Tallinna Ülikool

Digitehnoloogiate Instituut

Qt raamistiku võimalused graafilise kasutajaliidesega rakenduse loomisel Python keeles

Seminaritöö

Autor: Alex Neil

Juhendaja: Inga Petuhhov

Tallinn 2016

Sisukord

Sissejuhatus

Käesoleva seminaritöö eesmärk on tutvustada võimalusi, mida pakub Qt raamistik Python keeles graafilise kasutajaliidesega (GUI) rakenduse loomisel. Selle eesmärgi saavutamiseks plaanib autor koostada lühikese juhendi näidetega.

Juhend on mõeldud eelkõige õpilastele, kellel on juba Python keele ja objektorienteerituse põhimõtete algteadmised olemas, ning kes soovivad teada, kuidas luua rakendusi, millel on graafiline kasutajaliides. Juhendi mõistmiseks ei ole Qt raamistiku tundmine vajalik.

Loodud juhend antakse lugemiseks ja tutvumiseks kasutajatele, kellel on mõningased teadmised Python keelest. Tagasiside põhjal teeb autor vajadusel juhendisse täiendusi, et juhend oleks paremini arusaadav ja kasutatav.

Töö aktuaalsuse ja vajaduse põhjendus

Python on populaarne programmeerimiskeel. Python keele eelisteks on selle lihtsus, kiirus ja paindlikkus graafilise kasutajaliidese loomisel. Qt on üks enim kasutatav ja palju võimalusi pakkuv raamistik, mida kasutatakse GUI arendamisel.

Kuid samas on paljudel inimestel raskusi graafilise kasutajaliidesega rakenduse arendamise õppimisega. Kõige levinumaks põhjuseks on see, et isegi ei teata kust alustada ja kogu protsess tundub olevat liiga keeruline.

Autor loodab, et koostatav lühijuhend võiks saada kergesti loetavaks ja kasutatavaks materjaliks, et Qt mooduli näidete kaudu stimuleerida Python keele õppimist ja luua edasist huvi selle keele vastu.

Töö struktuur

Käesolev töö on jaotatud kolmeks peatükiks.

Esimeses peatükis räägitakse GUI-st üldiselt. Autor annab lühiülevaate sellest, mis on GUI, antakse lühike ülevaate GUI loomisest Python keeles, tutvustatakse Qt raamistiku ja seletatakse GUI rakenduse töötamise põhiprintsiipi.

Teises peatükis tutvustab autor PyQt raamistikku, räägib selle paigaldusest, annab mõnede PyQt moodulite kirjelduse ning jätkab samm-sammult PyQt kasutamise õpetusega.

Kolmandas peatükis tutvustab autor tööriista Qt Designer, mis oluliselt lihtsustab Qt abil graafilise kasutajaliidesega rakenduse loomist.

1. Graafilise kasutajaliidese loomine Python keeles

Python keel sobib hästi konsooliga töötamiseks, mis võib olla piisav inimestele, kellele meeldib terminali aknasse käske käsitsi sisse trükkida. Kuid Python sobib suurepäraselt ka kõiksuguste multiplatvormiliste rakenduste kiireks arendamiseks, kaasa arvatud graafiliste kasutajaliidestega programmide jaoks. Python keelega saab vähese lisavaevaga lisada kaasaegse GUI, mida on kergem kasutada, mis oleks palju atraktiivsem ning mis ei nõua palju pingutust.

1.1. Graafiline kasutajaliides

Madalaimal tasemel GUI kujutab endast akent (kas nähtavat või nähtamatut), millel on järgmised elemendid: juhtelemendid, menüü, paigutus ja interaktsioon (Jaganmohan & Loganathan, 2016):

- Juhtelemendid: nupud, tekstiväljad, sildid, jne.
- Menüü, mis asub tavaliselt GUI akna ülemises servas ja annab kasutajale võimaluse juhtida rakendust. Menüüs võivad asuda nupud ja teised juhtelemendid igasuguste käskude valimiseks.
- Paigutus: viis juhtelementide paigutuseks, mis on väga tähtis hea GUI disaini jaoks.
- Interaktsioon: toimub sisend-/väljundseadmete kasutamisel, nagu näiteks hiir ja klaviatuur.

GUI rakenduse arendamine toimub eelpoolnimetatud komponentide ümber ja kõige suurem väljakutse on interaktsiooni ala kujundamine. Õige sündmuste, kuularite (ingl *listeners*) ja sündmusetöötlejate kujundamine aitab luua parema GUI rakenduse.

On olemas palju erinevaid raamistikke, mis aitavad kaasa graafilise kasutajaliidese arendamisele. Nende raamistike kasutamine teeb GUI programmeerimise lihtsamaks.

1.2. Võimalused GUI arendamiseks Python keeles

Python pakub laia valikut teda toetavaid graafilisi raamistikke, näiteks selliseid populaarseid raamistikke nagu Tkinter, PyQt, PyGTK, wxPython, Pygames ja teisi. Ulatusliku nimekirja võib leida järgmisest allikast: https://wiki.python.org/moin/GuiProgramming.

Lühike ülevaade kõige levinumatest valikutest:

- **Tkinter**. See on standardse Pythoni paigalduspaketi osa ja paigaldatakse automaatselt koos Pythoniga. See pakub algelise graafilise kasutajaliidese lihtsa kujundusega. Miinuseks on, et Pythoni oma Tkinter-i dokumentatsioon on üsna minimaalne ja vidinate arv on piiratud. Tkinter-it ei ole kaua aega eriti uuendatud.
- **Kivy**. Hea valik atraktiivsete tahvelarvutistiilis liideste jaoks. See on üsna noor, aga samas paljulubav projekt. Sobib väga hästi arendamaks rakendusi tahvelarvutitele ja nutitelefonidele, kuna see toetab hästi näpuga ekraanil rakenduse juhtimist (ingl *swipe*).
- **Pygame**. See on raamistik, mis on loodud peamiselt 2D mängude ja multimeedia rakenduste arendamiseks.
- **Qt.** Üks parimaid raamistikke traditsiooniliste lauaarvutistiilis graafiliste kasutajaliideste loomiseks. See on võimekas ja palju võimalusi pakkuv raamistik, mis näeb erinevatel süsteemidel välja süsteemile omase stiiliga. Qt jaoks on olemas kaks seoste kogumit: PyQt, mis on vanem ja hästi väljaarenenud projekt, aga tasuta ainult juhul kui arendatav rakendus on avatud lähtekoodiga, ja PySide, mis on noorem, aga samas vabamate õigustega litsentsiga (LGPL). PyQt võib kasutada koos Qt Designer-iga – „*drag and drop*“ ja WYSIWYG (mida näed, selle saad) stiilis graafilise kasutajaliidese redaktoriga.

