Tallinna Ülikool

Informaatika Instituut

PYTHONI VÕIMALUSED GRAAFILISEKS KASUTAJALIIDESEKS

Seminaritöö

Autor: Aleksandr Kuhtin
Juhendaja: Inga Petuhhov

Tallinn 2011

SISUKORD

SISSEJUHATUS

Selline teema on valitud sellepärast, et ülikoolides pannakse vähe tähelepanu graafilisele kujundamisele, kasutades erinevaid programmeerimiskeeli. Python, minu arvates, on võimekas keel, mille abiga on võimalik teha nii ilusaid nuppe, kui ka hästi kujundatud raame erinevate funktsioonidega. Python`i moodulite tundmine võiks teha teie tööd lihtsamaks ja ilusamaks.

Miks teemaks on valitud Python?
Me teame, et informaatika maailmas on päris palju programmeerimiskeeli, mis formaalselt moodulitega seotud: Ada, Pascal, Ruby, Perl, Cobol, Python jt.
Python on kõige tuntud programmeerimiskeel sellest nimekirjast. (TIOBE Software BV, 2011)

Antud töö üldiseks eesmärgiks on anda ettekujutust Python`i moodulitest.
Algtasemel proovida tutvuda nende võimalustega. Kus on võimalik Pythoni´t kasutada ja kus ta võiks kasulikuks olla ning missuguses keskkonnas on mõttekam just Pythoni´t kasutada.
Vaatame Python`i moodulid läbi ning selle erinevusi ja mugavusi kasutamises.

Töö eesmärgi saavutamiseks vajaliku informatsiooni leidmiseks kasutan ametlikke Python`i veebimaterjale ja teisi tuntud internetilehekülgi. Paremaks arusaamiseks kasutan ka lähtekoodi, et oleks nähtav loogika, mida kasutatakse Python`is. Kasutan tabeleid ja jooniseid, et arendamise tulemust näha, neid me kasutame edaspidi analüüsimiseks.

1. TEOREETILINE TAUST

1.1. MIS ON PYTHON?

Python on üldotstarbeline interpreteeritav programmeerimiskeel, mida algselt arendati skriptimiskeeleks. (vaata Joonis 1) Python võimaldab mitut programmeerimisstiili, näiteks objektorienteeritud, protseduraalset või funktsionaalset programmeerimist.
Python`it peetakse küllalt lihtsaks keeleks.

ythoni Logo

Python`i töötas 1990-ndate[1] alguses Hollandis Stichting Mathematisch Centrumis välja Guido van Rossum. Python on keele ABC järglane. 2000. aasta mais lõid Guido van Rossum ja Python`i arendustiim BeOpen PythonLabs firma, mis sama aasta oktoobris ühines firma Digital Creations tiimiga (nüüd tuntud kui Zope Corporation). 2001. aastal loodi mittetulundusühing Python Software Foundation, mis omab Python`i autoriõigust, sponsoriks on ka Zope Corporation.

Python on oma nime saanud briti naljameeste telesarja "Monty Python`i lendav tsirkus" järgi. Dokumentatsiooni koodinäidetes üritatakse vältida liigset tõsidust viidetega grupi loomingule.

Kõik Python`i avalikustatud versioonid on avatud lähtekoodiga. Enamus, kuigi mitte kõik, väljalasked ühilduvad ka GPL litsentsiga. Python`i interpretaatorit ja teeke levitatakse tasuta.
Python on dünaamiliste andmetüüpidega keel, seega programmeerijal ei ole tarvis määratleda muutujate tüüpe. See suurendab programmeerija võimalusi, kuid on veaohtlik.

Python`i koodi interpreteerimine ja optimeerimine võivad olla mõnikord aeglased protsessid. Selle koha pealt sarnaneb Python Javaga, kuna ka Python`i programmid kompileeritakse enamasti baitkoodiks, kuigi see protsess on Python`i puhul läbipaistev. Siiski on Python`i programmid masinkoodi kompileeritud programmidest (C, C++) alati aeglasemad, isegi mitu korda. Samas tänapäeva arvutite kiiruse juures pole seda vahet lihtsamate ülesannete puhul märgata.

Python`i kasutamine erineb teistest keeltest arendamise kiiruse poolest; samas on olemas kõik objektorienteeritud programmeerimise vahendid. Python on hea keel prototüüpiseerimiseks: tihtipeale luuakse mingi arvutiprogrammi esialgne kavand selles keeles ning hiljem realiseeritakse see mõnes kiiremas kõrgkeeles. (Chun, 2006)

[1] Väljalaskeaeg - 1991 a.

Python on hästi toimetatav programmeerimiskeel koos moodulitega. Moodulid võivad teha programmi lihtsamaks ja kiiremaks.

1.2.MOODULID

Moodul on funktsionaalne ja lõpetatud programmi fragment (osa).

Moodul on eraldi toodud fail oma koodiga, mille funktsioon on töötada teistes programmides. Moodul annab võimalust jagada rasked ja mahukad ülesanded väikesteks ja mugavateks ülesanneteks. (Petuhhov, 2009) Selline omadus annab arendajatele võimalust kasutada liidest mitu korda. Liidese funktsionaalsus sisaldab funktsioonide kogumit (*function*), klasse (*class*) ja konstante (*constant*).

Omavahel võivad moodulid ühenduda pakkideks (*pack*) ja lisateegiteks (*library*).

Moodulid võivad olla tavalised, selles mõttes, et nad on kirjutatud sama keele abiga, millega on kirjutatud põhiprogramm, või avardatud moodulid. Avardatud moodul on kirjutatud teise keele abiga, tihtipeale on see madala taseme keel, sest selle põhieesmärk on tõsta funktsionaalsust ja programmi kiirust.

