Tallinna Ülikool

Informaatika Instituut

Pythoni raamistike Tkinter ja wxPython võrdlus

Seminaritöö

Autor: Mart-Indrek Süld

Juhendaja: Inga Petuhhov

Tallinn 2011

Sisukord

Sissejuhatus

Tänapäeva kiire elutempo juures puutume infotehnoloogiaga igapäevaselt kokku. See on möödapääsmatu. Suund, mille me oleme valinud, sunnib meid kõiki ka igapäevaselt täiendama, et olla kursis maailmas toimuvaga. See on osalt tingitud sellepärast, et it-sektor on viimastel aastatel teinud hüppelise kasvu ja muutunud asendamatuks, mille olemasoluta enam igapäeva toimetusi ette ei kujutata. Tegu on valdkonnaga, mis areneb justkui kiirteel ja sama plahvatuslikult, nagu on uuenenud meid igapäevaselt ümbritsevad tooted, on märgatavalt tõusnud ka inimeste huvi programmeerimise vastu.

Telefonidele, pihu- ja sülearvutitele on loodud tuhandeid rakendusi erinevate kodukasutajate poolt, millest parimaid on ettevõtete virtuaalsetelt poeriiulitelt miljoneid kordi allalaetud. Ühtlasi on olnud juhtumeid, kus rakendus on oodatust populaarsemaks osutunud ja selle edasiarendamiseks on tulnud ettevõtte rajada. See tähendab seda, et igaühel meist on potentsiaalne võimalus programmeerimisega hõlpsasti raha teenida. Praegusel ajal on infotehnoloogia populaarseim kui kunagi varem ja inimeste jaoks, kes seda kõige paremini mõistavad, ka väga hästi tasustatud.

Käesolevat seminaritööd ajendas kirjutama autori huvi ja igapäevane kokkupuude programmeerimisega. Tarkvaraarendajana on ta teadlik, et mooduseid erinevate ideede realiseerimiseks on mitmeid, mille seast tuleb välja valida sobivaim, mis püstitatud eesmärkide täitmiseks kõige efektiivseim on.

Seminaritöö eesmärgiks on võrrelda ja anda ülevaade kahest sarnasest raamistikust. Töö raames otsib autor vastuseid järgnevatele küsimustele:

- Milline raamistik sobib paremini programmeerimisega vähesel määral kokkupuutunud inimesele koostamaks graafilise kasutajaliidesega rakendusi?
- Mille poolest erinevad raamistikud üksteisest?

Püstitatud eesmärkide saavutamiseks tutvub autor erinevate artiklite ja näidetega. Seejärel loob ta mõlemat raamistikku kasutades sarnase loogikaga rakendusi, andmaks ülevaate teekide kasutamise raskusest. Seminaritöö on jaotatud viieks peatükiks. Esimene peatükk annab ülevaate programmeerimiskeelest Python ja temaga kaasaskäivatest raamistikest. Teises peatükis kirjeldatakse raamistikke Tkinter ja wxPython. Kolmandas osas antakse ülevaade näidisrakendusi kasutades teegist Tkinter. Neljandas osas koostatakse ülevaade näidisrakendusi kastades teegist wxPtyhon. Viimases peatükis antakse ülevaade raamistike sarnasustest ja erinevustest.

Töös on näidetena kasutatud autori enda programmeerimisoskusi ja internetis vabalt levivaid õpetusi, mille autor on valinud enda äranägemise järgi. Arusaadavuse huvides on töö lisana CD-plaadil saadaval kommenteeritud näidisrakendused.

1. Programmeerimiskeel Python

Python on võimekas dünaamiline programmeerimiskeel, mida kasutatakse laialdaselt tarkvara arendusel kuna pakub mitmeid erinevaid võimalusi rakenduse arendamiseks. Python on alustamiseks üks sobivamaid programmeerimiskeeli inimesele, kes pole programmeerimisega varasemalt kokkupuutunud, aga soovib siseneda programmeerimis-maailma. Keel on omal initsiatiivil õpitav, lausungid on kerged ja üheselt mõistetavad ning keel on laialt kasutusel, mis tähendab, et õppematerjale leidub nii algajale kui ka edasijõudnud programmeerijale. Programmeerimiskeelt võrreldakse tihti Tcl, Perl, Ruby, Scheme või Javaga. Pythoni erilisemad omadused võrreldes teiste programmeerimiskeeltega on:

- Puhas ja loetav süntaks
- Intuitiivne objekt-orienteeritus
- Protseduurilise koodi loomulik väljendus
- Hierarhiliste pakettide tugi, täielik modulaarsus
- Erindite-põhine vigade käsitlus
- Dünaamiliste andmetüüpide kõrge tase
- Ulatuslikud standard-teegid ja kolmanda osapoole moodulid praktiliselt igale ülesandele
- Laiendused ja erinevad moodulid C, C++'is kirjutatule
- Liideste ja rakenduste integreerimine

Pythoni kasutajad kasutavad tihti väljendit „patareid kaasaarvatud" standard teekide kirjeldamiseks, kuna käsitletakse kõike alates asünkroonsest töötlusest kuni .zip failide töötluseni. Keel ise on paindlik ja käsitleb peaaegu kõiki valdkondi. Python võimaldab kirjutada koodi kiirelt, nimelt oma veebiserveri on võimalik ehitada kõigest kolme koodireaga. Optimeeritud kompileerija tagab protsesside täitmise rakenduste poolt nõutavast veel kiiremini. Python on saadaval kõikidele peamistele operatsioonisüsteemidele: Windows, Linux/Unix, OS/2 ja Mac. On olemas versioonid, mis jooksevad isegi .NET või Java virtuaalmasina peal. Ühtlasi on hea teada, et sama kood jookseb muutmata kujul igal erineval platvormil, kus teda rakendada (About Python, 2011).

1.1. Raamistiku tutvustus

Raamistik on eelnevalt programmeeritud korduvkasutatavate objektiklasside hulk, mis annab kasutajale või programmile omavahel seotud funktsioonide kogum. Raamistiku kasutamisega hoitakse ressursse kokku töötava süsteemi loomise pealt ja tarkvaraarendajad saavad aega pühendada tarkvara loomiseks, ehk rakendus ehitatakse juba välja arendatud töötava süsteemi peale (Clifton, 2003).

Tarkvara raamistikke on kahte liiki:

1. Rakenduste loomise raamistik - abstraktsioon, mille tarkvara üldist funktsionaalsust tagavat koodi saab valikuliselt kasutaja muuta. Eelnevalt defineeritud programmeerimisliideste kollektsioon.
2. Veebi rakenduste raamistik - tarkvara raamistik, mis on disainitud pakkumaks tuge dünaamiliste veebilehtede, veebi rakenduste või veebi teenuste loomisel. Definitsioonid laialt levinud tegevuste programmeerimisliidestest, mis kaasnevad veebiarendusega.

1.2.Pythoni graafilise kasutajaliidese raamistikud

Pythonile on saadaval mitmeid erinevaid graafilise kasutajaliidese raamistikke või tööriistakomplekte (Tabel 1), alustades Tkinteriga, mis on üks populaarsemaid kasutusel olevatest raamistikest ja on ühtlasi saadaval koheselt Pythoni arvutisse paigaldamisel, lõpetades suure hulga erinevate spetsiifiliste platvormide lahendustega (About Python, 2011).