Python keeles tuleb graafilise kasutajaliidese arendamiseks teha valik ja otsustada, millist teeki kasutada. Käesoleva töö eesmärk on tutvustada Qt teegi võimalusi Python keeles. Qt-d toetavatest raamistikest valiti PyQt versioon 5, mis on üks vanimaid ja väljaarenenuimaid projekte, ja mis lubab kasutada kõiki Qt-s olemasolevaid võimalusi.

1.3. Qt raamistiku tutvustus

Qt on rohkelt võimalusi pakkuv mitmeplatvormiline graafilise kasutajaliidese arendamise teek. Qt raamistikku kuuluvad võrgutugi, lõimed (ingl *threads*), tugi Unicode kasutamiseks, regulaaravaldiste tugi, SQL andmebaaside tugi, SVG graafikatugi, OpenGL, XML, täisfunktsionaalne veebilehitseja, multimeediamoodulid

ja rikkalik valik graafilise kasutajaliidese vidinaid. Qt standardne installatsioon sisaldab veel Qt Designer-it – graafilise kasutajaliidese redaktorit.

Qt on kirjutatud C++ programmeerimiskeeles. Peale C++ keele võib Qt-d kasutada veel selliste keeltega nagu Python, Java, PHP, Perl ja paljud teised.

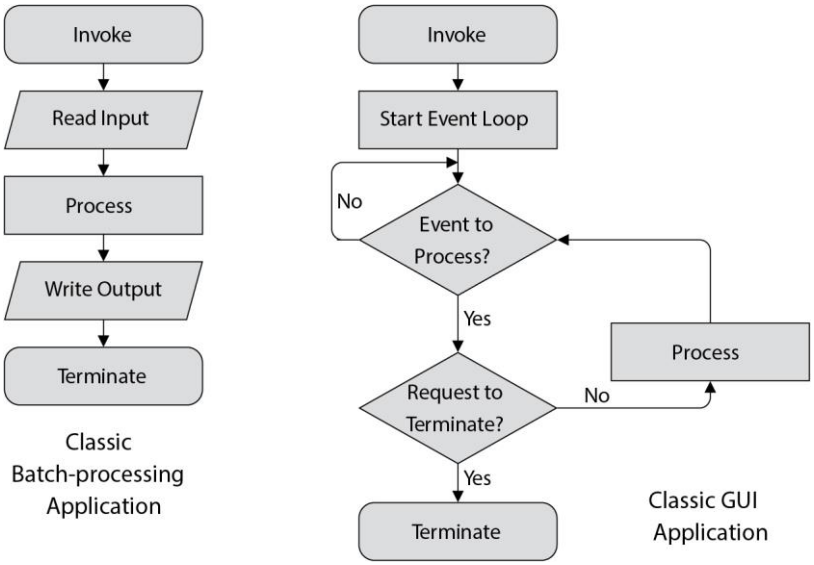
Qt raamistik on kaitstud GPL ja LGPLv3 (*Lesser General Public License*) litsentsidega. See tähendab, et raamistiku lähtekood on vabalt kättesaadav ning igaüks võib seda alla laadida, levitada ja muuta.

1.4. GUI rakenduse üldine töötamise printsiip

Vaatamata GUI loomise instrumentide suurele valikule ja nende näilisele keerukusele on nende üldine tööpõhimõte lihtne (Цилюрик, 2014):

- Luuakse lõpmatu tsükkel (peatsükkel), milles kuulatakse kasutaja poolt genereeritud sündmusi (klaviatuurilt sisestamine, hiireklahvi vajutus, kursori liikumine jne).
- Iga võimaliku (töödeldava) sündmuse jaoks määratakse tagasikutse (ingl *callback*) funktsioon. See funktsioon kutsutakse välja juhul, kui toimub kuulatav sündmus.
- Kõiki sündmuseid, millele ei olnud määratud tagasikutse funktsioone, ignoreeritakse.
- Igal rakenduse peatsükli kordumisel analüüsitakse eelmisest kordusest saabunud sündmuseid ja iga sellise sündmuse jaoks kutsutakse temale määratud funktsioon. Kui saabunud sündmusi on mitu, siis töödeldakse neid saabumise järjekorras.
- Iga graafilise kasutajaliidese instrument omab mingit graafiliste vidinate kogumit ja vahendeid nende vidinate paigutamiseks rakenduse aknas. Iga komponent omab spetsiifilist sündmuste kogumit, mida ta suudab genereerida.

Sellist lähenemist nimetatakse sündmusjuhitav programmeerimine (ingl *event-driven programming*) (Joonis 1). Kõik käesolevas juhendis toodud rakenduste näited kasutavad seda mudelit.



Joonis 1. Protseduurne rakendus vs GUI rakendus (Summerfield, 2008)

2. PyQt lühijuhend

Antud töös tutvustatakse Qt raamistiku kasutamist Python keeles läbi PyQt seoste kogumi (ingl *bindings*). Juhendisse on lisatud mõned autori arvates kõige levinumad ja lihtsamad PyQt komponendid. Töös on näidetena kasutatud autori enda tehtud koodinäiteid ja internetis vabalt levivaid õpetusi. Näited antakse koos programeerimiskoodi lahtiseletustega ja piltidega.

Python programmeerimiskeel toetab nii protseduurses kui ka objektorienteeritud stiilis kirjutamist. Kuna PyQt5 eeldab eelkõige objektorienteeritud programmerimist, kasutatakse selles juhendis edaspidi just seda lähenemist.

Antud juhendis allkirjastatakse ja nummerdatakse vaid suuremad ja terviklikud koodinäited, sama koodi üksikud laused ja osad, mida kasutatakse koodi seletuseks parema loetavuse huvides, on jäetud allkirjastamata.

