba mitu nuppu.

```
from easygui import *

nupud = ["lihtne", "ok", "keeruline"]
vajutati = buttonbox("Programmeerimine on ", choices = nupud)
msgbox("Te arvate, et programmeerimine on " + vajutati + "!")
```

Nüüd siis küsitakse nii.



Samuti näidatakse ka kasutaja valikut.



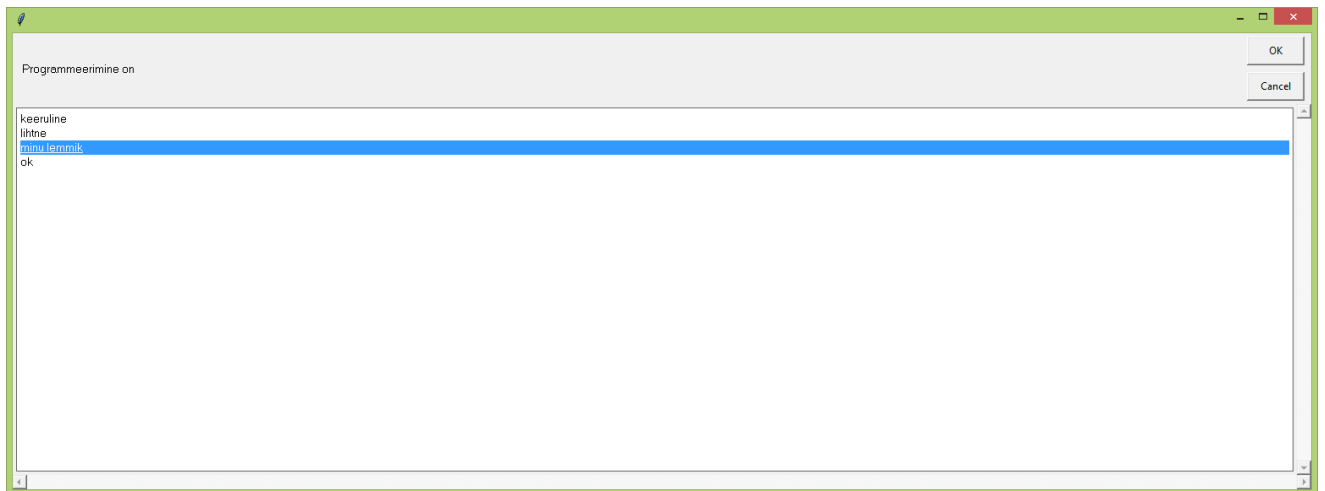
Kui kasutaja vajutab nuppu, siis `buttonbox()` tagastab valiku teksti, mis salvestatakse muutujasse `vajutati`. Kui kasutaja hoopis suleb dialoogiakna, siis tagastatakse `None`.

Kui valikuid on vähe ja nende tekstid on lühikesed, siis on nupud heaks võimaluseks. Pikemate tekstidega ja/või suurema hulga valiku korral võib olla mõistlikum valik järjendina esitamine.

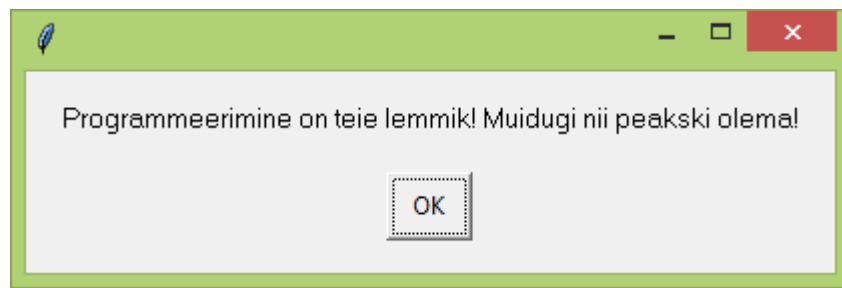
```
from easygui import *

variandid = ["minu lemmik", "lihtne", "ok", "keeruline"]
vajutati = choicebox("Programmeerimine on ", choices = variandid)
if vajutati == None:
    msgbox("Te ei valinud midagi!")
elif vajutati == "minu lemmik":
    msgbox("Programmeerimine on teie lemmik! Muidugi nii peakski olema!")
else:
    msgbox("Te arvate, et programmeerimine on " + vajutati + ", hmm, väga huvitav!")
```


Nii saame järgmise dialoogiakna.



Nüüd on järgmise akna tekst vastavalt vastusele kokku pandud.