1950-ndate aastate lõpus hakati arendama moodulite kontseptsiooni. Esimene publikatsioon oli 1976. aastal, kus oli kirjeldatud programmeerimiskeel Mesa, mis kasutas moodulit. Aastal 1977 hakkas teadlane *Niklaus Wirth* selle vastu huvi tundma. Samal aastal ta lõi uue keele – Modula-2.

Programmeerimiskeeled, mis formaalselt töötavad moodulitega: IBM S/360 Assembler, Cobol, RPG, PL/1, Ada, D, F(keeled), Fortran, Haskell, Blitz BASIC, OCaml, Pascal, ML, Modula-2, Oberon, Component Pascal, Zonnon, Erlang, Perl, Python ja Ruby. (Didkovski, 2002)
IBM System`is kasutati RPG, Cobol ja CL oma mooduleid, kui nad programmeeriti ILE keskkonnas. (Python Software Foundation, 2011)

Oma töös ma kasutan nelja moodulit: TkInter, PyGTK, PyQt, wxPython. Nendel on erinevad võimalused ja kasutamise valdkond ka erinev.

1.3. TKINTER

TkInter (***T**oolkit **inter**face* - ing. keeles) – graafiline integreeritud lisateek, mille baas on Tk (Toolkit).

Väga laialt kasutatakse GNU/Linux ja teistes Unix-platvormi süsteemides. TkInter´it on võimalik kasutada ka Microsoft Windows ja Mac OS X operatsiooni süsteemides.

Tänu sellele, et looja *Guido van Rossum* arvab, et TkInter lisateek on stabiilne ja võimekas, ainult selline moodul on olemas Python`i standardses distros. (Lundh, 1999)

1.4. PYGTK

PyGTK – on moodul Python`i seostamiseks GTK+ teekidega. (vaata Tabel 1)

PyGTK on vabatarkvara. PyGTK ei ole tervikuna tehtud Python`i programmeerimiskeeles, sest GTK+ on realiseeritud C keele abiga. Seetõttu on võimalikud situatsioonid, kui lingid, mis vajavad Python`i objekte, võivad muutuda nõrkadeks ja Python`i süsteem võib kustutada neid nõrku linke. (Finlay, 2006)

Tabel 1: PyGTK lühikirjeldus

PyGTK	
Tüüp	software development
Autor	James Henstridge
	Johan Dahlin
Arendajad	John Stowers
	Dieter Verfaillie jt
Kirjutanud	Python, GTK+
Operatsioon	UNIX
süsteemid	Windows
Viimane versioon	2.24.0 (1 april 2011)
Litsents	GNU GPL
Veebileht	pygtk.org

1.5. PYQT

Tabel 2: PyQt lühikirjeldus

PyQt	
Tüüp	Qt sidumine Pythoni jaoks
Autor	Riverbank Computing
Kirjutanud	C++, Python
Operatsioon süsteemid	mitmeplatvormiline
Litsents	GNU GPL and commercial
Veebileht	riverbankcompany.com

PyQt – Qt graafilise raamistiku sidumise kogum Python`i programmeerimiskeele jaoks. (vaata Tabel 2)

PyQt töötab igasugustel platvormidel, nii UNIX platvormidel, kui ka Mac ja Windows.

On olemas kaks varianti: PyQt4, mis toetab Qt4, ning PyQt3, mis toetab varasemaid versioone. PyQt realiseerib kõiki võimalusi, mis sisalduvad Qt raamistikus. See tähendab, et rohkem kui 600 klassi ja rohkem kui 6000 funktsiooni ja meetodi on võimalik kasutada.

- *Widget*`i kogum graafiliseks liideste kujundamiseks
- Erinevad *Widget*`i stiilid
- Andmebaasid (Oracle, SQL, ODBC)
- QScintilla – teksti redaktor
- Internatsionaliseerimine (i18n)
- XML
- SVG
- WebKit
- Toimetab videot ja audiot

PyQt sisaldab Qt Creator´it, mis võib genereerida Python`i koodi, et oleks mugavam teha disaini graafilise liidese jaoks. (Riverbank Computing Limited)

PyQt omab omi mooduleid:

- *QtCore — põhiline ja mitte graafiline klass, mis töötab regulaarsete võrranditega.*
- *QtGui — graafilise liidese komponendid.*
- *QtNetwork — võrkude programmeerimise klassid (UDP, TCP).*
- *QtOpenGL — klassid, mis annavad võimalusi kasutada OpenGL ja 3-D graafikat.*
- *QtScript — klassid, mis annavad võimalusi töötada JavaScriptiga.*
- *QtSql — klassid SQL andmebaaside jaoks.*
- *QtSvg — klassid vektor graafika SVG töötamiseks.*
- *QtXml — klassid XML jaoks.*

1.6. WXPYTHON

wxPython – on moodul mitmeplatvormilise GUI API jaoks, mis on kirjutatud põhiselt C++ keeles. (vaata Tabel 3) Nagu wxWidgets, wxPython on vabatarkvara.

wxPython`i loojad on Robin Dunn ja Harri Pasanen. Nad tegid oma esimese versiooni 1998. aastal. (Dunn, 2011)

Tabel 3: wxPython lühikirjeldus

wxPython	
Autor	Robin Dunn Harri Pasanen
Kirjutanud	Python, C++
Operatsioon süsteemid	mitmeplatvormiline
Viimane versioon	2.8.12.1 (23 juuli 2011)
Litsents	wxWindows
Veebileht	wxpython.org

1.7. MÕSTETE SELETUS

Lisateek on määratud selleks, et tänu selle on võimalik organiseerida dialooge programmis töötades graafilises kasutajaliides (GUI). Lisateegis on olemas põhikomponendid:

- Frame – Raam, mis sisaldab teisi visuaalseid komponente. Raami sisse tuleb silt (*label*).
- Label – silt, mis näitab teksti ja graafilist joonist.
- Entry – koht, kuhu sisestatakse tekst.
- Button – nupp, funktsioonide käivitamiseks.
- Menu – menüü, organiseerib *popup* ja *pulldown* menüüd.
- Message – sõnum, nagu märgistus, aga võimaldab kokku panna pikki ridu ja muuta nende suurust.
- Text – formateeritud tekst, mida on võimalik lisada joonistes ja akendes.