2.1. PyQt tutvustus

PyQt on Pythoni liides Qt raamistikuga töötamiseks, mis ühendab kõik Pythoni ja Qt eelised. See ühendab endas Qt raamistiku võimu ja Pythoni lihtsuse. PyQt arendaja on Riverbank Computing Ltd. Ametlik kodulehe aadress on: www.riverbankcomputing.com

PyQt toetab kõiki süsteeme, mis on toetatud Qt poolt, sealhulgas Windows, Unix, OS X, Linux, iOS ja Android. PyQt on ehitatud Pythoni moodulite kogumina ja tal on rohkem kui 1000 klassi ja mõnituhat funktsiooni ning meetodit. PyQt on kaetud kahe litsentsitüübiga. Arendajad saavad valida GNU GPL v3 ja kommertslitsentsi vahel.

Kuna Qt versioon 4.x ei ole enam Qt arendaja poolt toetatud, kasutab see juhend PyQt versiooni 5, mis toetab Qt v5 ja Pythoni versiooni 3.x. PyQt5 ei ühildu varasema PyQt4 versiooniga.

PyQt standardne allalaaditav installerimispakend ei sisalda Qt raamistiku installatsiooni ja kasutajal tuleb endal soetada Qt paigalduskomplekt. Aga samas on olemas GPL litsentsiga PyQt5 paigalduseks nn *wheel* (Pythoni mooduli paigalduspakett) koos Qt LGPL versiooniga.

2.2. PyQt paigaldus

See juhend eeldab, et Pythoni versioon 3.5 või hilisem on juba eelnevalt paigaldatud, vastasel juhul tuleb see paigaldada.

Kui see samm on sooritatud, saab jätkata PyQt paigaldamisega.

PyQt5 võib paigaldada kasutades *Wheel* vahendit. *Wheel* on uus Pythoni levitamise standard, mida kasutatakse Pythoni moodulite paigaldamiseks. Toetatud on ainult Pythoni versioon 3.5 või hilisem. *Wheel* installerimispakendid on kättesaadavad järgmistel süsteemidel: 32- ja 64-bit Windows, 64-bit OS X ja 64-bit Linux (Riverbank Computing Ltd, 2016).

Wheel-e paigaldatakse kasutades **pip3** programmi, mis on osa viimaste Pythoni versioonide paigalduspakettidest.

PyQt5 GPL litsentsiga *wheel*-i paigaldamine käivitatakse käsurealt:

```
>pip3 install pyqt5
```

See paigaldab kasutatava platvormi ja Pythoni versiooni jaoks sobiva *wheel*-i (eeldades, et mõlemad on toetatud). Wheel laaditakse automaatselt alla arvutisse.

Kui saadakse veateade, mis ütleb „*no downloads could be found that satisfy the rquirement*“, siis tõenäoliselt süsteemis olevat Pythoni versiooni ei toetata.

Qt-d ei pea eraldi paigaldama, kuna PyQt5 *wheel* juba sisaldab kõiki vajalikke vaba litsentsiga (LGPL) Qt komponente.

PyQt5 eemaldamiseks käivitatakse:

```
>pip3 uninstall pyqt5
```

Kui kasutatakse Microsoft Windows-i ja saadakse mingeid veateateid, tasub proovida käivitada CMD administraatori õigustega.

2.3. Põhiklassid ja moodulid

PyQt API on suur kollektsioon erinevatest klassidest ja meetoditest. Kuna klasse on väga palju, grupeeriti need rohkem kui 20 moodulisse. Allpool tuuakse ära mõningad enimkasutatavad moodulid (TutorialsPoint, kuupäev puudub):

- QtCore

  Sisaldab tuuma (ingl *core*) klasse, mis ei ole GUI klassid. Seda moodulit kasutatakse töötamiseks ajaga, failide ja kaustadega, erinevate andmetüüpidega, andmevoogudega, URL-idega, lõimete (ingl *threads*) ja protsessidega.
- QtGui

  Graafilise kasutajaliidese komponendid.
- QtMultimedia

  Sisaldab klasse multimeedia programmeerimiseks.
- QtWidgets

  Koosneb klassidest, mis sisaldavad UI elementide kogumit klassikalises lauaarvutistiilis kasutajaliideste loomiseks.
- QtNetwork

  Sisaldab klasse võrguga töötamiseks.
- QtSql

  Klassid andmebaasidega integreerimiseks kasutades SQL
- QtSvg

  Klassid SVG failide sisu kuvamiseks
- QtWebKit

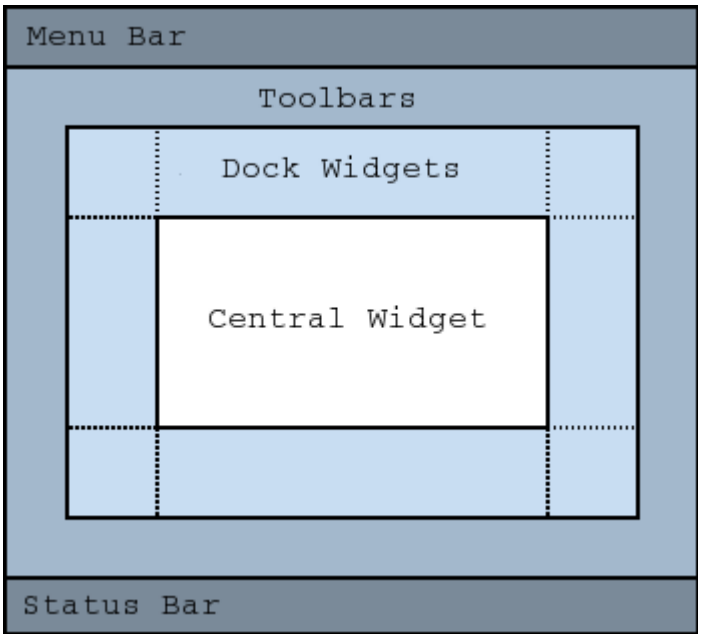
  Sisaldab klasse HTML-i näitamiseks ja toimetamiseks
- QtXml

  Klassid XML-iga töötamiseks

QApplication klass haldab GUI programmi põhilisi seadeid, töökäiku ja ressursse. See sisaldab programmi peatsüklit (ingl *main loop*), mille sees genereeritakse sündmused akna elementide poolt. See haldab samuti kogu süsteemi seadeid. Iga Qt abil tehtud rakenduse jaoks on olemas täpselt üks QApplication objekt, sõltumata akende arvust igal antud hetkel.

QWidget on klass, mis pärineb nii QObject kui ka QPaintDevice klassidest. See on alusklass kõikide kasutajaliideste objektide jaoks. Qwidget klassist pärinevad paljud laialt kasutatavad vidinate klassid, näiteks QDialog ja Qframe, millel on omakorda oma alamklasside süsteem.