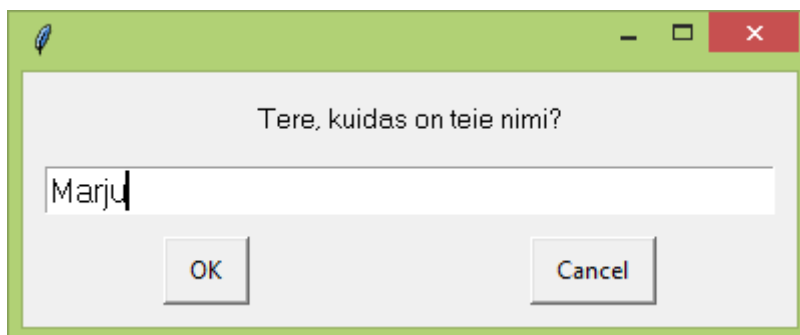
KASUTAJA SISESTAB ANDMEID

Kasutajalt saab andmeid küsida funktsiooni `enterbox` abil. Kui tahetakse täisarvulist vastust, siis võiks kasutada funktsiooni `integerbox`. Sellele saab ette anda ülemise ja alumise piiri. Kui sisestatu ei mahu piiridesse, siis küsitakse uuesti.

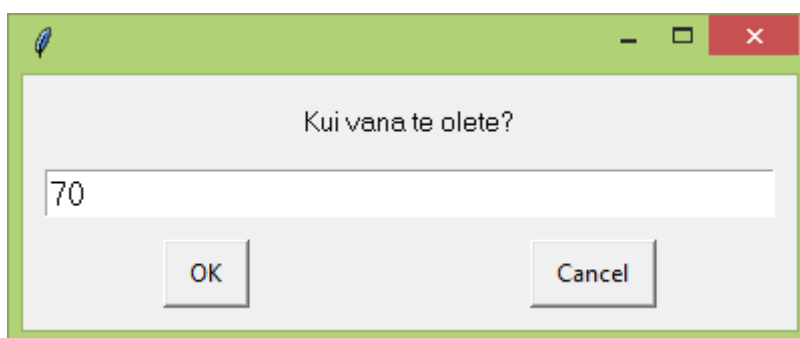
```
from easygui import *

nimi = enterbox("Tere, kuidas on teie nimi?")
vanus = integerbox("Kui vana te olete?", lowerbound = 1,
upperbound = 120)
if nimi == None or vanus == None or nimi == "":
    msgbox("Palun sisestage andmed korrektselt!")
else:
    msgbox("Tere, " + str(vanus) + "-aastane " + nimi + "!")
```

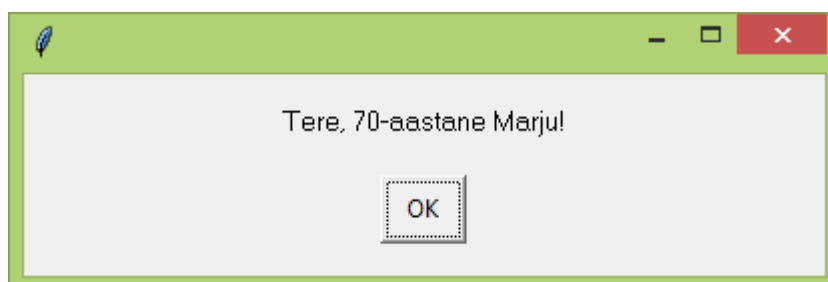
Kasutajalt küsitakse kaks küsimust. Esimese puhul kasutatakse funktsiooni `enterbox`



ja teise puhul funktsiooni `integerbox`.



Tulemus näidatakse `msgbox` abil.



EasyGui on veel võimalusi. Näiteks saab [lisada pilte](#). Julgesti võib iseseisvalt edasi uurida [EasyGui lehelt](#).

Vaata ka kokkuvõtvat [videot](http://uttv.ee/naita?id=24095): <http://uttv.ee/naita?id=24095>

EELMISTE OSADE PROGRAMMID KASUTAJALIIDSEGA

Lõpetuseks toome mõne selle nädala varasema näite graafilise kasutajaliidesega variandi.

Faili kirjutamise programm koos faili salvestamise aknaga

```
from easygui import *

nimi = enterbox("Palun sisesta oma nimi: ")
vanus = integerbox("Palun sisesta oma vanus: ", lowerbound = 1,
upperbound = 120)
aadress = enterbox("Palun sisesta oma aadress: ")

failinimi = filesavebox()

f = open(failinimi, "w")
f.write(nimi + "\n")
f.write(str(vanus) + "\n")
f.write(aadress + "\n")
f.close()
```

Failist lugemise programm koos faili valimise aknaga

```
from easygui import *

failinimi = fileopenbox()

f = open(failinimi)

nimi = f.readline().strip()
vanus = f.readline().strip()
aadress = f.readline().strip()

f.close()

tekst = "Nimi: " + nimi + "\n" + "Vanus: " + vanus + " aastat\n" +
"Aadress: " + aadress

msgbox(tekst)
```