Tabel 1 - kasutuselolevad raamistikud (GUI Programming in Python, 2011):

Pakett	*Platvorm*	*Kirjeldus*
EasyGUI	*Tk*	Moodul, mis võimaldab kiirelt programmeerida algset graafilist kasutajaliidest
Jynthon	*Java*	Implementatsioon integreerimiseks Pythoni kõrgetasemelise, dünaamilise ja objekt-orienteeritud Java platvormiga
Kivy	*Windows, MacOSX, Linux, Android*	Avatud lähtekoodiga raamistik loomaks uusi kasutajaliideseid
Lucid	*Windows, GTK, MacOS*	Paigutusmootor, mis kasutab operatsioonisüsteemide graafilise kasutajaliidese tööriistu
Ocean	*mitmed erinevad*	Graafilise kasutajaliidese teekide kogu mängude arendamiseks
Tkinter	*Tk*	Pythoni standartne graafilise kasutajaliidese teek. Üks populaarsematest ja kõige laiem kasutuselolevatest raamistikest
WxPython	*WxWidgets*	Laialt levinud C++'is loodud raamistik graafiliste rakenduste kirjutamiseks

2. Raamistikud Tkinter, wxPython ja nende kasutamine

Otsustasin kasutada Tkinteri ja wxPythoni raamistikke, kuna valiku tegemisel sai määravaks õppematerjalide rohkus ja tegu on konkureerivate raamistikega. Ühtlasi on suur tõenäosus, et graafilise kasutajaliidesega rakendusi luues puututakse ühel hetkel kokku just nendega.

Tkinteri kasutamiseks pole vaja kasutajapoolset lisategevust, kuna raamistik on vaikimisi saadaval Pythoni arvutisse paigaldamisel. Sellest tulenevalt tehakse tõenäoliselt esimesed graafilise kasutajaliidesega rakendused just antud raamistikku kasutades (Python interface to Tcl/Tk, 2011).

Tkinteriga sama kasutusvaldkonnaga raamistik on wxPython (UsefulModules, 2011), mille kasutamine on kiirelt õpitav, kuna süntaks ei ole keeruline ja raamistiku kasutamise kohta leidub hulganisti õppematerjale (What is wxPython, 2011).

Raamistikud peaksid järelikult küllaltki sarnased olema – nii koodi loetavuse kui ka õppematerjalide rohkuse poolest. Eesmärk on anda kasutajale ülevaade näidete põhjal kahest sarnasest raamistikust.

Mõlemat raamistikku kasutades luuakse Pythoni versioonil 2.7.2 samasisulised rakendused. Tegu on lihtsate programmidega, mille eesmärgiks on demonstreerida raamistike kasutamise erinevust ja raskust loomaks sarnase käitumise ja ülesehitusega rakendusi.

Tkinteri ja wxPythoni erinevuse toob kõige paremini välja rakendus, mis ei ole oma ülesehituselt keeruline, kuid samal ajal tutvustab raamistike võimalusi. Näiteks rakendus, mille ülesanne on liita kokku kasutaja poolt sisestatud arvud ja saadud tulemus kasutajale tagastada. Näites proovitakse kasutada lihtsamaid kasutajaliidese komponente, mida rakendusi luues tavaliselt vaja läheb:

- Raam, mille külge kinnitatakse graafilised komponendid
- Tekstisildid, mis kuvavad kasutajale teksti
- Teksti sisestuskastid, kuhu kasutaja sisestab teksti
- Nupp, millele vajutades proovitakse liita teksti sisestuskastides olevad tekstilised väärtused

Näidisrakenduse kirjeldus:

1. Kasutajale kuvatakse raam *windowsi* komponentidega
2. Raamis sisaldub 3 tekstisilti, 2 teksti sisestuskasti ja nupp
3. Kasutaja peab arvude liitmiseks mõlemad sisestuskastid täitma ja nupule vajutama
4. Nupule vajutades kontrollitakse teksti välju:

a. Kui üks sisestuskastidest on tühi, teavitatakse kasutajat sellest ja lastakse teksti sisestuskastid uuesti täita
b. Muidu, kui mõlemad sisestuskastid on täidetud, kuvatakse kasutajale teksti sisestuskastidest saadud andmed
    i. Kasutajale kuvatakse teksti sisestuskastidest saadud arvude summa
    ii. Kui liitmisel ilmnes tõrge, palutakse kasutajal veenduda andmete õigsuses

Näidisrakenduse sisendid ja väljundid:

- Sisendid:
    - Kaks tekstivälja, mida rakendus kasutab kasutaja poolt sisestatud teksti lugemiseks
    - Nupp, millele vajutades rakendus loeb tekstiväljadesse kirjutatud parameetrid
- Väljundid:
    - Hüpikaken, mida kasutatakse kasutajale tagasiside andmiseks
    - Tekstisilt, mida rakendus kasutab arvude summa esitlemiseks

2.1. Tkinteri ja WxPythoni kasutamine

Raamistikega rakendusi luues peavad programmeerijal olema algteadmised objekt-orienteeritud programmeerimisest. Vajalik on arusaamine objektide loomisest, tema atribuutidest ja meetoditest, sest graafilise kasutajaliidesega rakendust luues on silmale nähtavad komponendid vaadeldavad eraldi objektidena. Igal komponendil on oma tunnused (atribuudid) ja tegevused (meetodid), mis eristavad teda teistest objektidest (Jentson, 1999). Näiteks tekstisilt ja teksti sisestuskast võivad visuaalselt sarnased tunduda, aga süsteemi poolt vaadatuna on tegu erinevate komponentidega. Tekstisilt on mõeldud teksti kuvamiseks kasutajale ja omab seetõttu atribuuti nimega `text` (Atribuudi väärtus ongi kasutajale kuvatav tekst). Teksti sisestuskast on mõeldud programmi ja kasutajavaheliseks suhtluseks, mille tõttu omab komponent meetodit nimega `getText()`, mis tagastab meetodi väljakutsujale tekstikastis oleva teksti.

3. Graafilise kasutajaliidese raamistik Tkinter

Tkinter on Pythoni paigaldamisel arvutisse vaikimisi kaasatulev graafilise liidese pakett, mis on adapter Tk-liidese ja tarkvaraarendaja vahel, mis tagab kõikidele graafilistele tööriistadele ligipääsu. Tk on erinevate programmeerimiskeelte standartne graafilise kasutajaliidese utiliitide komplekt, mida kasutatakse rakenduste loomiseks Windowsi, Mac OS X, Linuxi ja teiste platvormide peal(Welcome to the Tcl Developer Xchange, 2011). Tkinter on üks paljudest saadaolevatest liidese pakettidest, kuid graafiliste komponentide loomiseks on ta üks enim kasutatavaid raamistikke tänu oma lihtsusele (Lundh, 1999).

Tkinteri teek on Pythoni standard-liides tagamaks ligipääsu graafilise kasutajaliidese komponentidele. Tk ja Tkinter on saadaval nii Windowsi, Macintoshi kui ka enamikel Unix platvormidel ja alates versioonist 8.0 võimaldab Tk kasutada süsteemisiseselt defineeritud kasutajaliidese komponente (*native look and feel)* (Effbot, 2011).