QMainWindow ja temaga seotud klassid on peaakna haldamiseks. Tavaline GUI rakendus kasutab just QMainWindow objekti oma peaakna loomiseks. QMainWindow objektil on oma paigutus, millele võib lisada teisi vidinaid, näiteks QToolBars, QDockWidgets, QMenuBar ja QStatusBar. Aknal on keskala, millesse võib paigaldada ükskõik millise vidina (The Qt Company, 2016). Näidisakna illustratsioon on allpool (Joonis 2):



Joonis 2. QMainWindow (The Qt Company, 2016)

2.4. PyQt aken

Akna tekitamiseks PyQt raamistikus on olemas mitu erinevat võimalust. QWidget on alamklass kõikide teiste GUI objektide jaoks. QDialog põhineb QWidget klassil ja seda kasutatakse, kui on vaja näidata dialoogiakent (aken kus kasutaja peab andmeid sisestama). QMainWindows kasutatakse tavaliselt rakenduse peaakna loomiseks ja sellel on menüüriba, staatusriba ja tiitliriba tugi.

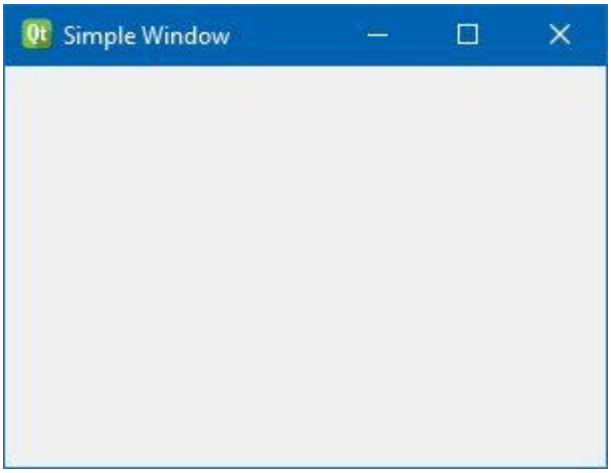
Alloleva koodi (Koodinäide 1) abil luuakse lihtne QWidget raamaken arvuti ekraanil:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon


class SimpleWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
```

```
    def initUI(self):
        self.setWindowTitle('Simple Window')
        self.setWindowIcon(QIcon('Qt_icon.png'))
        self.setGeometry(400, 400, 300, 200)
        self.show()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = SimpleWindow()
    sys.exit(app.exec_())
```

Koodinäide 1. Lihtsa raamakna loomise kood

Allpool on toodud koodi käivitamise tulemus (Joonis 3).



Joonis 3. PyQt lihtne aken

Ühe ja sama GUI rakenduse välimus võib erineda sõltuvalt operatsioonisüsteemist, mille peal ta käivitatakse, kuna tegelik vidinate joonistamine ekraanil tehakse OS-i graafilise süsteemi halduri poolt, mitte graafiliste teekide poolt.

Allpool on toodud koodi (Koodinäide 1) lahtiseletus.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon
```

Siin imporditakse vajalikud klassid. QApplication ja QWidget asuvad PyQt5.QtWidgets paketis. QIcon vidin, mida kasutatakse ikooni kuvamiseks, asub PyQt5.QtGui paketis.

```
class SimpleWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
```

Siin tekitatakse uus klass nimega SimpleWindow. See klass pärineb QWidget klassist (seepärast on ülemklassi nimi kirjutatud sulgudesse). See tähendab, et SimpleWindow objekti loomisel kutsutakse välja kaks konstruktorit: esimene konstruktor loodud SimpleWindow klassi jaoks ja teine ülemklassi jaoks. Meetod `super()` tagastab SimpleWindow klassi ülemobjekti (QWidget) ja me kutsume välja selle ülemobjekti konstruktori. Meetod `__init__()` on Python keele konstruktori meetod (pöörata tähelepanu topeltalakriipsudele).

```
self.initUI()
```

GUI loomine edastatakse `initUI()` meetodile.

```
def initUI(self):
    self.setWindowTitle('Simple Window')
    self.setWindowIcon(QIcon('Qt_icon.png'))
    self.setGeometry(400, 400, 300, 200)
    self.show()
```

Kõik siin kasutatud meetodid on päritud QWidget klassist. Meetodiga `setWindowTitle()` määrame akna tiitli, mida kuvatakse akna tiitelribal.

Meetod `setWindowIcon()` määrab programmi ikooni. Selleks loodi kõigepealt QIconi objekt. QIcon objekti argumendiks on tee ikooni faili asukohale (kuna antud juhul asub ikooni fail projekti kaustas, piisab faili nimest).

Meetod `setGeometry()` teeb kahte asja: see paigaldab akna ekraanile ja määrab akna suuruse. Kaks esimest parameetrit on akna x- ja y- asukohakoordinaadid ekraani ülemisest vasakust nurgast. Kolmas ja neljas parameeter on akna laius ja kõrgus.

Meetod `show()` kuvab vidina ja kõik selle lapsvidinad, kui sellised on olemas, ekraanile (esialgu tekitakse vidin mälus ja selle meetodiga kuvatakse reaalselt ekraanile).

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = SimpleWindow()
    sys.exit(app.exec_())
```

Iga PyQt5 programm peab looma programmi objekti, kus sys.argv parameeter on käsurea argumentide kogum. Pythoni skripte võib käivitada kestas (ingl *shell*). Nende argumentide abil võib kontrollida skripti käivitamist.

QApplication klassi objekt nimega **app** ja meie poolt kirjutatud SimpleWindow klassi objekt nimega **example** on loodud. Programmi peatsükkel on käivitatud.