7.5 Silmaring: Tekstikodeeringud

Erinevad tähestikud

Vahel tuleb ette, et tekstis paistab mingi täht kummalisena - võib-olla on see asendatud teise tähega, arusaamatu märgiga, küsimärgiga või hoopis mitme märgiga. Eestikeelses tekstis on sellisteks asendatud tähtedeks just õ, ä, ö, ü ja ka š, ž. Need on tähed, mis ladina põhitähestikus puuduvad. Selliseid teistele võõraid tähti leidub paljudes ladina tähestikule põhinevates tähestikes. Näiteks on [poola tähestikus](#) ą, ć, ę, ł, ń, ó, ś, ź, ż, [saksa keeles](#) aga kasutatakse tähti ä, ö, ü ja ß. Selliste tähtedega on eriti arvutitega seoses mõnevõrra rohkem muresid kui ladina tähestiku põhitähtedega. Mõnedel juhtudel ongi neist loobutud - näiteks kriipsukesed, kaarekesed ja konksukesed jäetakse ära või täht asendatakse mingite tähtede kombinatsiooniga. Tegelikult kannavad need omapärased tähed siiski lisaks keelelisele tähendusele ka teatud ajaloo ja identiteediga seotud väärtust ja nendest loobumist ei tohiks kergekäeliselt ette võtta.

Lisaks ladina tähestikul põhinevatele tähestikele on kasutusel palju teisi süsteeme, millel omakorda alamsüsteeme. Näiteks [vene](#), [ukraina](#) ja [valgevene](#) tähestikud on sarnased, aga mitte viimse täheni samad. Mõnede keelte puhul kasutatakse paralleelselt erinevaid tähestikke. Nii on näiteks [serbia keele](#) puhul kasutusel nii ladina tähestikul kui ka kirillitsal põhinevad variandid.

Arvestades, et mitmetes keeltes on kasutusel hieroglüüfid, mida võib olla väga palju (eriti võrreldes paarikümne tähega meie kandi tähestikes), on tegemist tuhandete erinevate märkidega, mida maailmas tekstide kirjanekuks kasutatakse.

7 või 8 bitti. ASCII jt

Natuke lihtsustatult võib öelda, et arvutis kujutatakse tähti (ja teisigi märke) kahendarvudena - ühtede ja nullidega. Vaja on teatud standardit selleks, et konkreetne täht paistaks ka mujal sellisena nagu algselt mõeldud. Vahepeal ju "tõlgitakse" märk kahendarvuks ja jälle tagasi.

1963. aastal avaldati [ASCII](#) (American Standard Code for Information Interchange) standardi esimene versioon. Eks standardiseerimispüüdeid oli ennegi ja lokaalselt need ka toimusid, aga ASCII mõju oli laiem ja see on ka mitmete edasiste standardite aluseks.

ASCII põhitabelis on 128 märki. Konkreetse märgi tähistamiseks kasutatakse kahendarvu, milles on seitse kahendnumbrit (e bitti). Erinevaid võimalusi on sellisel juhul $128 (= 2^7)$ ja nii ongi ASCII põhitabelis 128 märki. Põhitabelis on erinevate erimärkide ja numbrite kõrval ladina tähestiku need tähed, mida kasutatakse inglise keeles. On nii suurtähed kui väiketähed. Seejuures suurtähed A-Z (koodid kümnendsüsteemis 65 kuni 90, kahendsüsteemis 1000001 kuni 1011010) on väikestest a-z (koodid kümnendsüsteemis 97

kuni 122, kahendsüsteemis 1100001 kuni 1111010) eespool. Põhitabelis olevate märkidega saab ingliskeelseid tekste juba kenasti kirja panna.

Kui aga tahaks kasutada teisi tähti, siis juba ei saa. Loomulik oli võtta mängu kaheksas bitt, mis annab 128 märki juurde. Nii tehtigi, aga võimalikke kandidaate järgmisele 128 kohale oli rohkem, kui sinna ära mahtus. Oli ju vajadus erinevate ladina tähestikul baseeruvate tähestike spetsiaalsete tähtede jaoks, samuti kreeka tähtede jaoks, kirillitsa tähtede jaoks jm.

Siin oligi hargnemise koht selles mõttes, et kasutusele võeti erinevad kooditabelid, kus sama koodiga võisid olla erinevad märgid. Järgnevas tabelis on mõned märgid erinevatest kooditabelitest.

Kodeering	Kood 177	Kood 195	Kood 212	Kood 213
Windows-1252 (CP1252)	±	Ã	Ô	Õ
ISO/IEC 8859-5 (mitteformaalselt Latin/Cyrillic)	Б	У	Д	е
ISO/IEC 8859-7 (mitteformaalselt Latin/Greek)	±	Γ	Τ	Υ

Sellise lähenemise üks nõrku kohti on, et ühes ja samas tekstis ei saa hästi kasutada erinevate kooditabelite märke.