2. PRAKTILINE TEOSTUS

2.1. TKINTER

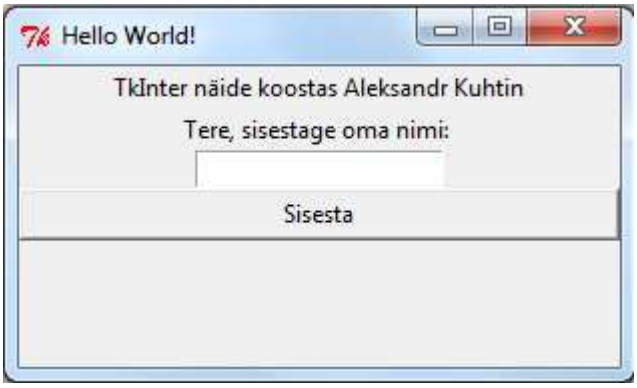
Proovime siis natuke tutvuda TKInterˇiga. Teeme lihtsa *raami*, et aru saada loogikast ja struktuurist.

Hello World – aken.

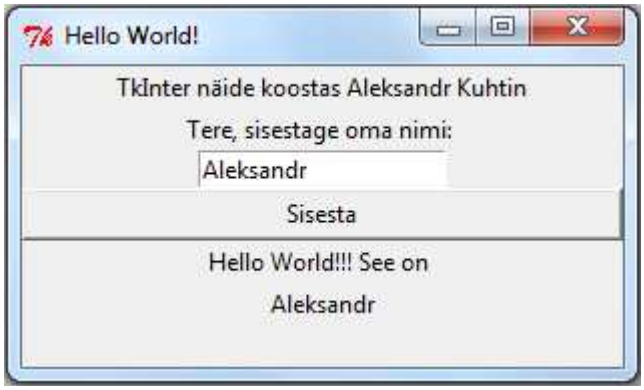
Ma toon näidet, kus on nähtav lähtekood. (vaata Koodinäide 1)

```
import tkinter
from tkinter.constants import *

class App(tkinter.Frame):
    def __init__(w, master=None):
        tkinter.Frame.__init__(w, master)
        w.pack(fill=BOTH)
        w.create_widgets()
    def create_widgets(w):
        w.var = tkinter.StringVar()
        w.var.set('')
        w.sisu = tkinter.Label(w, text="TkInter näide koostas Aleksandr Kuhtin")
        w.sisu.pack()
        w.tekst = tkinter.Label(w, text='Tere, sisestage oma nimi: ')
        w.tekst.pack()
        w.entry = tkinter.Entry(w, textvariable=w.var)
        w.entry.pack()
        w.nupp = tkinter.Button(w, text='Sisesta', command=w.vajutamine)
        w.nupp.pack(fill=BOTH)

        w.f = tkinter.Frame(w.master)
        w.f.pack(fill=BOTH)
        w.f.v = tkinter.StringVar()
        w.f.l = tkinter.Label(w.f, textvariable=w.f.v)
        w.f.l.pack(fill=X)


    def vajutamine(w):
        w.f.sisu1 = tkinter.Label(w, text="Hello World!!! See on ")
        w.f.sisu1.pack()
        w.f.v.set(w.var.get())


if __name__ == '__main__':
    pr = App()
    pr.master.title('Hello World!')
    pr.master.geometry('300x150+500+500')
    pr.mainloop()
```

Koodinäide 1: TkInter "Hello World" lähtekood



Joonis 2: TkInter "Hello World" programm



Joonis 3: TkInter "Hello World" programm peale nuppu vajutamist

Antud näide demonstreerib kõige lihtsamat varianti sellest, mida oskab TkInter. (vaata Joonis 2 ja Joonis 3) Esimene rida impordib TkInterˇi mooduli sisse (alates Pythoni versioonist 3.1, TkInter kirjutatakse „tkinter“):

```
from tkinter import*
from tkinter.constants import *
```

Järgmine funktsioon loob *raami:*

```
class App(tkinter.Frame):
    def __init__ w, master=None):
        tkinter.Frame.__init__(w, master)
        w.pack(fill=BOTH)
        w.create_widgets()
```

Järgmised sammud loovad nuppe, tekste ja ruumi teksti sisestamiseks:

```
    def create_widgets(w):
        w.var = tkinter.StringVar()
        w.var.set('')
```

tkinter.Label() - loob silti tekstide jaoks.

```
        w.sisu = tkinter.Label(w, text="TkInter näide koostas Aleksandr
Kuhtin")
        w.sisu.pack()
        w.tekst = tkinter.Label(w, text='Tere, sisestage oma nimi: ')
        w.tekst.pack()
```

tkinter.Entry() - loob ruumi teksti sisestamiseks.

```
        w.entry = tkinter.Entry(w, textvariable=w.var)
        w.entry.pack()
```

tkinter.Button() - loob nuppu.

```
        w.nupp = tkinter.Button(w, text='Sisesta', command=w.vajutamine)
        w.nupp.pack(fill=BOTH)
```

Järgmised sammud teevad raami teksti jaoks, mis peab tulema peale nuppu vajutamist:

```
        w.f = tkinter.Frame(w.master)
        w.f.pack(fill=BOTH)
        w.f.v = tkinter.StringVar()
        w.f.l = tkinter.Label(w.f, textvariable=w.f.v)
        w.f.l.pack(fill=X)
```

Järgmine funktsioon kirjeldab tegevust, mis tuleb pärast nupule vajutamist. Pärast nupule vajutamist, süsteem peab kirjutama „Hello World!! See on …. (teie nimi)“:

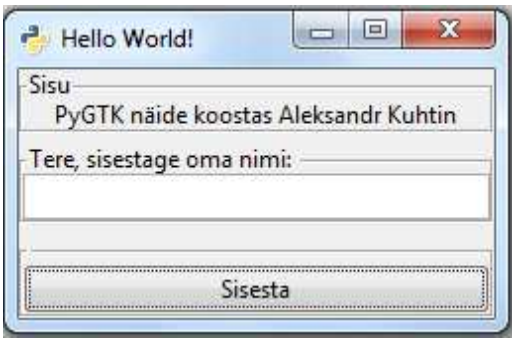
```
    def vajutamine(w):
        w.f.sisu1 = tkinter.Label(w, text="Hello World!!! See on ")
        w.f.sisu1.pack()
        w.f.v.set(w.var.get())
```

Järgmine funktsioon muudab akna pealkirja ja määrab suurust:

```
if __name__ == '__main__':
    pr = App()
    pr.master.title('Hello World!')
    pr.master.geometry('300x150+500+500')
    pr.mainloop()
```

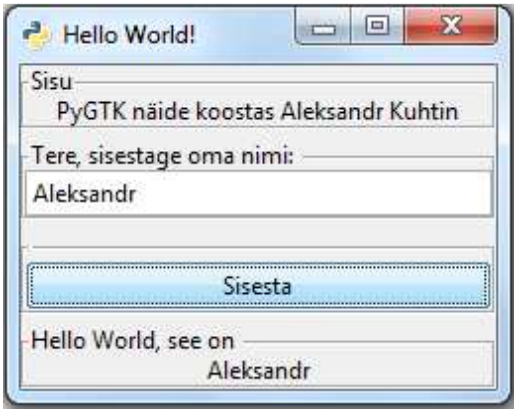
2.2. PYGTK

Teine moodul PyGTK. Proovime siis teha PyGTK mooduliga „Hello World“ raami ja nuppu (vaata Joonis 4), mille peale ka võime vajutada ja funktsiooni aktiveerida. (vaata Joonis 5)



Joonis 4: PyGTK "Hello World" programm

Näeme, et siin on tekst, ruum teksti sisestamiseks ja nupp. Mõlemale oli tehtud raam, sest ühel raamil kaks funktsiooni PyGTK formaadis olla ei saa. Alati on vaja eelkõige teha uus raam ja ainult siis on võimalik lisada nuppe või teksti.



Joonis 5: PyGTK "Hello World" programm peale nuppu vajutamist

Pöörame oma tähelepanu koodile. (vaata Koodinäide 2)

```
# -*- coding: utf-8 -*-
import gtk
class HelloWorld:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect("destroy", lambda w: gtk.main_quit())
        self.window.set_default_size(240, 90)
        self.window.set_title('Hello World!')
        self.vbox = gtk.VBox(False, 5)
        self.window.add(self.vbox)
        frame = gtk.Frame("Sisu")
        label = gtk.Label("PyGTK näide koostas Aleksandr Kuhtin")
        frame.add(label)
        self.vbox.pack_start(frame, False, False, 0)
        frame = gtk.Frame("Tere, sisestage oma nimi: ")
        entry = gtk.Entry()
        frame.add(entry)
        entry.add_events(gtk.gdk.KEY_RELEASE_MASK)
        entry.connect("key-release-event", self.on_key_release)
        self.vbox.pack_start(frame, False, False, 0)
        frame = gtk.Frame("")
        nupp = gtk.Button('Sisesta')
        nupp.connect('clicked', self.vajutamine)
        frame.add(nupp)
        self.vbox.pack_start(frame, False, False, 0)
        self.window.show_all ()
    def on_key_release(self, widget, event):
        self.vastus = widget.get_text()
    def vajutamine(self,nupp):
        print 'Hello World!'
        self.frame = gtk.Frame("Hello World, see on ")
        self.label = gtk.Label("")
        self.frame.add(self.label)
        self.vbox.pack_start(self.frame, False, False, 0)
        self.label.set_text(self.vastus)
        self.window.show_all ()
def main():
        gtk.main()
        return 0
if __name__ == '__main__':
        HelloWorld()
        main()
```

Koodinäide 2: PyGTK "Hello World" lähtekood

Selline kood annab võimalust kasutada „ä, ö, ü jne“ programmis.

```
# -*- coding: utf-8 -*-
```

Nagu kõikide moodulite jaoks, on vaja esiteks importida lisateeke. Kui tegemist on mahuka programmiga, siis lisatakse ka *„import pygtk“*. Kuna ma tegin kerge versiooni, antud *pygtk* lisateek vajalik ei ole.

```
Import gtk
```

Klass, kus määratakse akna suurused ja eriomadused.

```
class HelloWorld:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect("destroy", lambda w: gtk.main_quit())
        self.window.set_default_size(240, 90)
```

Paneb aknale pealkirju.

```
        self.window.set_title('Hello World!')
```

Vbox – vertikaalne ala, on olemas ka horisontaalne. Esiteks on vaja enda jaoks ära otsustada, kuidas peab teie programm välja nägema.

```
vbox = gtk.VBox(False, 5)
self.window.add(vbox)
```

Loome raami ja anname sellele nime.

```
frame = gtk.Frame("Sisu")
```

Selle raami sees loome silti tekstiga.

```
label = gtk.Label("PyGTK näide koostas Aleksandr Kuhtin")
```

Siin määrame missugust silti missugusesse raami paneme.

```
frame.add(label)
```

Siin määrame, kuidas peab meie raam välja nägema. (vertikaalselt)

```
vbox.pack_start(frame, False, False, 0)
```

Loome siin ruumi teksti sisestamiseks ja anname käsklust, et ta loeks, mida me sisestame.

```
frame = gtk.Frame("Tere, sisestage oma nimi: ")
entry = gtk.Entry()
frame.add(entry)
entry.add_events(gtk.gdk.KEY_RELEASE_MASK)
entry.connect("key-release-event", self.on_key_release)
self.vbox.pack_start(frame, False, False, 0)
```