```
sys.exit(app.exec_())
```

`exec()` on QApplication klassi staatiline meetod. Selle meetodi väljakutsumisega algatatakse programmi peatsükkel. Alates sellest punktist algab sündmuste töötlemine. Sündmuse saabumisel edastab peatsükkel selle sündmuse programmi vidinale. Peatsükkel lõpetatakse kui kutsutakse välja `exit()` meetod. Meetod `sys.exit()` tagab programmi korrektse lõpetamise. Süsteemi teavitatakse, kuidas programm lõppes.

2.5. PyQt sündmused ja signaalid

PyQt5 kasutab unikaalset „signaal ja pesa“ (ingl *signal and slot*) mehhanismi sündmuste käsitlemiseks. Signaale ja pesasid kasutatakse objektidevahelise kommunikatsiooni jaoks. Iga QObject (kaasaarvatud kõik PyQt vidinad, kuna nad pärinevad QWidget-ist, mis on omakorda QObjekti alamklass) toetab signaal ja pesa mehhanismi. Signaal väljastatakse kui toimub konkreetne sündmus. Pesaks võib olla ükskõik milline Pythoni meetod või funktsioon. Pesa kutsutakse välja kui sellele ühendatud signaal väljastatakse. Enamikul vidinatel on eelnevalt defineeritud pesad ja võib lihtsalt ühendata eelnevalt defineeritud signaali eelnevalt defineeritud pesale, nii et ei pea muretsema, kuidas soovitud käitumist saada (Bodnar, 2016).

Saatja on objekt, mis väljastab signaali. Vastuvõtja on objekt, mis võtab vastu signaali. Pesa on meetod, mis reageerib sündmusele.

Järgnev näide (Koodinäide 2) illustreerib sellist mehhanismi.

```
import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QLCDNumber,
QSlider, QVBoxLayout
from PyQt5.QtGui import QIcon

class Example(QWidget):
```

```
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('PyQt Example')
        self.setWindowIcon(QIcon('Qt_icon.png'))
        self.setGeometry(400, 400, 300, 200)

        lcd = QLCDNumber(self)
        slider = QSlider(Qt.Horizontal, self)

        vbox = QVBoxLayout()
        vbox.addWidget(lcd)
        vbox.addWidget(slider)

        self.setLayout(vbox)

        slider.valueChanged.connect(lcd.display)

        self.show()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = Example()
    sys.exit(app.exec_())
```

Koodinäide 2. LCD näidiku ja kerimisribaga aken

Koodi (Koodinäide 2) selgitus.

```
lcd = QLCDNumber(self)
slider = QSlider(Qt.Horizontal, self)
```

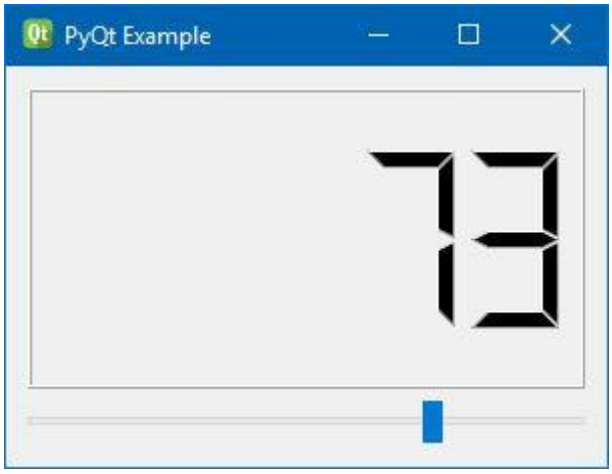
Siin luuakse QLCDNumber vidin, mis kuvab arve LCD stiilis. Luuakse ka QSlider vidin, mis kuvab horisontaalse või vertikaalse kerimisriba. Kui parameetreid pole kaasa antud, luuakse kerimisriba vaikimisi vahemikuga 0..99 ja sammuga 1.

```
slider.valueChanged.connect(lcd.display)
```

Siin ühendatakse signaal `valueChanged`, mille väljastab kerimisrea objekt `slider` (saatja), objekti `lcd` (vastuvõtja) pesaga `display`.

Teiste sõnadega, kerimisriba `slider` liigutamisel edastatakse signaal `valueChanged` LCD väljale `lcd`, millel on olemas `display` pesa, kus numbreid kuvatakse.

Allpool on koodikäivitamise tulemus (Joonis 4). Liigutades kerimisriba muutub kuvatav number.



Joonis 4. Aken kerimisriba ja LCD väljaga

2.6. PyQt nupud ja tekstiväljad

Qt raamistik pakub klasse erinevat tüüpi nuppe loomiseks - QRadioButton, QCheckBox, QtoolButton ja QPushButton. Enim kasutatav nendest on QPushButton

Allpool toodud kood (Koodinäide 3) tekitab akna, milles on kaks nuppu ja tekstiväli. Tekstivälja tekst muutub sõltuvalt sellele, millisele nupule vajutati.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget,
QPushButton, QLabel
from PyQt5.QtGui import QIcon

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('PyQt Example')
        self.setWindowIcon(QIcon('Qt_icon.png'))
        self.setGeometry(400, 400, 300, 200)

        self.label = QLabel(self)
        self.label.move(100, 20)

        button1 = QPushButton('Button 1', self)
        button1.move(50, 60)

        button2 = QPushButton('Button 2', self)
        button2.move(170, 60)
```

```
        button1.clicked.connect(self.buttonClicked)
        button2.clicked.connect(self.buttonClicked)

        self.show()

    def buttonClicked(self):
        sender = self.sender()
        self.label.setText(sender.text() + ' was pressed')
        self.label.adjustSize()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = Example()
    sys.exit(app.exec_())
```

Koodinäide 3. Nuppude ja tekstiväljaga akna kood

Koodi (Koodinäide 3) selgitus.

```
from PyQt5.QtWidgets import QApplication, QWidget,
QPushButton, QLabel
```

Nuppude ja tekstivälja kasutamiseks loodud PyQt5 programmis uuendatakse `import` rida ja imporditakse QPushButton ja QLabel vidinad.

```
self.label = QLabel(self)
self.label.move(100, 20)
```

Teksti kuvamiseks luuakse QLable vidin ja paigutatakse see soovitavale asukohale aknas.

```
button1 = QPushButton('Button 1', self)
button1.move(50, 60)

button2 = QPushButton('Button 2', self)
button2.move(170, 60)
```

Luuakse kaks nuppu ja paigutatakse need soovitavale asukohale.

```
button1.clicked.connect(self.buttonClicked)
button2.clicked.connect(self.buttonClicked)
```

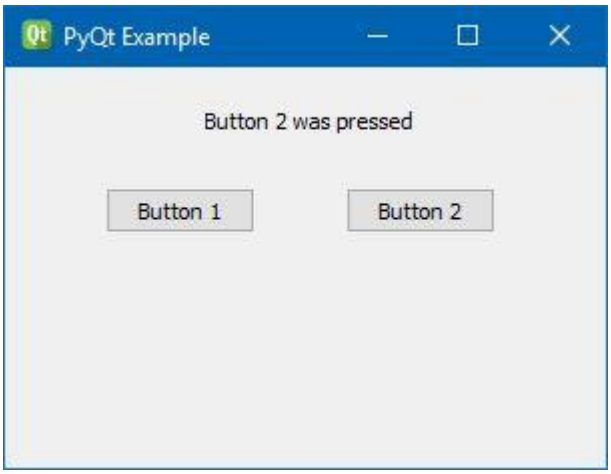
Ühendatakse mõlemad nupud sama pesaga (meetodiga, mis reageerib signaalile) ja sellele meetodile antakse nimeks `buttonClicked`.

```
def buttonClicked(self):
    sender = self.sender()
    self.label.setText(sender.text() + ' was pressed')
    self.label.adjustSize()
```

Defineeritakse `buttonClicked()` meetod. Selles meetodis tuvastatakse, millist nuppu oli vajutatud, kutsudes välja `sender()` meetod. `sender()` on QObject klassi meetod, mis tagastab viite objektile, mis saatis signaali.