Mitu baiti. UTF-8. Unicode

Edasi püüti standardi poole, kus kõik vajalikud märgid on erinevate koodidega olemas. 1993. aasta alguses tutvustati ametlikult kodeeringut [UTF-8](#). UTF-8 on alates 2008. aastast veebis levinuim koodeering (praegu on hinnanguliselt umbes 87% veebilehtedest selles kodeeringus). UTF-8 on lühend fraasist **U**niversal **C**oded **C**haracter **S**et + **T**ransformation **F**ormat – **8**-bit. UTF-8 kasutab kaheksabitilisi koodiühikuid, mida konkreetse märgi jaoks võib olla kasutuses 1 kuni 4. Kaheksa bitti annab kokku ühe baidi ja baitide kaupa on mõistlik "arveldada".

UTF-8 esimesed 128 märki (ja koodidki) on samad, mis ASCII põhitabelis. UTF-8 kasutab neil juhtudel ühte kaheksabitilist koodiühikut. Kõik ASCII kodeeringus tekstid on seega kohe ka UTF-8 kodeeringus.

Järgmise 1920 märgi puhul kasutatakse kahte kaheksabitilist koodiühikut. Siin on erinevate ladina tähestikul põhinevate tähestike erilisemad tähed, samuti kreeka, kirillitsa, heebrea, araabia, kopti ja armeenia tähed. Kolme kaheksabitilist koodiühikut kasutatakse näiteks hiina, korea ja jaapani kirjamärkide puhul. Nelja koodiühikut on vaja ülejäänud Unicode'i tabelis olevate märkide jaoks, mille hulgas on näiteks matemaatilised sümbolid ja piktogrammide.

Unicode on rahvusvaheline standard arvutite tekstide kodeerimiseks. [Unicode'i tabelid](#) on olemas väga erinevate sümbolite jaoks. Lehitsege tabeleid julgesti. Kuna kahendsüsteemi arvud lähevad kiiresti pikaks, siis on tabelites kasutatud vastavaid kuueteistkümnendsüsteemi arve.

Pythonis

Ka Pythonis saab kasutada erinevaid tekstikodeeringuid. Nii saame näiteks faili avamisel tekstikodeeringu määrata: `open(failiNimi, "w", encoding="UTF-8")`.

Nüüd püüame aga Unicode'i tabelites olevaid huvitavaid märke ekraanile saada. Vastav [inglisekeelne materjal](#) on selles abiks olnud. Proovime saada ekraanile [kreeka](#) suure delta (kood 0394) ja [jaava](#) tähe, mille kood on A996.

```
print('\u0394\uA996')
```

7.6 Seitsmenda nädala kontrollülesanded 7.1, 7.2, 7.3

Seitsmendal nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks vähemalt üks järgmistest ülesannetest, kas 7.4a, 7.4b või 7.4c (võib ka kaks või kolm lahendada). Lahendused tuleb esitada Moodle'is, kus osa neist kontrollitakse automaatselt, aga osa on inimliku kontrolliga.

Moodle'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

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

Kontrollülesanne 7.1. Telegramm

Aastakümneid oli [telegramm](#) infovahetamisel väga olulisel kohal. Telegrammiga teatati saabumistest, õnnitleti jpm. Praeguseks on telegrammid paljudes maades (ka [Eestis](#)) ajalukku jäänud ja teisteski kasutatakse neid järjest vähem. Noorem generatsioon pole telegrammidega tõenäoliselt üldse kokku puutunud ja ka vanemad ei mõtle telegrammidele eriti sageli. [Muuseumides](#) ja [ajalooblogides](#) võib ühtteist siiski leida. Ilmselt on telegramme ka kodustes arhiivides.

Telegrammis kasutati ainult suurtähti. Täpitähti kasutada ei saanud ja nii oli Ä asemel kasutusel AE, Õ ja Ö asemel OE ja Ü asemel UE.

Palju õnne sünnipäevaks

asemel oli kirjas

PALJU OENNE SUENNIPAEVAKS

Olgu (UTF-8 kodeeringus) failis sõnum, mis on kirjutatud tavalisel moel.

Kirjutada programm, mis

- küsib kasutajalt failinime,
- loeb vastavast failist sõnumi ja
- väljastab selle ekraanile telegrammi stiilis. Tuleb asendused
 - Ä, ä → AE
 - Õ, õ, Ö, ö → OE
 - Ü, ü → UE
 - Kõik tähed tuleb muuta suurtähtedeks.
 - Teisi märke ei muudeta.