Tk liides ise asub binaarses moodulis nimega _tkinter, mis oligi Tkinter varasemates versioonides. Moodul sisaldab Tk madalama taseme liideseid, mida programmeerijad ei tohiks tarkvara arendades otseselt kunagi kasutada, kuna tavaliselt on see süsteemi siseselt jagatud teek (Lundh, 1999).

Lisaks Tk-liidese moodulile, sisaldab Tkinter ka paljusid Pythoni mooduleid. Neist 2 kõige tähtsamat on Tkitneri moodul ise ja moodul nimega Tkconstants. Tkinteriga tekitatud rakenduste terviklikud koodid on Lisa 3: „Tkinteriga tekstisiltide tekitamine.“ ja Lisa 4: „Tkinteriga liitmisrakenduse tekitamine.“

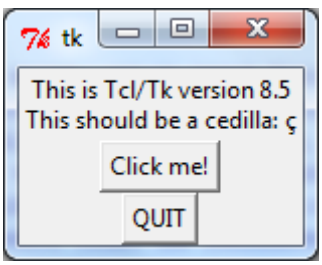
3.1.Raamistiku testimine

Tkinteri kasutamiseks on kõigepealt vajalik veenduda raamistiku olemasolus arvutis. Selleks on kiire ja lihtne meetod:

```
#Tkinter raamistiku importimine
import Tkinter
#Tkinter raamistiku testimine
Tkinter._test()
```

Koodinäide 1 – raamistiku testimine

Raamistiku olemasolul kuvatakse kasutajale Tk-infot sisaldav aken (joonis 1).



Joonis 1 – Raamistiku olemasolu kinnitus

3.2.Tekstisiltide tekitamine

Tekstisilte sisaldava rakenduse loomiseks on kõigepealt vajalik tekitada põhikonteiner. Tavaliselt on selleks raam, sest ta võimaldab enda külge objektide kinnitamise (vt. koodinäide 2).

```
#Põhiakna klass
class myFrame(Frame):
    #Klassi konstruktor
    def __init__(self, master=None):
        #Raami tekitamine
        Frame.__init__(self, master)
#Põhkiakna objekti tekitamine
frame = myFrame()
```

Koodinäide 2 – Raami loomine

Tekstisilte luues tuleb määrata kasutajale kuvatav teksti-atribuut ja konteiner, mille külge silt kinnitatakse. Koodinäites 3 on näha tekstisildi tekitamist.

```
firstLabel = Label(frame, text="Esimene tekstisilt")
firstLabel.pack()
```

Koodinäide 3 – Tekstisildi loomine

Sarnaselt paljude teiste programmeerimiskeeltega, saab ka Tkinteriga parameetreid määrata objekte luues või siis hilisemas faasis. Näiteks raamile saab määrata miinimum- ja maksimumsuuruse, tekstisildile värvuse ja paiknevuse konteineris (vt. koodinäide 4).

```
frame.master.title("Tkinteriga tekitatud raam")
frame.master.minsize(300, 100)
frame.master.maxsize(1000, 400)
secondLabel = Label(frame, text="Tekstisilt vasakul", fg="blue")
secondLabel.pack(side=LEFT)
thirdLabel = Label(frame, text="Tekstisilt paremal", fg="red")
thirdLabel.pack(side=RIGHT)
```

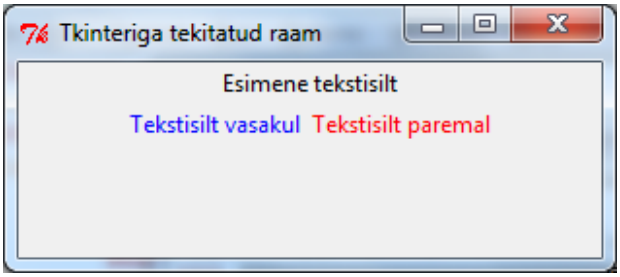
Koodinäide 4 – Objektide loomine ja parameetrite määramine

Kui objektid on loodud ja parameetrid määratud, tuleb raam käivitada (vt. koodinäide 5).

```
frame.mainloop()
```

Koodinäide 5 – Põhiobjekti käivitamine

Raami käivitamisel kuvatakse kasutajale loodud objektid (vt. joonis 2).



Joonis 2 – Raam koos siltidega

3.3. Liitmisrakenduse loomine

Tkinter võimaldab programmeerijale mitmekülgset lähenemist rakenduste loomisel: raam-objekti saab luua klassina defineerides või kasutades Tk-paketi aluskonteinerit, mille külge kinnitatakse raam ja temas sisalduvad komponendid (vt. koodinäide 6).

```
root = Tk()
root.title("Liitmisrakendus")
root.minsize(250,100)
textFrame = Frame(root)
```

Koodinäide 6 – Tk-paketi aluskonteineri loomine

Tkinteriga liitmisrakenduse tekitamiseks on vaja kahte tekstikasti, ühte nuppu ja ühte tekstisilti. Rohkemate tekstiväljade kasutamine ei ole kohustuslik, aga soovituslik, kuna seeläbi suurendatakse programmi visuaalset loetavust, sest graafiliste komponentide arvu kasvuga tõuseb ühtlasi ka raskus rakenduses orienteerumisel (vt. koodinäide 7).

```
firstInfoLabel = Label(textFrame)
firstInfoLabel["text"] = "Esimene arv"
firstInfoLabel.pack()
firstInputArea = Entry(textFrame)
firstInputArea.pack()
secondInfoLabel = Label(textFrame, text="Teine arv")
secondInfoLabel.pack()
secondInputArea = Entry(textFrame)
secondInputArea.pack()
infoLabel = Label(textFrame, text="Summa:")
infoLabel.pack()
button = Button(textFrame, text="Liida")
```

Koodinäide 7 – Komponentide loomine

Sarnaselt teistele programmeerimiskeeltele, nupule vajutus iseenesest ei tee midagi. Ta on päästik, mis võib anda süsteemile käsu sooritada mistahes arv tegevusi. Koodinäites 8 seotakse meetod nupule vajutusega.

```
#Nupule vajutus käivitab määratud meetodi
button["command"]=meetodiNimi
```

Koodinäide 8 – Meetodiga nupuvajutusele reageerimine

Peale komponentide loomist ja konteineri külge omistamist, on Tkinteri puhul vajalik määrata nende paiknemine. Kõige kergem viis seda teha, on kasutada `pack()`-meetodit, mis paigutab komponendid konteinerisse blokkide põhiselt (vt. koodinäide 9). Võimalik on valida veel `grid()`-meetodi, mis paigutab komponendid konteinerisse tabeli põhiselt ja `place()`-meetodi vahel, mis paigutab komponendid konteineris spetsiifilisele asukohale. `Place()`-meetod on kõige raskemini hoomatav paigutamise meetod, mille kasutamist peaks võimaluse korral vältima. Täpsemalt saab vormindamisest lugeda aadressil http://effbot.org/tkinterbook/grid.htm

```
textFrame.pack()
```

Koodinäide 9 – Komponentide paigutamine konteinerisse

Kui komponendid on loodud ja parameetrid määratud, tuleb määrata nupuvajutuse loogika. Nupule klikkides peab programm kontrollima tekstiväljadesse sisestatud teksti ja andma sellest tulenevalt tagasisidet. Kasutajale tagasiside andmiseks on Tkinteris võimalik kasutada hüpikaknaid. Koodinäites 10 teavitatakse kasutajat sellest, et teksti sisestuskastis puudub tekst.

```
def meetodiNimi():
    if firstInputArea.get().strip() == "":
        tkMessageBox.showerror("Viga!", "Esimeses kastis puudub tekst")
```