QLabel klassi meetodiga `setText()` kuvatakse tekstiväljal vajutatud nupu nimi. Tekstivälja suuruse kohandamiseks vastavalt teksti suurusele kutsutakse välja `adjustSize()` meetod.

Koodikäivitamise tulemus (Joonis 5):



Joonis 5. Aken kahe nupu ja ühe tekstiväljaga

2.7. PyQt sisestusdialoog

Peaaegu igal graafilise kasutajaliidesega programmil on vähemalt üks dialoogiaken ja enamustel GUI programmidel on üks peaaken paljude dialoogidega. Dialooge kasutatakse andmete sisestamiseks, andmete muutmiseks, programmi seadete muutmiseks jne.

PyQt API sisaldab teatud hulka eelseadistatud Dialog vidinaid, näiteks InputDialog, FileDialog, ColorDialog jne.

Edasi vaadeldakse QInputDialog klassi, mis pakub lihtsa ja mugava dialoogi, millega kasutaja saab sisestada ühte tüüpi andmeid.

Sisestatavad andmed võivad olla tekstiandmed, arvandmed või element nimekirjast. Kasutajale tuleb teatada tekstiväljaga, millist tüüpi andmeid peab sisestama.

QInputDialog klass pakub viit staatilist funktsiooni: getText(), getMultiLineText(), getInt(), getDouble() ja getItem(). Kõiki neid funktsioone kasutatakse sarnasel viisil (The Qt Company, 2016).

Järgnevas koodis (Koodinäide 4) kasutatakse klassi QInputDialog mingisuguse teksti ja täisarvu saamiseks kasutajalt:

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget,
QPushButton, QInputDialog, QLabel
from PyQt5.QtGui import QIcon

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('PyQt Example')
        self.setWindowIcon(QIcon('Qt_icon.png'))
        self.setGeometry(400, 400, 300, 200)

        self.button1 = QPushButton('Enter name', self)
        self.button1.move(20, 40)
        self.button1.clicked.connect(self.showDialogName)

        self.labelName = QLabel(self)
        self.labelName.move(130, 45)
        self.labelName.setText("Your name")

        self.button2 = QPushButton('Enter age', self)
        self.button2.move(20, 80)
        self.button2.clicked.connect(self.showDialogAge)

        self.labelAge = QLabel(self)
        self.labelAge.move(130, 85)
        self.labelAge.setText("Your age")

        self.show()

    def showDialogName(self):
        text, okPressed = QInputDialog.getText(self, "Get
text", 'Enter your name:')
        if okPressed:
            self.labelName.setText(str(text))
            self.labelName.adjustSize()

    def showDialogAge(self):
        intNumber, okPressed = QInputDialog.getInt(self, "Get
integer", 'Enter your age:', min=0, max=150)
        if okPressed:
```

```
            self.labelAge.setText(str(intNumber))

if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = Example()
    sys.exit(app.exec_())
```

Koodinäide 4. Lihtsa sisestusdialoogi akna kood

Selles näites (Koodinäide 4) luuakse kahe nupu ja tekstiväljaga aken. Ühele nupule vajutamisega avaneb dialoog teksti sisestamiseks, teisele nupule vajutamisega avaneb dialoog täisarvu sisestamiseks. Sisestatud andmeid kuvatakse vastavatel tekstiväljadel.

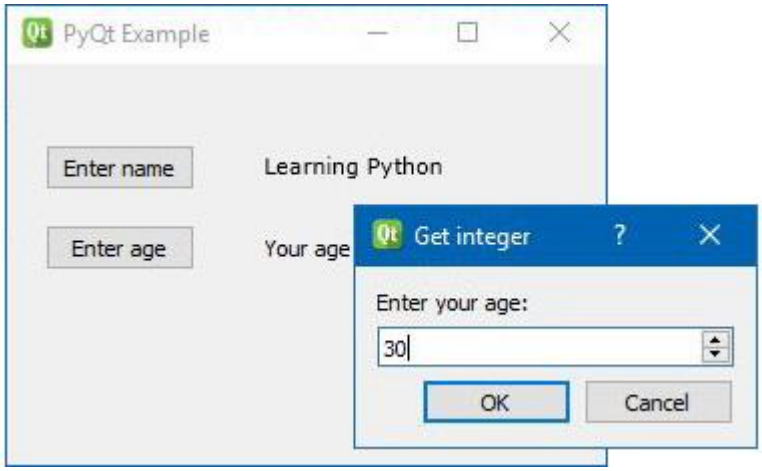
```
text, okPressed = QInputDialog.getText(self, "Get text",
'Enter your name:')
```

See rida kuvab dialoogi teksti sisestamiseks. Esimene stringi-tüüpi argument on dialoogiakna tiitel, teine on tekst, mida kuvatakse dialoogi aknas. Dialoog tagastab sisestatud teksti ja tõeväärtuse. Tõeväärtuseks on **True**, kui vajutakse OK nupule.