Näide programmi tööst:

Näiteks, kui faili *telegramm.txt* sisuks on:

Palju õnne sünnipäevaks, kallis sünnipäevalaps!

siis programm peab andma tulemuse :

```
>>> %Run yl7.1.py
Sisestage failinimi: telegramm.txt
PALJU OENNE SUENNIPAEVAKS, KALLIS SUENNIPAEVALAPS!
>>> |
```

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

Kontrollülesanne 7.2. Päevik

Inimesed on ikka päevikut pidanud - mõned salaja, mõned avalikult. Ülesandeks on päevikupidamise programm teha.

Kirjutada programm, mis

- küsib kasutaja käest ühe sissekande (võib eeldada, et sissekanne on ilma reavahetusteta);
- kirjutab (UTF-8 kodeeringus) faili ***paevik.txt lõppu*** kolm rida:
 - esimesel real praegune kuupäev ja kellaeg sellisel kujul, nagu seda tagastab funktsioon `datetime.today()` (vt näidet);
 - teisel real kasutaja sisestatud kirje;
 - tühi rida (pole kohustuslik).

Kui faili *paevik.txt* ei eksisteeri, siis tuleb see luua. Kui aga fail juba eksisteerib, siis ei tohi selle faili olemasolevast sisust midagi üle kirjutada. Failinimi peab automaatkontrolli läbimiseks kindlasti olema ***paevik.txt*** (mitte *päevik.txt*) ja fail peab olema kodeeringus UTF-8 (encoding="UTF-8").

Praeguse kuupäeva ja kellaaja saamisel aitab järgmine programmilõik.

```
from datetime import datetime
kuupäev_kellaeg = datetime.today()
print("Kuupäev ja kellaeg: " + str(kuupäev_kellaeg))
```


Näide programmi tööst:

Faili *paevik.txt* sisu enne programmi käivitamist:

```
2016-05-06 00:00:42.010288  
Lahendan kontrollülesannet 7.2
```

Programmi töö:

```
>>> %Run yl7.2.py  
Sisesta sissekande tekst: Sain ülesande lahendatud.  
>>> |
```

Faili *paevik.txt* sisu pärast programmi käivitamist:

```
2016-05-06 00:00:42.010288  
Lahendan kontrollülesannet 7.2  
  
2016-05-06 00:27:35.460279  
Sain ülesande lahendatud.
```

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

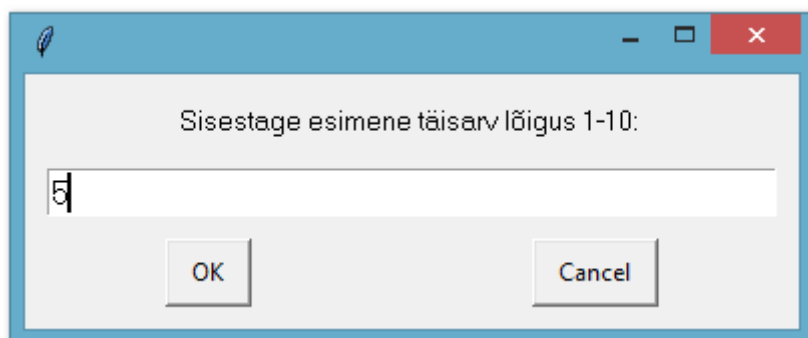
Kontrollülesanne 7.3. Kalkulaator

Koostada EasyGUI graafilise kasutajaliidesega kalkulaatori programm, mis

- laseb kasutajal
 - sisestada kaks täisarvu lõigus 1-10 (*integerbox*);
 - nuppude abil valida liitmise, lahutamise või korrutamise vahel (*buttonbox*);
- väljastab arvutuse tulemuse (*msgbox*).

Automaatkontrolliks peab faili nimi olema **yl73.py**.

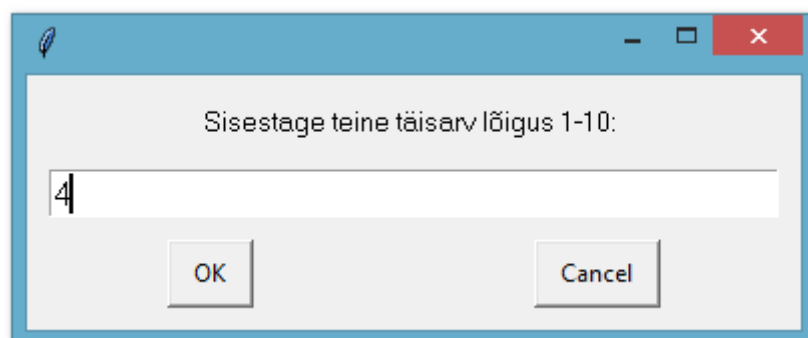
Näide programmi tööst:



Sisestage esimene täisarv lõigus 1-10:

5


OK Cancel



Sisestage teine täisarv lõigus 1-10:

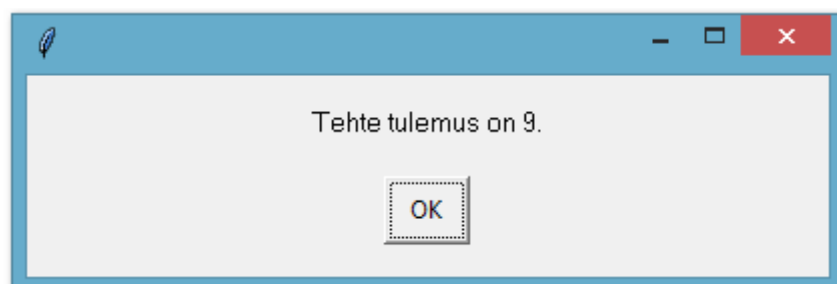
4

OK Cancel



Valige tehe:

+ - *



Tehte tulemus on 9.

OK

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

7.7 Seitsmenda nädala kontrollülesanded 7.4abc

Seitsmendal nädalal tuleb esitada nelja kohustusliku ülesande lahendused. Neljanda ülesande puhul on võimalik valida lahendamiseks vähemalt üks järgmistest ülesannetest, kas 7.4a, 7.4b või 7.4c (võib ka kaks või kolm lahendada). Lahendused tuleb esitada Moodle'is, kus osa neist kontrollitakse automaatselt, aga osa on inimliku kontrolliga.

Moodle'is on ka nädalalõputest 10 küsimusega, millest tuleb vähemalt 9 õigesti vastata.

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

Järgmisest kolmest ülesandest (7.4a, 7.4b, 7.4c) tuleb lahendada vähemalt üks.

Kontrollülesanne 7.4a Täiendatud peo eelarve

Ülesandes 6.3 pidite kirjutama programmi, mis arvutab peo maksimaalse ja minimaalse eelarve küsides inimeste arvu kasutajalt. Selles ülesandes tuleb inimeste arv saada failist.

Juubelile on kutsutud hulk inimesi, kellest osa on teatanud, et nad tulevad ja ülejäänute kohta ei ole midagi teada.

Juubelikorraldaja paneb nimekirja faili järgmisel kujul:

? Anna
+ Peeter
+ Ülle
? Eva
? Juhan
+ Maria
+ Epp
+ Anu

Faili igal real on märk + (tuleb) või ? (ei tea veel) ja inimese nimi. Tehke see fail ise mingi tekstiredaktoriga (võib ka Thonnyga). Faili kodeeringuks kasutage UTF-8.

Peo eelarve koosneb kahest osast: söök ja ruumi rent. Söögi peale arvestatakse iga osaleja kohta 10 eurot. Ruumi rent maksab sõltumata osalejate arvust 55 eurot. Planeerimiseks on vaja programmi, mis arvutab

- maksimaalse eelarve (arvestades, et kõik kutsutud inimesed tulevad kohale) ja
- minimaalse eelarve (arvestades, et kohale tulevad ainult need, kes on juba seda teatanud).

Programmi loomisel on mõistlik aluseks võtta ülesande 6.3 lahendus, sh funktsioon `eelarve`, mis võtab argumentiks külaliste arvu ning arvutab ja tagastab eelarve (10 euro iga külalise jaoks ja 55 eurot ruumi rendiks). Programm

- küsib kasutajalt failinime;
- loeb sellest failist informatsiooni külaliste kohta;
- arvutab ja väljastab ekraanile kutsutud inimeste arvu;
- arvutab ja väljastab ekraanile inimeste arvu, kes on juba tulemisest teatanud;
- arvutab ja väljastab ekraanile maksimaalse eelarve, kasutades koostatud funktsiooni `eelarve`;
- arvutab ja väljastab ekraanile minimaalse eelarve, kasutades koostatud funktsiooni `eelarve`.

Näide programmi tööst:

Näiteks, kui faili `nimekiri.txt` sisu on ülaltoodu, siis programm peab andma tulemuse :

```
>>> %Run yl7.4a.py
Sisesta failinimi: nimekiri.txt
Kutsutud on 8 inimest
5 inimest tuleb
Maksimaalne eelarve: 135 EUR
Minimaalne eelarve: 105 EUR
>>> |
```

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

Kontrollülesanne 7.4b Nimepäev

Kui aadressile https://courses.cs.ut.ee/MTAT.TK.012/2015_fall/uploads/Main/ lisada kuunimi (nt. https://courses.cs.ut.ee/MTAT.TK.012/2015_fall/uploads/Main/jaanuar), siis sellelt aadressilt võib leida lehe, kus on kirjas selle kuu nimepäevalised nii, et igal real on ühe päeva nimepäevalised (esimesel real on selle kuu esimese päeva nimepäevalised, teisel real on selle kuu teise päeva nimepäevalised jne.). "märts" asemel tuleb kasutada ilma täpitahtedeta versiooni "marts".