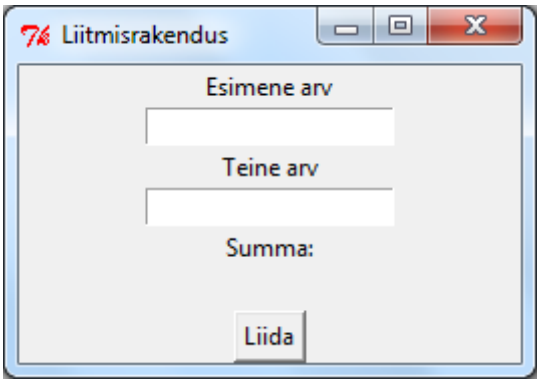
Koodinäide 10 – Tagasiside andmine tekstikasti põhjal

Kasutajat tuleb teavitada ka vea esinemisel ülesannete täitmisel (vt. koodinäide 11).

```
try:
    first = int(firstInputArea.get().strip())
    second = int(secondInputArea.get().strip())
except ValueError:
    tkMessageBox.showwarning(„Andmete teisendamine ebaõnnestus", "Veendu
andmete õigsuses")
```

Koodinäide 11 – Tagasiside andmine vea tekkimisel

Rakenduse käivitamisel kuvatakse kasutajale loodud komponendid (vt. joonis 3).



Joonis 3 – Liitmisrakendus

Rakenduse tööpõhimõttest ülevaate saamiseks vaata „Lisa 1: Tkinteri liitmisrakenduse loogika„.

4. Graafilise kasutajaliidese raamistik wxPython

wxPython on Ryan Dunni poolt loodud Pythoni raamistik, mis on mõeldud graafilise kasutajaliidesega rakenduste valmistamiseks. Raamistik võimaldab programmeerijal luua keerulise funktsionaalsusega programme erinevatel platvormidel. Toetatud on 32- ja 64-bitine Windows, Mac ja mitmed Unixil põhinevad süsteemid. wxPython koondab endasse erinevad wx-tööriistad, mis on kirjutatud programmeerimiskeeles C++ (What is wxPython, 2011).

wxPython koosneb viiest põhimoodulist:

1. Windows – sisaldab erinevaid aknaid, mis moodustavad rakenduse:
    - paneel (*panel*),
    - dialoog (*dialog*),
    - raam (*frame*),
    - keritav aken (*scrolled window*).
2. GDI (*Graphics device interface*) – erinevate klasside kogu, mida kasutatakse graafiliste väljundite genereerimiseks. Moodul sisaldab klasse manipuleerimiseks fontide (*fonts*), värvide (*colors*), pintslite (*burshes*), pliiatsite (*pens*) või piltidega (*images*).
3. Core - moodul sisaldab elementaarseid klasse, mida kasutatakse arendamisel nagu:
    - objekti-klass (*object class*), mis on kõikide klasside ema,
    - suuruste muutjad (*Sizers*),
    - sündmused (*Events*),
    - erinevad geomeetria klassid nagu punkt (*Point*) ja ristkülik (*Rectangle*).
4. Controls – moodul sisaldab erinevaid tööriistu, mida kasutatakse graafilise liidesega rakendustes nagu nupud ja tööriistariba.
5. Misc – moodul sisaldab klasse, mida kasutatakse logimiseks, rakenduse seadistamiseks, süsteemi seadete muutmiseks ja näiteks töötamiseks ekraaniga.

Sarnaselt Pythonile, on wxPython avatud lähtekoodiga, mis tähendab, et kõik võivad sisse viia parandusi või lisada erinevaid lisasid ja see on kõigile tasuta kasutamiseks mõeldud (Bodnar, 2011). Raamistikuga tekitatud rakenduste terviklik kood on lisades vastavalt Lisa 5 – „wxPythoniga tekstisiltide tekitamine.“ ja Lisa 6 – „wxPythoniga liitmisrakenduse tekitamine.“

4.1. Raamistiku testimine

wxPython on saadaval lisamoodulina, mis tähendab seda, et ei piisa ainult Pythoni paigaldamisest arvutisse, vaid on vajalik kasutajapoolne lisategevus. Lisamoodul on allalaadimiseks saadaval aadressil http://www.wxpython.org/download.php#stable.

Kasutamiseks tuleb allalaetud moodul arvutisse paigaldada ja sellega piirdub kasutajapoolne tegevus.

Mooduli olemasolus veendumiseks on lihtne moodus:

```
try:
    import wx
except ImportError:
    raise ImportError,"wxPython ei ole arvutisse paigaldatud!"
```

Kui programmi käivitamisel vigu ei esinenud, siis tähendab, et moodul on arvutisse edukalt paigaldatud ja saadaval kasutamiseks.

4.2.wxPythoniga tekstisiltide tekitamine

WxPythoniga rakenduste loomisel tuleb kõigepealt luua põhikonteiner, mille külge kinnitatakse alamkomponendid. Tavaliselt on selleks raam, kuna tegu on kõrgeima taseme konteineriga (vt. koodinäide 12).

```
app = wx.App(False)
frame = wx.Frame(None, wx.ID_ANY).
```

Koodinäide 12 – Rakenduse ja raami loomine

Tekstisilte luues tuleb määrata kasutajale kuvatav teksti-atribuut ja konteiner, mille külge nad kinnitatakse (vt. koodinäide 13).

```
firstLabel = wx.StaticText(frame, label="Esimene tekstisilt")
```

Koodinäide 13 – Tekstisildi loomine

wxPythoniga on komponentide parameetrite määramine küllaltki kasutajasõbralik. Meetodite nimed on loogilised ja ülevaatlikud, mille tulemusena on kood loetav ja hästi tõlgendatav (vt. koodinäide 14).

```
frame.SetMinSize((300, 100))
frame.SetMaxSize((1000, 400))
firstLabel = wx.StaticText(frame, label="Esimene tekstisilt")
firstLabel.SetForegroundColour((255, 0, 0))
secondLabel = wx.StaticText(frame, label="Teine tekstisilt")
secondLabel.SetForegroundColour((0, 255, 0))
thirdLabel = wx.StaticText(frame, label="Kolmas tekstisilt")
thirdLabel.SetForegroundColour((0, 0, 255))
```

Koodinäide 14 – Objektide loomine ja parameetrite määramine

wxPython ei võimalda komponentide automaatset konteinerisse paigutamist. Koodinäites 15 on loodud eraldi objekt, mis vastutab asukoha määramise eest.

```
layoutManager = wx.FlexGridSizer(2,2,5,5)
layoutManager.Add(firstLabel)
layoutManager.Add(secondLabel)
layoutManager.Add(thirdLabel)
frame.SetSizerAndFit(layoutManager)
frame.Fit()
```

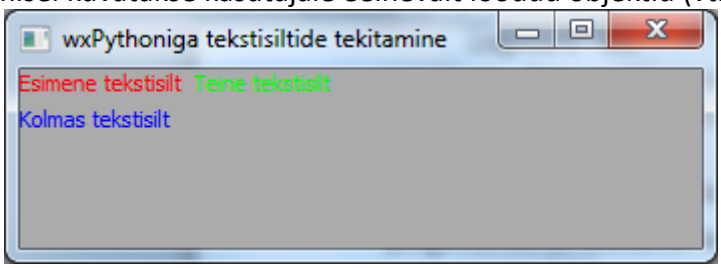
Koodinäide 15 – Komponentide konteinerisse paigutamine

Kui komponendid on konteinerisse paigutatud, jääb üle raam nähtavaks teha ja rakendus käivitada (vt. koodinäide 16).

```
frame.Show(True)
app.MainLoop()
```

Koodinäide 16 – Rakenduse käivitamine

Rakenduse käivitamisel kuvatakse kasutajale eelnevalt loodud objektid (vt. Joonis 3).