```
intNumber, okPressed = QInputDialog.getInt(self, "Get
integer", 'Enter your age:', min=0, max=150)
```

See rida kuvab dialoogi täisarvu sisestamiseks. Parameetrid antakse järgmises järjekorras: self, aknatiitel, tekstisõnum dialoogi akna sees, vaikeväärtus, miinimumväärtus, maksimumväärtus ja sammu suurus (nendest esimesed kolm on kohustuslikud, viimased neli parameetrit on valikulised). Tagastatakse täisarv (integer) ja tõeväärtus, mis on **True** kui vajutati OK nuppu.

Koodi käivitamise tulemus (Joonis 6):



Joonis 6. Lihtne dialoogiaken

3. Qt Designer – tööriist GUI loomiseks

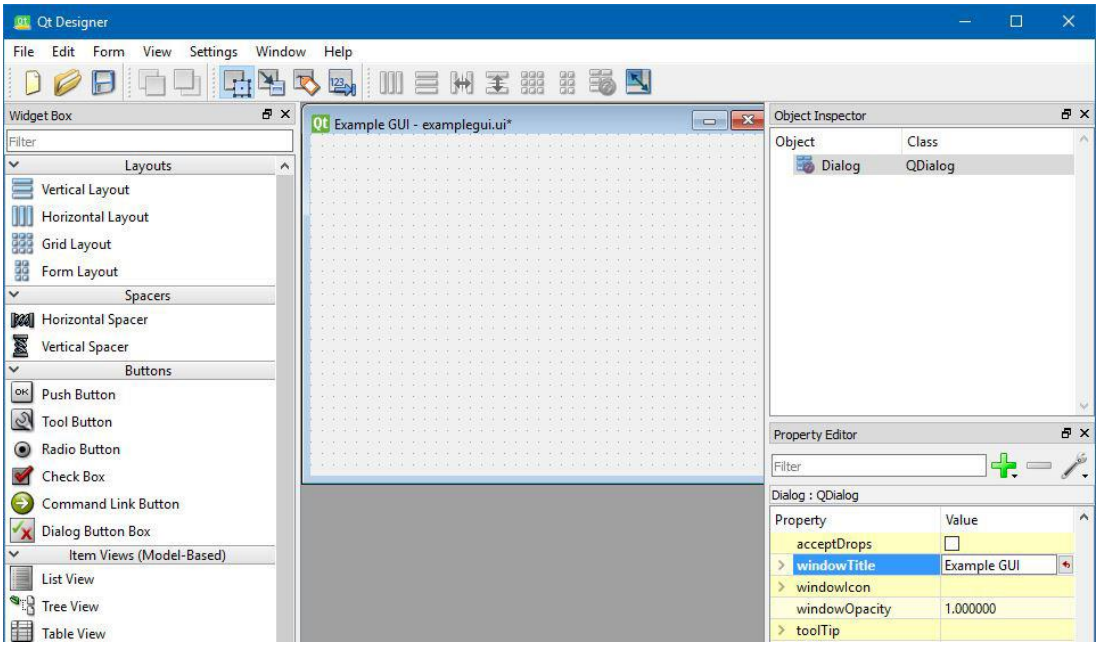
Graafilise kasutajaliidese rakenduse loomine kirjutades programmeerimiskoodi käsitsi võib osutuda igavaks, keeruliseks ja töömahukaks tegevuseks. Isegi lihtsa dialoogiakna programmeerimiseks võib tarvis olla kirjutada sadu koodiridu. Õnneks pakub PyQt võimaluse kasutada Qt Designer-it – tööriista Qt komponentidest graafiliste kasutajaliideste loomiseks. Qt Designer võimaldab luua GUI visuaalselt, kasutades lihtsat pukseerimist (*drag and drop*). Valminud tulemus salvestatakse *.ui failina, mis tegelikult kujutab endast XML faili, milles kirjeldatakse kõiki kasutaja poolt moodustatud GUI komponente ja nende omadusi. Tekitatud faili võib konverteerida Pythoni klassiks, mis on valmis Pythoni rakendusse importimiseks. Qt Designer-it võib kasutada dialoogide, enda tehtud vidinate ja peaakna kasutajaliideste loomiseks (Summerfield, 2008).