NB! Kui ülaltoodud aadressilt andmeid ei saa kätte (nt Macide kasutajad), siis palun proovida <http://kodu.ut.ee/~eno/mooc/jaanuar> jt.

Kirjutada programm, mis

- küsib kasutajalt kuunime (võib eeldada, et kasutaja sisestab kuunime õigesti ja "märts" asemel kirjutab "marts"),
- küsib kasutajalt päeva (võib eeldada, et sisestatud kuus leidub sellise järjekorranumbriga päev),
- loeb vastavalt aadressilt selle kuu nimepäevad (kasulik oleks nendest koostada järjend, abiks võib olla meetod `splitlines()`) ja
- väljastab ekraanile sisestatud kuupäevale vastavad nimepäevalised.

Näide programmi tööst:

```
>>> %Run yl7.4b.py
Sisestage kuu: veebruar
Sisestage päev: 15
Kuupäeval 15. veebruar on nimepäevad järgmiste nimedega inimestel:
Tiina, Neidi
>>> |

>>> %Run yl7.4b.py
Sisestage kuu: mai
Sisestage päev: 31
Kuupäeval 31. mai on nimepäevad järgmiste nimedega inimestel:
Helga, Helge, Helgi, Helja, Helje, Heljo, Helju, Elga, Olga, Olli
>>> |
```

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

Kontrollülesanne 7.4c Elutee number

Numeroloogias peetakse tähtsaks elutee numbrit, mille arvutamiseks tuleb liita kokku sünnikuupäeva ja -aasta numbrid nii, et jõutakse lõpuks ühe numbrini.

Näiteks, oletame, et sünnikuupäev on 15.05.1975. Teha tuleb niisiis järgnev tehe:
 $1+5+5+1+9+7+5 = 33$, $3+3 = 6$, seega on elutee number 6.

Aga kui sünnikuupäevaks on nt. 17.11.1981, siis arvutada tuleb järgmiselt: $1+7+1+1+1+9+8+1 = 29$, $2+9 = 11$, $1+1=2$.

Elutee numbrit arvutab järgmine (rekursiivne) funktsioon, mis võtab argumendiks sünnikuupäeva:

```
#argument s on sõne, esialgu see on kuupäev, edasi juba arvutatud arv
def elutee(s):
    #abimuutaja numbri arvutamiseks
    n = 0
    # tsükel, mis vaatab iga sümboli sõnes
    for i in s:
        if i != ".":
            n += int(i) # arvutame summat
    # kui saadud arv on väiksem kui 10, siis ongi elutee number käes
    if n < 10:
        return n
    # kui saadud arv on 10 või suurem, siis on vaja uuesti arvutada,
    # selleks kasutame jälle sama funktsiooni
    else:
        return elutee(str(n))
```

Failis *sunnikuupaevad.txt* on mingi hulk sünnikuupäevi, iga sünnikuupäev eraldi real. Kirjutada programm, mis tekitab selle faili põhjal 9 tekstifaili nimedega *eluteenumber1.txt*, *eluteenumber2.txt*, ..., *eluteenumber9.txt* ning jagab sünnikuupäevad nendesse failidesse vastavalt elutee numbrile (elutee numbri arvutamiseks kasutada funktsiooni *elutee*). Näiteks sünnikuupäev 15.05.1975 tuleb kirjutada faili *eluteenumber6.txt*.

Näide programmi tööst:

Kui faili *sunnikuupaevad.txt* sisu on

```
07.02.1969
17.11.1981
29.03.1955
```

siis faili *eluteenumber7.txt* sisu peab olema

```
07.02.1969
29.03.1955
```

ja faili *eluteenumber2.txt* sisu peab olema

```
17.11.1981
```

Kõik ülejäänud 7 faili peavad selle näite korral küll tekkima, aga jääma tühjaks.

Kui olete juba hulk aega proovinud ülesannet iseseisvalt lahendada ja see ikka ei õnnestu, siis võib-olla saate abi [murelahendajalt](#). Püütud on tüüpilisemaid probleemseid kohti selgitada ja anda vihjeid.

7.8 Lugu: Alkeemia VI

LUGU: ALKEEMIA VI

Selle loo on kirjutanud Ander Peedumäe, kes oli 2016. aastal Tartu Ülikooli informaatika
eriala 1. kursuse tudeng.