Samamoodi on vaja teha nuppude jaoks, luua uus raam ja otsustada missuguses positsioonis peab ta olema. Näidisnupp on paigaldatud vertikaalselt. Lisame ka käskluse nuppu funktsiooni vajutamiseks ja selle funktsiooni me kirjeldame eraldi.

```
frame = gtk.Frame("")
nupp = gtk.Button('Sisesta')
nupp.connect('clicked', vajutamine)
frame.add(nupp)
vbox.pack_start(frame, False, False, 0)
```

Selline käsklus loeb kõike, mida me kirjutasime klassis ja konstrueerib programmi. Mis on unikaalne selles moodulis, et ei ole vaja teha erinevaid muutujaid. Ükskõik, kas meil on üks raam või rohkem, nende jaoks võib olla üks muutuja.

```
self.window.show_all ()
```

Meetod, mis võtab teksti sisestamiseruumist.

```
def on_key_release(self, widget, event):
    self.vastus = widget.get_text()
```

See on meetod, kus kirjeldatakse, mida peab programm tegema pärast nupule vajutamist. Kui nupule vajutatakse, siis luuakse *frame*ˇid ja *label*ˇid uute tekstide jaoks.

```
def vajutamine(nupp):
    print 'Hello World!'
    self.frame = gtk.Frame("Hello World, see on ")
    self.label = gtk.Label("")
    self.frame.add(self.label)
    self.vbox.pack_start(self.frame, False, False, 0)
    self.label.set_text(self.vastus)
    self.window.show_all ()
```

Põhifunktsioon, mis käivitab kogu programmi.

```
def main():
        gtk.main()
        return 0
```

Siin määratakse, missugused põhiosad on kasutusel programmis.

- HelloWorld() – klass
- Main() – põhifunktsioon

```
if __name__ == '__main__':
        HelloWorld()
        main()
```

2.3. PYQT

Kolmas moodul on PyQt. (Maliński, 2008) Väga mugav ja võimekas moodul, mis sisaldab oma keskkonnas ka teisi mooduleid. Proovime ka selle mooduliga teha akent „Hello World“.
PyQt „Hello World “ näidis, mis koosneb kolmest failist: *„test.pyw“*(vaata Koodinäide 3), *„mainform.py“*(vaata Koodinäide 4), *„mainform.ui“*(vaata Joonis 6).

Test.pyw - põhifail, mis käivitab kogu programmi.

```
import sys
from PyQt4 import QtCore, QtGui
import mainform

def main():
    app = QtGui.QApplication(sys.argv)
    form = mainform.MainForm()
    form.show()
    app.exec()

if __name__ == "__main__":
    sys.exit(main())
```

Koodinäide 3: PyQt test.pyw programm

Esiteks on vaja importida PyQt põhimooduleid QtCore, QtGui.

```
from PyQt4 import QtCore, QtGui
```

Siis importime mooduli mainform`i, et kujundada akent.

```
import mainform
```

Järgmiseks on vaja luua põhiprogrammi objekt.

```
app = QtGui.QApplication(sys.argv)
```

Loome objekti vormi.

```
form = mainform.MainForm()
```

See on käsklus objekti vormi ja sisu näitamiseks.

```
form.show()
```

Programmi käivitamise funktsioon.

```
app.exec()
```

Mainform.py – fail, kus kirjeldatakse akna suurusi ja eriomadusi.

```
from PyQt4 import QtCore, QtGui, uic

class MainForm(QtGui.QDialog):

    def __init__(self):

        super(MainForm, self).__init__()

        uic.loadUi("mainform.ui", self)

        self.label_4.hide()
```

Koodinäide 4: PyQt mainform.py lähtekood

Esiteks on vaja importida PyQt põhimooduleid QtCore, QtGui ja uic.

```
from PyQt4 import QtCore, QtGui, uic
```