Joonis 4 –Tekitatud aken koos siltidega

4.3.Liitmisrakenduse loomine

wxPythoniga on mitmeid mooduseid rakenduste loomiseks. Objekte saab luua nii põhimeetodis (*main-method*) kui ka klassi initsialiseerides (vt. koodinäide 17).

```
#Põhimeetod
if __name__ == "__main__":
    app = wx.App()
    frame = wxFrameClass(None,-1,"Liitmisrakendus wxPythoniga")
    app.MainLoop()
class wxFrameClass(wx.Frame):
    #Klassi initsialiseerimine
    def __init__(self,parent,id,title):
        #Raami initsialiseerimine
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
```

Koodinäide 17 – Põhiobjekti loomine

Graafiliste komponentide loomiseks on üks võimalus seda teha näiteks raami tekitamisel. Sellisel juhul on tähtis meeles pidada, et objektide loomine toimuks konstruktoris või meetodis, mis kutsutakse klassi initsialiseerimisel välja. Ühtlasi peab meeles pidama, et objekte luuakse raami siseselt. Seetõttu ei kinnitata neid objektimuutuja, vaid klassi enda (*self*) külge (vt. koodinäide 18).

```
self.firstLabel = wx.StaticText(self,label=u"Esimene arv:")
self.firstInputArea = wx.TextCtrl(self,value=u"Sisesta siia arv")
self.secondLabel = wx.StaticText(self,label=u"Teine arv:")
self.secondInputArea = wx.TextCtrl(self,value=u"Sisesta siia arv")
self.thirdLabel = wx.StaticText(self,label=u"Summa:")
self.fourthLabel = wx.StaticText(self,label=u"")
button = wx.Button(self,label="Liida")
```
Koodinäide 18 – Komponentide loomine

Peale komponentide loomist on vajalik siduda nupuvajutus kindla tegevusega. wxPythoniga on vaja lisada kuulaja, mis jääb ootama teavitust nupule vajutusest. Ehk nupp teavitab kuulajana registreeritud komponenti, mis omakorda vallandab eelnevalt määratud tegevuse (vt. koodinäide 19).

```
#Nupuvajutuse kuulajaks registreerimine
self.Bind(wx.EVT_BUTTON, self.meetodiNimi, button)
```
Koodinäide 19 – Nupuvajutusele reageerimine

Komponentide paigutamine konteinerisse on wxPythonit kasutades loogiline ja arusaadav. Asukoha määramise eest vastutavaid objekte saab luua mitmeid ja neid saab paigutada vajadusel ka üksteise sisse (vt. koodinäide 20). Võimalusi on mitmeid. Täpsemalt saab vormindamisest lugeda aadressil: http://zetcode.com/wxpython/layout/

```
#Komponentide lisamine 4-rea ja 2-veeruga tabelisse.
gs = wx.GridSizer(4, 2, 5, 5)
vbox = wx.BoxSizer(wx.VERTICAL)
gs.AddMany([
    (self.firstLabel,      5, wx.EXPAND),
    (self.firstInputArea,  5, wx.EXPAND),
    (self.secondLabel,     5, wx.EXPAND),
    (self.secondInputArea, 5, wx.EXPAND),
    (self.thirdLabel,      5, wx.EXPAND),
    (self.fourthLabel,     5, wx.EXPAND),
    (button,               5, wx.EXPAND)
    ])
#Vormindaja paigutamine vormindajasse.
vbox.Add(gs, proportion=1, flag=wx.EXPAND)
#Määra vormindajas olevad elemendid raami külge
self.SetSizer(vbox)
#Tee raam nähtavaks
self.Show(True)
```
Koodinäide 20 – Komponentide paigutamine konteinerisse

Eelnevalt määrati nupuvajutuse kuulajaks raam ja sooritatavaks tegevuseks raami küljes asuv meetod. Käesoleval juhul peab meeles pidama, et raam saab kasutada kõiki tema küljes olevaid komponente, teiste objektide teenuste kasutamiseks peavad objekti parameetrite või meetodite ligipääs olema avalik. wxPythoniga rakendusi luues on võimalik kasutajale tagasiside andmiseks kasutada hüpikaknaid (vt. koodinäide 21).

```
def displayText(self,event):
        #Esimese teksti sisestuskasti sisu ei tohi tühi olla
        if self.firstInputArea.GetValue() == "":
            wx.MessageBox(„Esimeses kastis puudub tekst“, „Viga!“, wx.OK |
wx.ICON_INFORMATION)
```

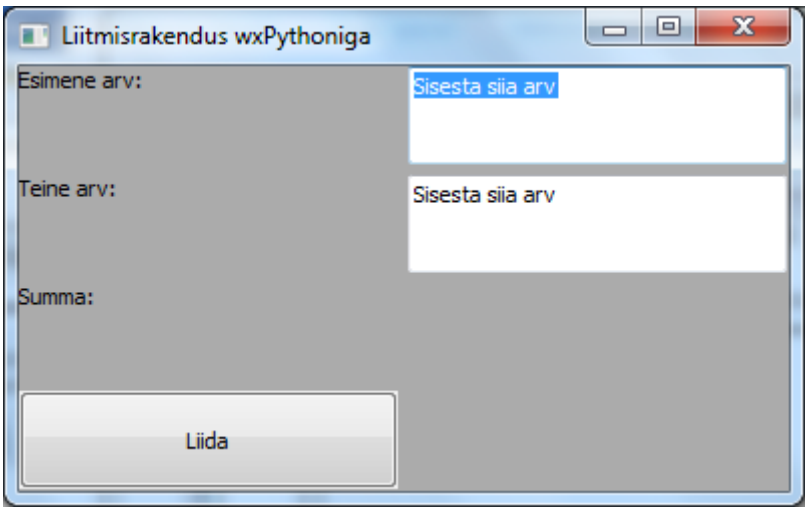
Koodinäide 21 – Tagasiside andmine tekstikasti põhjal

Kasutajat tuleb teavitada ka vea esinemisel ülesannete täitmisel (vt. koodinäide 21).

```
try:
    first = int(self.firstInputArea.GetValue())
    second = int(self.secondInputArea.GetValue())
    self.fourthLabel.SetLabel(str(first + second))
#Tegu ei olnud numbritega
except ValueError:
    wx.MessageBox(„Veendu andmete õigsuses“, „Andmete teisendamine
ebaõnnestus“, wx.ICON_ERROR | wx.ICON_INFORMATION)
```

Koodinäide 21 – Tagasiside andmine vea tekkimisel

Rakenduse käivitamisel kuvatakse kasutajale loodud komponendid (vt. joonis 4).