Ra ja Tea olid rännanud juba suurema osa uuest nädalast. Selja taha jäi hiiglaslik mäeahelik, rohelised väljad ja võsastunud põllud – kõik inimtühjad ja kummalised. Paari päeva eest sisenesid nad tihedasse padrikusse, mis tee peale ette jäi. Muidugi ei olnud nende tee just kõige kindlam, sest lähim asi kaardile, mille nad mägiküllast said, oli alkeemiku hütist leitud sõejoonis. Igal hommikul saatis Tea enda savikotka puude kohale tiirutama, et leida võimalikke asulaid või jõgesid, mis teekonda lihtsustaksid. Paraku saabus tiivuline tihtipeale sama targalt kui lahkus. Sel hommikul oli pilt siiski helgem: läbi linnu silmade nägi Tea jõge, mis kaugemal metsa vahel voolas. Ra teadis, et see on nende parim võimalus leida teisi hingelisi ja otsustas viivitamatult liikuda.

Paar tundi jalutamist paljastas jõekääru, mis viis metsalagendikuni. Seal paistis jõe peal olevat sild. Tea oli valmis tihikust välja ronima, et sillani jõuda, kuid Ra peatas ta äkiliselt. „Seal on keegi.“ Ra kissitas silmi ja ta nägi koguni kahte kuju. Tea jälgis pingsalt silda ning seekord märkas inimesi ka tema. Sosistatud käsu peale tõusis Tea linnust kaaslane õhku ning lendas silla kõrval konutava männi latva. Tea nägi sillal istumas üleni musta riietatud meest, kes jõi teed koos naisega, kelle pruuni kleiti ehtisid pruunid suled. Lisaks oli kõrge puu otsast näha kahte ehitist. Välja ühes otsas küürutas vana veski ning teises õõtsus kiitsaka puu latva ehitatud onn. Järsult kõlas klirin. Sillal istujate teepott purunes kildudeks ning musta riietatud mees kadus mustade sulgede pilves. Hetk hiljem ilmus ta silla ühes otsas, raamat käes. Pruunisuline naine kadus ning tekkis teise silla otsa hoides vitstest rõngaid. Lahti läks maagiline lahing.

Rändurid mõistsid, et tegemist on teist sorti maagidega, kellega alkeemikuid tihti segi aetakse. Rongataoline mees oli ilmselt võlur. Tema maagia tuli loitsudest, mida ta oli enda raamatusse kogunud, ning ta luges neid ükshaaval meisterlikult ette. Ra oli kindel, et mõni neist loitsudest oli alkeemikute kirjutatud ning arvatavasti oli rongaisand osa neist varastanud. Tõepoolest ei olnud võluritel kõige parem maine, kuid veel halvemini räägiti nõidadest, kellega oli ilmselt pruunide sulgedega naise puhul tegemist. Ta viskas nõiasõõre siia ja sinna ning andis neile veidras keeles käske. Teadagi ei valmistanud nõid enda loitse ise ette vaid tegi lepinguid olenditega teistelt tasanditelt, kes käsu peale vajalikke maagilisi vahendeid löid.

Rohi lagendikul kõrbes tulelontide all, puuladvad paindusid ning nõiasõõridest vupsasid välja taimedest, seentest ja kividest sõdurid, kes võlurile vastu astusid.

Tea kiskus Rad käisest ning sosistas tungival: „Jookseme! Enne, kui nad kogu metsa maha põletavad!“ Noormees ega ei kuulnudki teda, vaid hakkas mööda lagendiku serva rühkima, et jõuda veski juurde, mis kaugemal paistis. Tea oli sunnitud järgnema.

„Mida sa teed, Ra?“ küsis Tea ärritunult, „Me peame siit kaduma!“ Nad olid jõudnud juba üsnagi vesiveski külje alla.

„Usalda mind,“ vastas Ra ja hakkas puuehitise akendest sisse vaatama.

Riiulid olid raamatuid täis ning veelgi rohkem kirjutisi oli kuhjatud lauale ning laotud põrandale virnadesse. Ra teadis, et oli leidnud rongataolise võluri puhkepaiga.

„Siin peab olema midagi, mis aitaks meid tagasi. Ma olen selles kindel.“ Tea pööritas silmi ning küsis mürgiselt: „Kuidas me need sinu arust sealt kätte saame? Kas sa arvad, et ta jättis enda võtme mati alla?“

Ra muigas. Ta kirjutas kiiresti enda kaardi tagaküljele mingid read. Veskest kostis kolks. „Mida sa tegid?“ päris Tea segadusega.

„Ma avasin ühe tema raamatutest. Tead, kui ma öösiti tallitöö kõrvale alkeemiat õppisin, siis olid kõik raamatukogud juba kinni. Ma õppisin üht-teist lisaks ning sain vaid raamatu nime abil neile juurdepääsu.“

Tea oli sõnatu. Mitte, et Ra trikid oleksid teda üleliia üllatanud, vaid põhjusel, et nendega oli liitunud keegi kolmas. Mustas rüüs rongaisand seisis veski katusel ning vaatas valgete silmadega Teale otsa.