Põhivormi prototüübi koostamine.

```
class MainForm(QtGui.QDialog):
```

Konstruktor.

```
def __init__(self):
    super(MainForm, self).__init__()
```

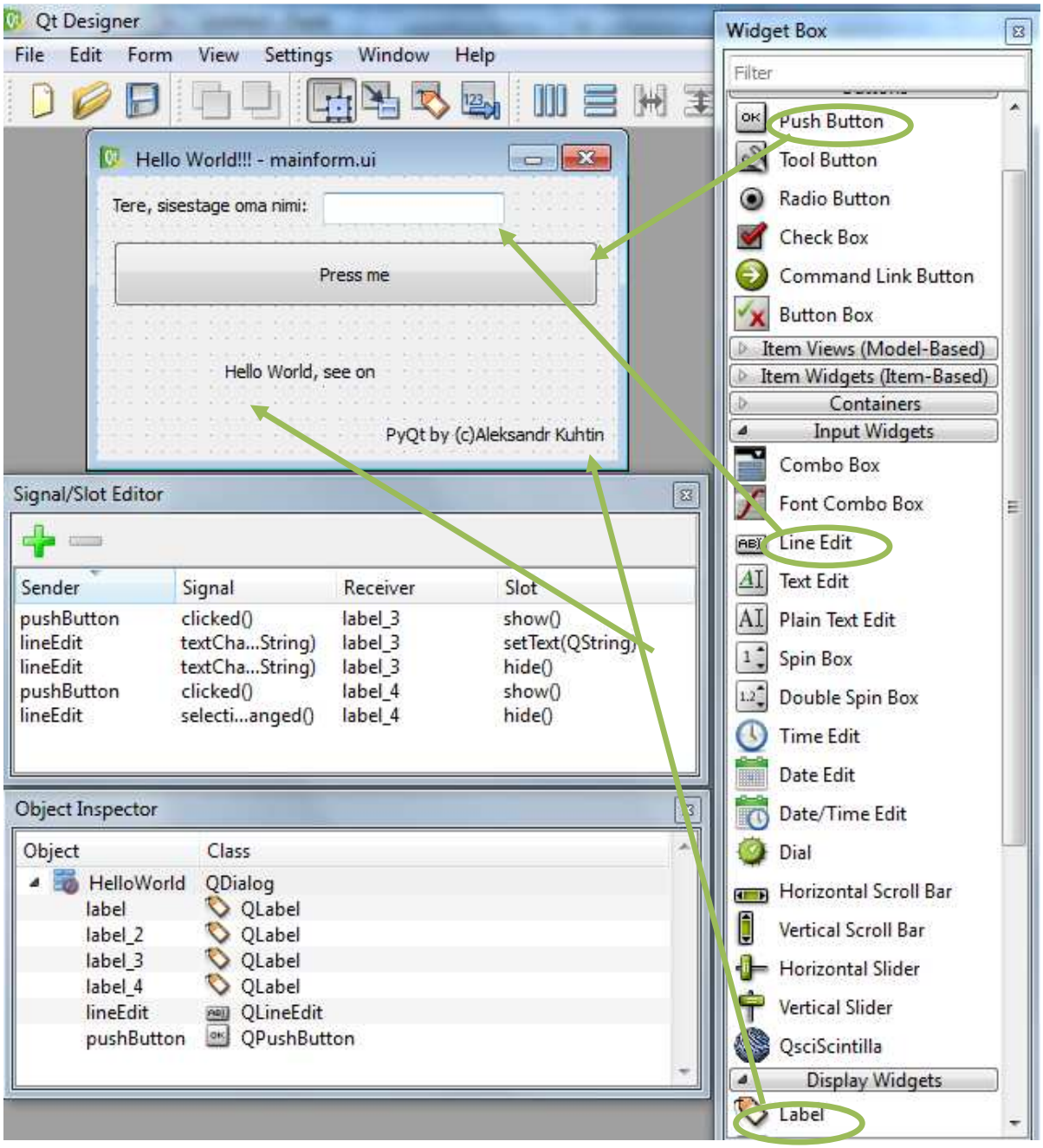
Visuaalselt laadib mainform`i, kui kujundust.

```
uic.loadUi("mainform.ui", self)
```

Label_4 peetakse.

```
self.label_4.hide()
```

Mainform.ui – fail, mis on tehtud Qt Designer programmis.



Joonis 6: mainform.ui (QtDesigner)

Visuaalselt koostatakse kujundust raami jaoks.

- Push Button – tegin nupu valmis.
- Label – lõin ruumi, kuhu ma kirjutasin teksti sisse.
- Line Edit – ruum teksti sisestamiseks.

Signal/Slot Editor – vajutades F4, te võite siseneda sellises töökorralduses, kus on võimalik lisada funktsioone nuppude ja tekstide jaoks.

Kui vajutatakse nupule, siis label_3 tuleb nähtavale.

pushButton clicked() label_3 show()

Kui sisestatakse/muudetakse tekst, siis label_3 kirjutatakse tähendusi.

lineEdit textChanged(QString) label_3 seText(QString)

Kui sisestatakse/muudetakse tekst, siis label_3 peetakse.

lineEdit textChanged(QString) label_3 hide()

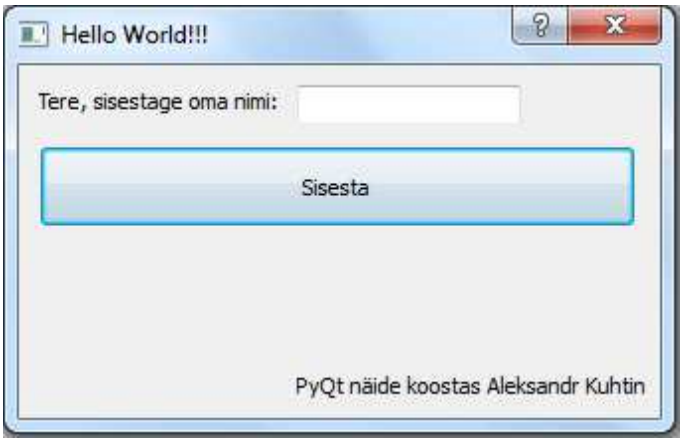
Kui vajutatakse nupule, siis label_4 tuleb nähtavale.

pushButton clicked() label_4 show()

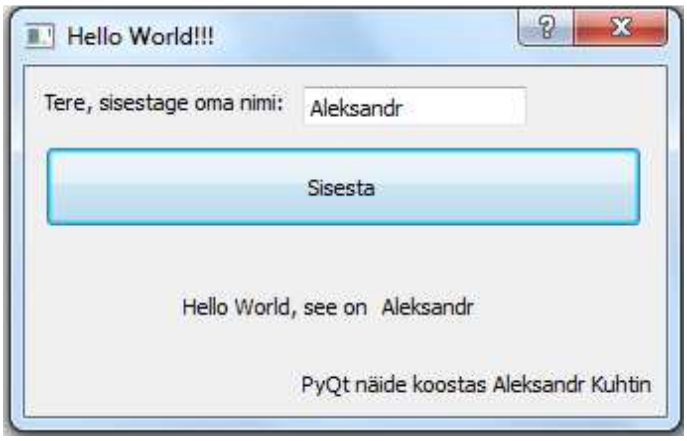
Kui sisestamise tekst eristatakse, siis label_3 peetakse.

lineEdit selectionChanged() label_3 hide()

Kui kõik need kolm faili on tehtud korralikult, siis kui me käivitame Pythonis faili *„test.pyw“*, võime näha oma raame „Hello World“. (vaata Joonis 7 ja Joonis 8)