Joonis 5 – Liitmisrakendus

Rakenduse tööpõhimõttest ülevaate saamiseks vaata „Lisa 2: wxPythoni liitmisrakenduse loogika„.

5. Tkinteri ja wxPythoni võrdlemine

Raamistikud Tkinter ja wxPython täidavad mõlemad oma eesmärki. Selleks on graafilise kasutajaliidesega rakenduste loomine. Liitmisrakenduse tekitamiseks kulus wxPythoniga 70 rida koos kommentaaridega ja Tkinteriga vastavalt 57. Kuid wxPython tundub programmeerija jaoks parem valik olevat. Seda oma hallatavuse ja koodi loetavuse poolest. wxPythoni eelis Tkinteri ees on raamistiku suurem sarnasus populaarse programmeerimis-platvormi Javaga, milles on koostatud 18.5% kõikidest rakendustest ehk wxPythoniga rakendusi koostades on teiste keeltega sarnasusi lihtsam märgata (Bodnar, 2011).

Tkinter on lihtne senikaua, kuni seda on loodava rakenduse loogika. Keerulisema sisuga muutub Tkinteri kasutamine võrreldes wxPythoniga väga keeruliseks, kuna viimane teeb sama töö ära vähem keerukama koodiga ja on paremini hallatav (Choosing wxPython over Tkinter, 2010). Lisaks arendatakse wxPythonit aktiivselt. Tihtipeale tähendab see kahjuks ka seda, et uue versiooniga võivad kaasneda ka uued programmivead, aga loodetavasti kompenseerivad selle pidevad uuendused ja erinevad lisad (What is wxPython, 2011).

Järgnevas tabelis (Tabel 2) on välja toodud Tkinteri ja wxPythoni erinevused.

Tabel 2 – Tkinteri ja wxPythoni võrdlus

	Tkinter	wxPython
Kättesaadavus	Saadaval Pythoni paigaldamisel	Vajalik kasutajapoolne lisategevus
Paigutamise mehhanismid	Kolm paigutamise mehhanismi	Viis paigutamise mehhanismi
Graafilise kasutajaliidese komponentide kasutamine	Operatsioonisüsteemil lastakse joonistada kasutajaliidese komponendid	Implementeeritakse operatsioonisüsteemi siseselt defineeritud kasutajaliidese komponendid
Stabiilsus erinevatel platvormidel	Töökindlusega erinevatel platvormidel probleeme ei ole	Töökindlusega võib erinevatel platvormidel probleeme esineda
Arendamine	Järjepidevad uuendused puuduvad	Pidevalt uuendamisel
Arendamisega kaasnevad probleemid	Lisatööd uuendustega ei kaasne	Uuendused võivad põhjustada skriptide ümberkirjutamise vajaduse
API-liides	Erinevad versioonid omavad loetavat API-liidest	Uuendustest tulenevalt on erinevatel versioonidel tihtipeale puudulik API-liides

Vähesel määral programmeerimisega kokkupuutunud inimesele on graafilise liidesega rakenduste loomiseks parim valik wxPython. Raamistiku süntaksi tundma õppimine nõuab vähem vaeva ja loob rohkem võimalusi erinevate rakenduste loomiseks, kui seda teeb Tkinter. Lisaks on erinevate inimeste sõnul Tkinteri näol iganenud tehnoloogiaga, mis on ajale jalgu jäänud ja vajab väljavahetamist (Choosing wxPython over Tkinter, 2010).

Kokkuvõte

Käesoleva seminaritöö eesmärgiks oli uurida kahe sarnase raamistiku kasutamismugavust ja teha järeldus, kumb on algajale programmeerijale sobilikum koostamaks graafilise kasutajaliidesega rakendusi.

Töös tutvustati programmeerimiskeelt Python ja anti ülevaade, milleks ta mõeldud on. Lisaks kirjeldati erinevaid Pythoni graafilise kasutajaliidese raamistikke, koos nende kasutamiseks vajaminevate eeltingimustega. Seejärel tutvustati põhjalikumalt kahte populaarseimat raamistikku Tkinter ja wxPython, mille uurimisel käesolev seminaritöö põhineb.

Raamistikud tundusid esmapilgul sarnased, sest mõlemad on mõeldud graafilise kasutajaliidesega rakenduste tekitamiseks, kuid näidete läbitöötamise tulemusena selgus, et wxPython on kasutajasõbralikum ja programmeerijale alustamiseks kahtlemata parim valik. Raamistikke võrreldes jäi mulje, et Tkinter teeb sama töö ära lühemalt kui wxPtyhon, kuid seda ainult näiliselt. Erinevalt Tkinterist, säilitab wxPython oma lihtsuse ka keeruka funktsionaalsuse juures, mis on peamistest põhjustest, miks raamistik peaks olema algajale alustamiseks parim valik.

Seminaritöö andis autorile arvestataval määral uusi teadmisi tänapäeval laialt kasutusel olevast programmeerimiskeelest nimega Python ja kahest tema populaarseimast raamistikust.

Autori hinnangul täitis käesolev seminaritöö oma ülesande, kuna töö käigus täideti püstitatud eesmärgid.

Kasutatud kirjandus

About Python. (2011). Viimati vaadatud 15.10.2011 aadressil http://www.python.org/about/

Bodnar, J. (2011). wxPython dialogs. Viimati vaadatud 17.10.2011 aadressil http://zetcode.com/wxpython/dialogs/

Bodnar, J. (2011). Layout management in wxPython. Viimati vaadatud 17.10.2011 aadressil http://zetcode.com/wxpython/layout/

Bodnar, J. (2011). Introduction to wxPython. Viimati vaadatud 17.10.2011 aadressil http://zetcode.com/wxpython/

Clifton, M. (2003). What Is A Framework? Viimati vaadatud 13.11.2011 aadressil http://www.codeproject.com/KB/architecture/WhatIsAFramework.aspx

Effbot. (2011). An introduction to Tkinter. Viimati vaadatud 16.10.2011 aadressil http://effbot.org/tkinterbook/

GUI Programming in Python. (2011). Viimati vaadatud 15.10.2011 aadressil http://wiki.python.org/moin/GuiProgramming

Jentson, I. (1999). Struktuur- ja objektorienteeritud programmeerimise põhimõtted. Viimati vaadatud 31.10.2011 aadressilhttp://nrg.tartu.ee/algkursus/Teema12.htm

Lundh, F. (1999). An Introduction to Tkinter. Viimati vaadatud 16.10.2011 aadressil http://www.pythonware.com/library/tkinter/

Oakley, B. (2010). Choosing wxPython over Tkinter. Viimati vaadatud 02.11.2011 aadressil http://wiki.wxpython.org/Choosing wxPython over Tkinter/

Python interface to Tcl/Tk. (2011). Tkinter. Viimati vaadatud 15.10.2011 aadressil http://docs.python.org/library/tkinter.html

Sauvage, S. (2008). Building a basic GUI application step-by-step in Python with Tkinter and wxPython. Viimati vaadatud 16.10.2011 aadressil http://sebsauvage.net/python/gui/

Seshardi, P. (2009). Python Tkinter Entry Widget: A single line text input box. Viimati vaadatud 16.10.2011 aadressil http://www.prasannatech.net/2009/05/tkinter-entry-widget-single-line-text.html

UsefulModules. (2011). Viimati vaadatud 21.11.2011 aadressil http://wiki.python.org/moin/UsefulModules

Welcome to the Tcl Developer Xchange! (2011). Viimati vaadatud 12.12.2011 aadressil http://www.tcl.tk/

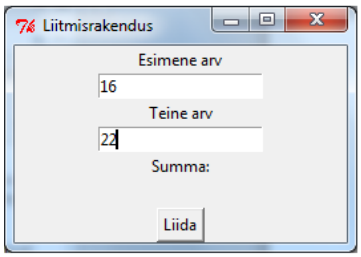
What is wxPython? Viimati vaadatud 16.10.2011 aadressil http://www.wxpython.org/what.php

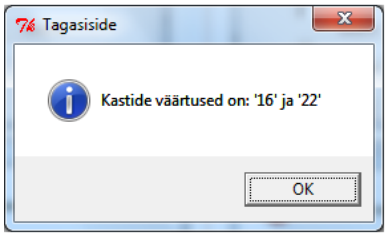
wxPython downloads. Viimati vaadatud 15.10.2011 aadressil http://www.wxpython.org/download.php#stable

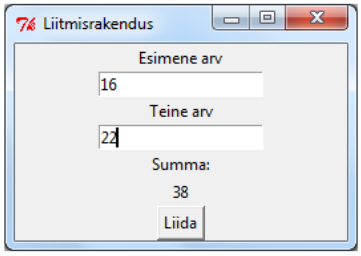
Lisad: joonised

Lisa 1: Tkinteri liitmisrakenduse loogika

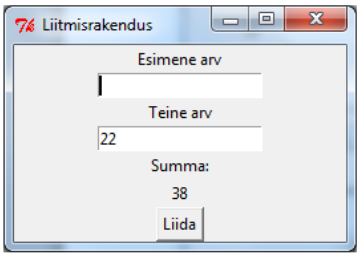
Kasutaja sisestab korrektsed parameetrid

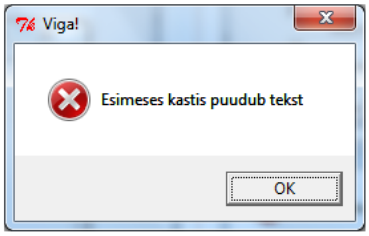




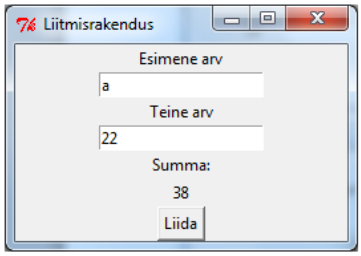


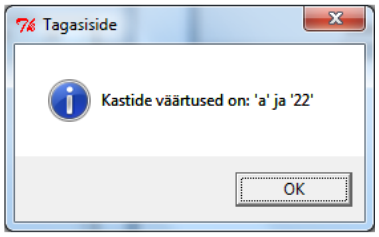
Kasutaja jätab väljad täitmata

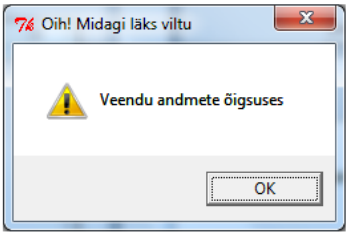




Kasutaja sisestab valed parameetrid

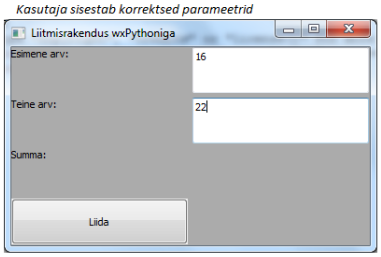


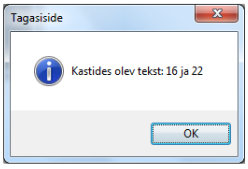


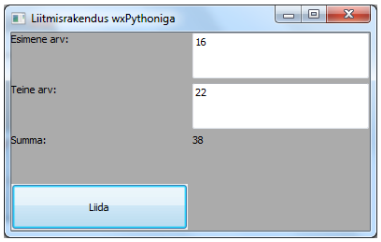


Lisa 2: wxPythoni liitmisrakenduse loogika

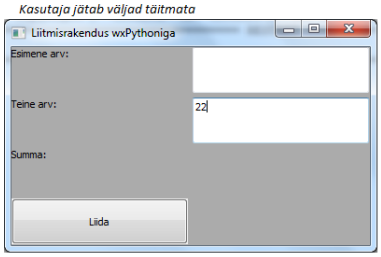
Kasutaja sisestab korrektsed parameetrid

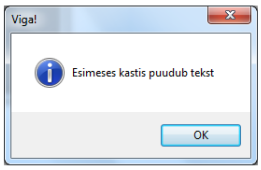




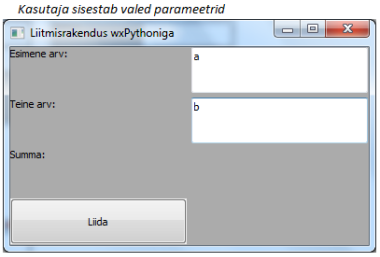


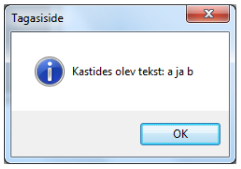
Kasutaja jätab väljad täitmata

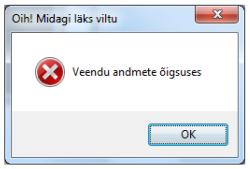




Kasutaja sisestab valed parameetrid







Lisad: koodinäited

Lisa 3: Tkinteriga tekstisiltide tekitamine

```
#Tkinteri importimine
from Tkinter import *
#Põhiakna klass
class myFrame(Frame):
    #Klassi konstruktor
    def __init__(self, master=None):
        #Raami tekitamine
        Frame.__init__(self, master)
        self.pack()
#Põhkiakne objekti tekitamine
frame = myFrame()
#Põhiakna pealkirja määramine
frame.master.title("Tkinteriga tekitatud raam")
#Põhiakna miinimumsuuruse määramine
frame.master.minsize(300, 100)
#Põhiakna maksimumsuuruse määramine
frame.master.maxsize(1000, 400)
#Esimese sildi tekitamine ja asukohaks Põhiakna määramine
firstLabel = Label(frame, text="Esimene tekstisilt")
firstLabel.pack()
#Teise sildi tekitamine ja asukohas Põhiakna määramine
secondLabel = Label(frame, text="Tekstisilt vasakul", fg="blue")
secondLabel.pack(side=LEFT)
#Kolmanda sildi tekitamine ja asukohaks Põhiakna määramine
thirdLabel = Label(frame, text="Tekstisilt paremal", fg="red")
thirdLabel.pack(side=RIGHT)
#Põhiakna käivitamine
frame.mainloop()
```