Joonis 7: PyQt "Hello World" programm

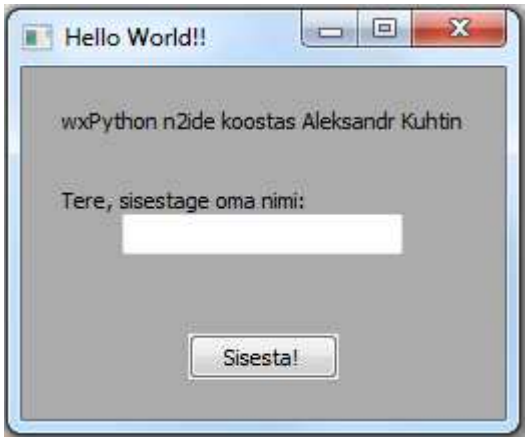


Joonis 8: PyQt "Hello World" programm peale nuppu vajutamist

2.4. WXPYTHON

wxPython – viimane moodul, millest ma oma töös kirjutan. (Bodnar, 2011)

Proovime selle mooduli abiga teha sama akent (vaata Joonis 9), kus on teksti koht ja nupp, millele saab vajutada. (vaata Joonis 10)



Joonis 9: wxPython "Hello World" programm

Joonis 10: wxPython "Hello World" programm peale nuppu vajutamist

```
import wx

class Frame(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, title=title)

        self.tekst = wx.StaticText(self,
                                   label="wxPython n2ide koostas Aleksandr Kuhtin")
        self.tekst1 = wx.StaticText(self, -1,
                                    "Tere, sisestage oma nimi: ",wx.Point(20,60))
        self.kusimus = wx.TextCtrl(self,
                                   20,
                                   "",
                                   wx.Point(170, 60),
                                   wx.Size(140,-1))
        self.nupp = wx.Button(self, label="Sisesta!")
        self.nupp.Bind(wx.EVT_BUTTON, self.Teade)

        Sizer = wx.BoxSizer(wx.VERTICAL)
        Sizer.Add(self.tekst, 0, wx.ALIGN_CENTER|wx.ALL, 20)
        Sizer.Add(self.kusimus, 0, wx.ALIGN_CENTER|wx.ALL, 20)
        Sizer.Add(self.nupp, 0, wx.ALIGN_CENTER|wx.ALL, 20)

        self.SetSizerAndFit(Sizer)

        wx.EVT_TEXT(self, 20, self.EvtText)

    def EvtText(self, event):
        self.vastus = wx.StaticText(self, -1,
                                    'Hello World! See on %s\n' % event.GetString(),
                                    wx.Point(20, 100))
        self.vastus.Hide()

    def Teade(self, event):
        self.vastus.Show()

app = wx.App(redirect=False)
frame = Frame(None, "Hello World!!")
frame.Show()
app.MainLoop()
```

Koodinäide 5: wxPython "Hello World" lähtekood

Nüüd analüüsime lähtekoodi. (vaata Koodinäide 5)

Nagu teistel moodulitel on vaja wxPython mooduli esialgselt importida.

```
import wx
```

Klass “Frame”, kus määratakse akna sisu.

- **`wx.StaticText()`** – see on funktsioon, et tekitada ruumi, kuhu oleks võimalik sisestada oma teksti.
- **`wx.Button()`** – loob nuppu, koos tekstiga.
- **`_.Bind(wx.EVT_BUTTON, self. )`** – funktsioon nuppudele vajutamiseks. „EVT.BUTTON“ peab kirjutama suurte tähtedega, kui kirjutada väikestega, siis wxPython moodul ei tunne funktsioone.
- **`wx.TextCtrl()`** – see on funktsioon ruumi loomiseks, kuhu on võimalik teksti sisestada.

```
class Frame(wx.Frame):
```

```
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, title=title)
        self.tekst = wx.StaticText(self, label="wxPython n2ide koostas
Aleksandr Kuhtin")
        self.tekst1 = wx.StaticText(self, -1, "Tere, sisestage oma nimi:
",wx.Point(20,60))
       self.kusimus = wx.TextCtrl(self, 20, " ", wx.Point(170, 60),
wx.Size(140,-1))
        self.nupp = wx.Button(self, label="Sisesta!")
        self.nupp.Bind(wx.EVT_BUTTON, self.Teade)
```

Sizer – kui teised moodulid saavad iseseisvalt paigutada nii tekste, kui ka nuppe, siis wxPythoni jaoks on vaja eraldi teha paigutamine, sest ta ei tee järjekordset paigutamist. Selleks on vaja määrata igale elemendile kirjeldust, suurust jne.

```
Sizer = wx.BoxSizer(wx.VERTICAL)
Sizer.Add(text, 0, wx.ALIGN_CENTER|wx.ALL, 20)
Sizer.Add(self.kusimus, 0, wx.ALIGN_CENTER|wx.ALL, 20)
Sizer.Add(nupp, 0, wx.ALIGN_CENTER|wx.ALL, 20)
self.SetSizerAndFit(Sizer)
```

See on funktsioon, mis kirjutab kohe tekste, mis on sisestamise seisukorras.

```
wx.EVT_TEXT(self, 20, self.EvtText)
```

See on funktsioon sisestatud tekstide näitamiseks. **`.GetString()`** – see funktsioon võtab bufferist tekste. Sinna on veel lisatud funktsioon, mis paneb **`self.vastus`** tähendust peitu.

```
def EvtTekst (self, event):
    self.vastus = wx.StaticText(self, -1, 'Hello World! See on %s\n' %
event.GetString(), wx.Point(20, 100))
    self.vastus.Hide()
```

See on meetod, mille abil tehakse funktsiooni nuppudele vajutamiseks. Siin on määratud, et kui vajutatakse nupule, siis tuleb **`self.vastus`** nähtavaks.

```
def Teade(self, event):
    self.vastus.Show()
```

Programmi käivitamise funktsioonid.

```
app = wx.App(redirect=False)
frame = Frame(None, "Hello World!!")
frame.Show()
app.MainLoop()
```

KOKKUVÕTE

Antud töö eesmärgiks oli teada saada, mis moodulid üldse olemas on ja kuidas nendega töötada kasutades Python`it.
Me saime teada veidi sellistest moodulitest: TkInter, PyQt, wxPython ja PyGTK. Nad on omavahel erinevad ja igaüks on omapärane.