Lisa 4: Tkinteriga liitmisrakenduse tekitamine

```
from Tkinter import *
import tkMessageBox

def displayText():
    #Sisestatud teksti kuvamine kasutajale
    global firstInputArea
    global secondInputArea
    global fourthInfoLabel
    #Kontrollimine et tekstiväljad ei oleksid tühjad
    if firstInputArea.get().strip() == "":
        tkMessageBox.showerror("Viga!", "Esimeses kastis puudub tekst")
    elif secondInputArea.get().strip() == "":
        tkMessageBox.showerror("Viga!", "Teises kastis puudub tekst")
    else:
        tkMessageBox.showinfo("Tagasiside", "Kastide väärtused on: " +
      firstInputArea.get().strip() +" ja " + secondInputArea.get().strip())
        #Arvude kokkuliitmiseks peame veenduma, et tegu on numbritega
        try:
            first = int(firstInputArea.get().strip())
            second = int(secondInputArea.get().strip())
            fourthInfoLabel["text"] = first + second
        except ValueError:
            tkMessageBox.showwarning(
                    "Oih! Midagi läks viltu",
                    "Veendu andmete õigsuses")

if __name__ == "__main__":
    root = Tk()
    root.title("Liitmisrakendus")
    root.minsize(250,100)
    # Gui komponente sisaldava konteineri tekitamine
    textFrame = Frame(root)
    #Esimese teksti-sildi tekitamine
    firstInfoLabel = Label(textFrame)
    firstInfoLabel["text"] = "Esimene arv"
    firstInfoLabel.pack()
    #Esimese teksti sisestuskasti tekitamine
    firstInputArea = Entry(textFrame)
    firstInputArea.pack()
    #Teise teksti-sildi tekitamine
    secondInfoLabel = Label(textFrame, text="Teine arv")
    secondInfoLabel.pack()
    #Teise teksti sisestuskasti tekitamine
    secondInputArea = Entry(textFrame)
    secondInputArea.pack()
    #Kolmanda teksti-sildi tekitamine
    thirdInfoLabel = Label(textFrame, text="Summa:")
    thirdInfoLabel.pack()
    #Vastuse teksti-sildi tekitamine
    fourthInfoLabel = Label(textFrame, text="")
    fourthInfoLabel.pack()
    #Nupu tekitamine
    button = Button(textFrame, text="Liida", command=displayText)
    button.pack()
    #GUI komponentide pakkimine
    textFrame.pack()
    #Objekti käivitamine
    root.mainloop()
```

Lisa 5: wxPythoniga tekstisiltide tekitamine

```
#wxPythoni raamistiku importimine
try:
    import wx
except ImportError:
    raise ImportError,"wxPython ei ole arvutisse paigaldatud!"

#Loo uus raam (create new Frame())
app = wx.App(False)
#Raam(frame) on kõige pealmine windowsi komponent
frame = wx.Frame(None, wx.ID_ANY)
#Põhiakna pealkirja määramine
frame.SetTitle("wxPythoniga tekstisiltide tekitamine")
#Põhiakna miinimumsuuruse määramine
frame.SetMinSize((300, 100))
#Põhiakna maksimumsuuruse määramine
frame.SetMaxSize((1000, 400))
#Esimese sildi tekitamine
firstLabel = wx.StaticText(frame, label="Esimene tekstisilt")
#Sildi värvuse määramine punaseks
firstLabel.SetForegroundColour((255, 0, 0))
#Teise sildi tekitamine
secondLabel = wx.StaticText(frame, label="Teine tekstisilt")
#Sildi värvuse määramine roheliseks
secondLabel.SetForegroundColour((0, 255, 0))
#Kolmanda sildi tekitamine
thirdLabel = wx.StaticText(frame, label="Kolmas tekstisilt")
#Sildi värvuse määramine siniseks
thirdLabel.SetForegroundColour((0, 0, 255))
#Paigutamise eest vastutava objekti tekitamine ja talle objektide
kinnistamine
layoutManager = wx.FlexGridSizer(2,2,5,5)
layoutManager.Add(firstLabel)
layoutManager.Add(secondLabel)
layoutManager.Add(thirdLabel)
frame.SetSizerAndFit(layoutManager)
frame.Fit()
#Näita raami
frame.Show(True)
#Põhiobjekti käivitamine
app.MainLoop()
```

Lisa 6: wxPythoniga liitmisrakenduse tekitamine

```
#wxPython raamistiku importimine
try:
    import wx
except ImportError:
    raise ImportError,"wxPythoni moodul puudub"

class wxFrameClass(wx.Frame):
    #Klassi initsialiseerimine
    def __init__(self,parent,id,title):
        #Raami initsialiseerimine
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent

        #Paigutamise eest vastutava objekti tekitamine
        gs = wx.GridSizer(4, 2, 5, 5)
        vbox = wx.BoxSizer(wx.VERTICAL)

        #Esimese tekstisildi tekitamine
        self.firstLabel = wx.StaticText(self,label=u"Esimene arv:")
        #Esimese teksti sisestuskasti tekitamine
        self.firstInputArea = wx.TextCtrl(self,value=u"Sisesta siia arv")
        #Teise tekstisildi tekitamine
        self.secondLabel = wx.StaticText(self,label=u"Teine arv:")
        #Teise teksti sisestuskasti tekitamine
        self.secondInputArea = wx.TextCtrl(self,value=u"Sisesta siia arv")
        #Kolmanda tekstisildi tekitamine
        self.thirdLabel = wx.StaticText(self,label=u"Summa:")
        #Neljanda tekstisildi tekitamine
        self.fourthLabel = wx.StaticText(self,label=u"")
        #Nupu tekitamine
        button = wx.Button(self,label="Liida")
        self.Bind(wx.EVT_BUTTON, self.displayText, button)

        #Komponentide lisamine
        gs.AddMany([
            (self.firstLabel,      5, wx.EXPAND),
            (self.firstInputArea,  5, wx.EXPAND),
            (self.secondLabel,     5, wx.EXPAND),
            (self.secondInputArea, 5, wx.EXPAND),
            (self.thirdLabel,      5, wx.EXPAND),
            (self.fourthLabel,     5, wx.EXPAND),
            (button,               5, wx.EXPAND)])
        vbox.Add(gs, proportion=1, flag=wx.EXPAND)
        #Komponentide lisamine raami külge
        self.SetSizer(vbox)
        self.Show(True)

    def displayText(self,event):
        #Esimese teksti sisestuskasti sisu ei tohi tühi olla
        if self.firstInputArea.GetValue() == "":
            wx.MessageBox(„Esimeses kastis puudub tekst", „Viga!", wx.OK |
wx.ICON_INFORMATION)
        #Teise teksti sisestuskasti sisu ei tohi tühi olla
        elif self.secondInputArea.GetValue() == "":
            wx.MessageBox(„Teises kastis puudub tekst", „Viga!", wx.OK |
wx.ICON_INFORMATION)
        else:
            #Sisestuskastidesse kirjutatud info kuvamine kasutajale
```

```
            wx.MessageBox(„Kastides olev tekst:
„+self.firstInputArea.GetValue()+ „ ja „+self.secondInputArea.GetValue(),
„Tagasiside“, wx.OK | wx.ICON_INFORMATION)
            #Arvude kokkuliitmiseks peame veenduma, et tegu on numbritega
            try:
                first = int(self.firstInputArea.GetValue())
                second = int(self.secondInputArea.GetValue())
                self.fourthLabel.SetLabel(str(first + second))
            #Tegu ei olnud numbritega
            except ValueError:
                wx.MessageBox(„Veendu andmete õigsuses“, „Oih! Midagi läks
viltu“, wx.ICON_ERROR | wx.ICON_INFORMATION)

if __name__ == "__main__":
    app = wx.App()
    frame = wxFrameClass(None,-1,“Liitmisrakendus wxPythoniga“)
    app.MainLoop()
```