TkInter on stabiilne ja võimekas moodul, mis paigaldab python`i kausta manuaalselt ja töötab suurepäraselt. Toimetab ka uut Python`i versiooni ja selle mooduliga on võimalik töötada ka 64-bitilises süsteemis.

PyQt on väga mugav ja lihtne tänu sellele, et on olemas eraldi tehtud redaktor, mis genereerib iseseisvalt Python`i koodi. Minu arvates, kõige mugavam ja kergem kasutamises moodul. Väga palju võimalusi. Selle mooduli abil võib teha ilusaid ja mahukaid programme.

wxPython on hea moodul, aga, minu arvates, on seal puudu mõned funktsioonid, mis võiksid toimida automaatselt. Ma räägin paigutamisest, sest seda peab tegema arendaja ja muidu see võtab aega, ning kood tuleb suurem.

PyGTK on kapriisne moodul, sest ta töötab ainult Python`i versiooniga 2.7 ja varem. Veel üks tingimus – Python`i programm peab olema 32-bitiline. On olemas pakk *„all-in-one“*, mis teeb installeerimist lihtsamaks, aga kui te proovisite alam-mooduleid iseseisvalt paigaldada siis võis olla tõrkeid. Mõnikord on probleeme DLL failidega ja võivad olla probleemid kompileerimises. PyGTK 2.24 versioon ei ole stabiilne, kui on vaja kasutada PyGTK, siis võtke varem versioon, nt 2.22.

Ma proovisin nelja moodulit ja mulle meeldis PyQt4. Võin tähelepanu pöörata ka TkInter`ile, sest TkInter on ka päris mugav oma loogilises struktuuris. Sain teada, et selliste moodulite abiga on võimalik teha unikaalseid ja kasulikke programme.

KASUTATUD KIRJANDUS

Bodnar, J. (16. Oktoober 2011. a.). *The wxPython tutorial*. Kasutamise kuupäev: 7. November 2011. a., allikas ZetCode: http://www.zetcode.com/wxpython/

Chun, W. J. (2006). *Core Python Programming, Second Edition.* New Jersey: Prentice Hall.

Didkovski, I. (2002). *Языки программирования*. Kasutamise kuupäev: 12. November 2011. a., allikas http://e-skin.hut.ru/langs

Dunn, R. (29. September 2011. a.). *How to learn wxPython*. Kasutamise kuupäev: 7. November 2011. a., allikas wxPython: http://wiki.wxpython.org/How%20to%20Learn%20wxPython

Finlay, J. (2. Märts 2006. a.). *PyGTK 2.0 Tutorial*. Kasutamise kuupäev: 7. November 2011. a., allikas PyGTK Team : http://www.pygtk.org/pygtk2tutorial/

Lundh, F. (3. November 1999. a.). *An Introduction to Tkinter*. Kasutamise kuupäev: 7. November 2011. a., allikas Information and Telecommunication Technology Center: http://www.ittc.ku.edu/~niehaus/classes/448-s04/448-standard/tkinter-intro.pdf

Maliński, P. (14. Juuli 2008. a.). *Introduction to PyQt4*. Kasutamise kuupäev: 7. November 2011. a., allikas http://www.rkblog.rk.edu.pl/w/p/introduction-pyqt4/

Petuhhov, I. (2009). *Moodulid*. Kasutamise kuupäev: 7. November 2011. a., allikas Tallinna Ülikool: http://www.cs.tlu.ee/~inga/progbaas/Materjalid/Python_funktsioon_2009.pdf

Python Software Foundation. (6. November 2011. a.). *Modules*. Kasutamise kuupäev: 7. Novembe 2011. a., allikas Python: http://docs.python.org/tutorial/modules.html

Riverbank Computing Limited. (kuupäev puudub). *What is PyQt?* Kasutamise kuupäev: 7. November 2011. a., allikas Riverbank Computing: http://www.riverbankcomputing.co.uk/software/pyqt/intro

TIOBE Software BV. (Oktoober 2011. a.). *TIOBE Programming Community Index for October 2011*. Kasutamise kuupäev: 7. November 2011. a., allikas Tiobe: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

LISAD

Lisa 1:

Position Oct 2011	Position Oct 2010	Delta in Position	Programming Language	Ratings Oct 2011	Delta Oct 2010	Status
1	1	=	Java	17.913%	-0.25%	A
2	2	=	C	17.707%	+0.53%	A
3	3	=	C++	9.072%	-0.73%	A
4	4	=	PHP	6.818%	-1.51%	A
5	6	↑	C#	6.723%	+1.76%	A
6	8	↑↑	Objective-C	6.245%	+2.54%	A
7	5	↓↓	(Visual) Basic	4.549%	-1.10%	A
8	7	↓	Python	3.944%	-0.92%	A
9	9	=	Perl	2.432%	+0.12%	A
10	11	↑	JavaScript	2.191%	+0.53%	A
11	10	↓	Ruby	1.526%	-0.41%	A
12	12	=	Delphi/Object Pascal	1.104%	-0.45%	A
13	13	=	Lisp	1.031%	-0.05%	A
14	14	=	Transact-SQL	0.909%	+0.09%	A
15	23	↑↑↑↑↑↑↑↑	PL/SQL	0.903%	+0.30%	A-
16	24	↑↑↑↑↑↑↑↑	Lua	0.802%	+0.25%	A
17	16	↓	RPG (OS/400)	0.757%	+0.05%	A--
18	15	↓↓↓	Pascal	0.721%	-0.05%	A
19	-	=	Assembly*	0.622%	-	B
20	17	↓↓↓	Ada	0.609%	-0.09%	B