Kõigepealt tuleb Qt Designer-i alla laadida ja paigaldada. Qt Designer on osa Qt installeerimise paketist ja selle võib alla laadida järgmiselt aadressilt: https://www.qt.io/download/

Selles juhendis kasutatakse Qt versiooni 5.7.

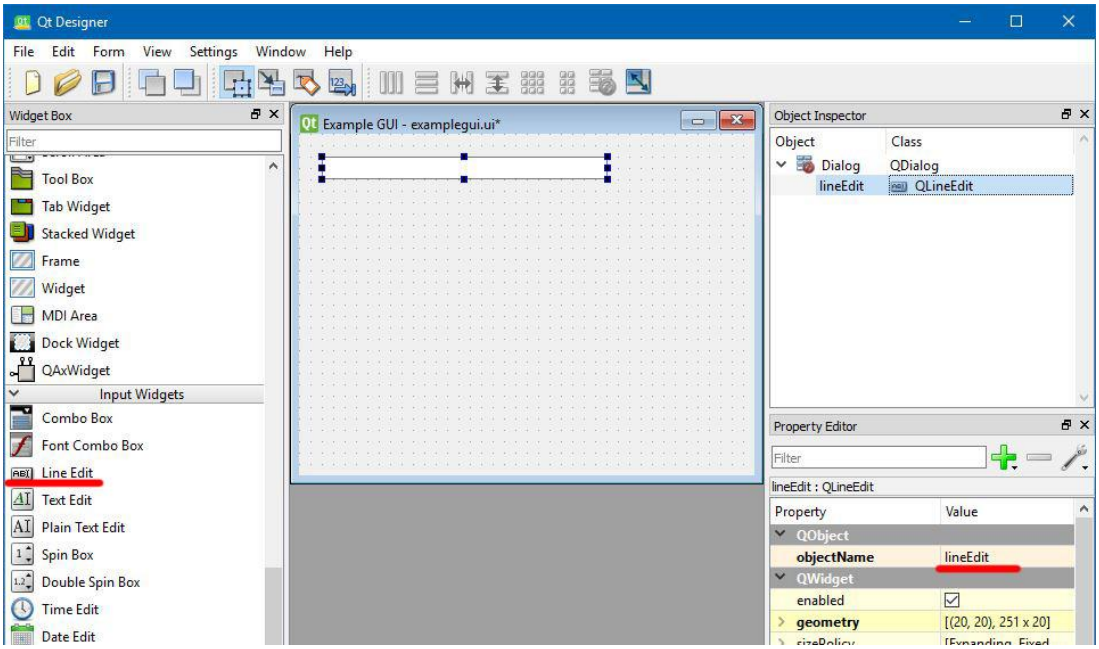
Kui paigaldamine on lõpule viidud, käivitatakse programm Qt Designer. Kui kasutatakse Microsoft Windows-it, tekib start menüüsse programmi ikoon. Juhul kui seda sealt ei leia, võib proovida otsida Qt kaustast. Minu arvutis näiteks asub programm järgmisel aadressil: „C:\Qt\5.7\msvc2015_64\bin\designer.exe“.

Järgnevalt luuakse lihtne dialoogiaken tekstisisestusväljaga, kus kasutaja saab andmeid nimekirjale lisada ja kustutada. Selleks valida **File > New**, siis **Dialog without buttons** ja vajutada **Create**. Ekraanile tekib tühi dialoogiaken (Joonis 7):



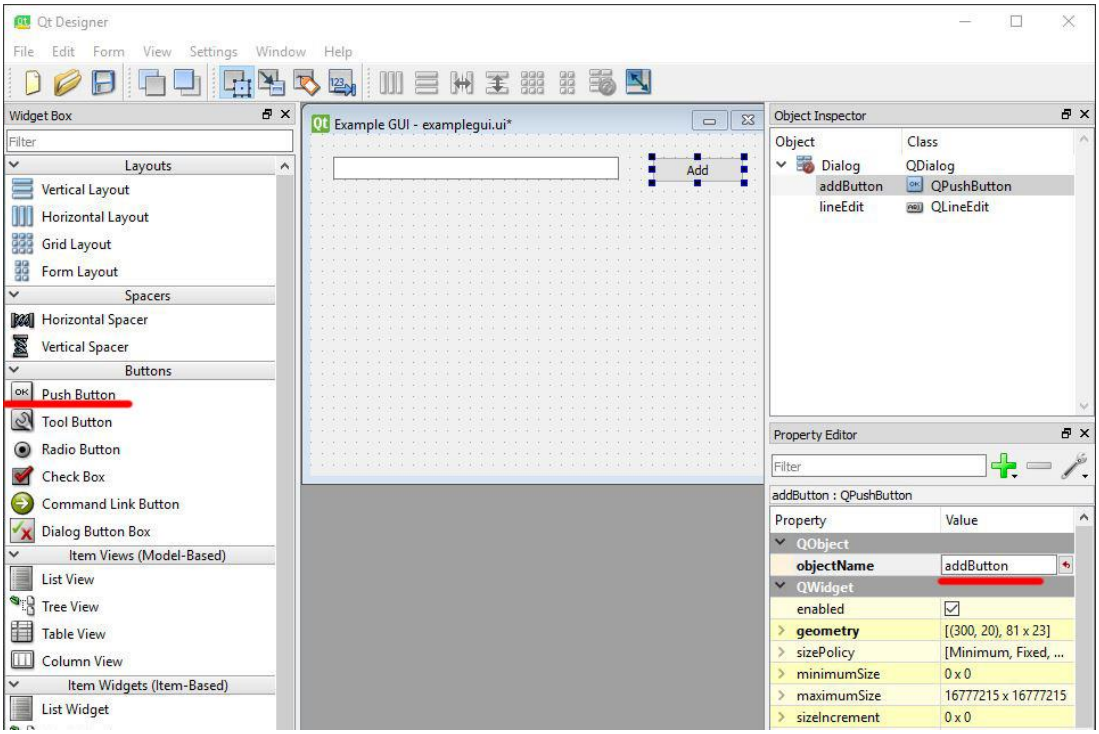
Joonis 7. Dialoogiakna loomine Qt Designer-i abil

Edasi, tõmmata vasakust tööriistapaneelist **Line Edit** vidin tekkinud dialoogiakna ülemisse vasakusse nurka ja muuta selle vidina suurus vastavalt soovile. Suurust saab muuta vidina serva hiirega tõmmates. Tekkinud objekti nimeks panna „lineEdit“. Seda saab teha **Property Editor** paneelil, mis asub paremal küljel (Joonis 8).



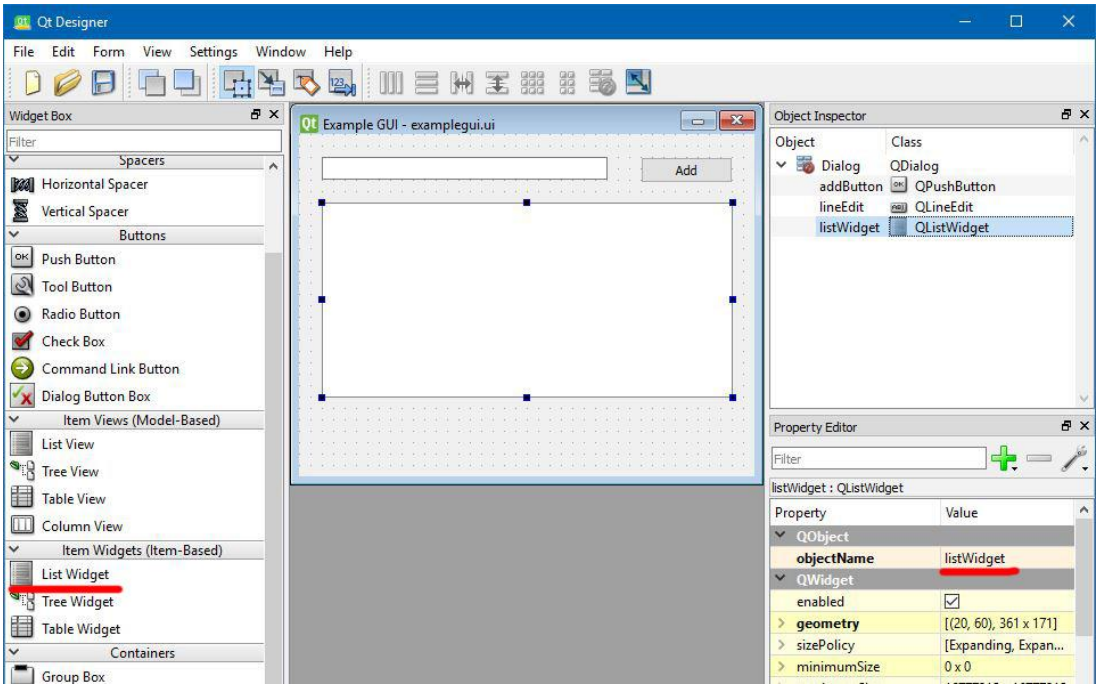
Joonis 8. Teksti sisestusvälja lisamine

Järgmiseks tõmmata vasakust tööriistapaneelilt vidin **Push Button** eelmisel sammul tekitatud sisestamisreast paremale. Sellele panna nimeks „addButton“ ja kirjutada selle tekstiväljale „Add“ (Joonis 9).



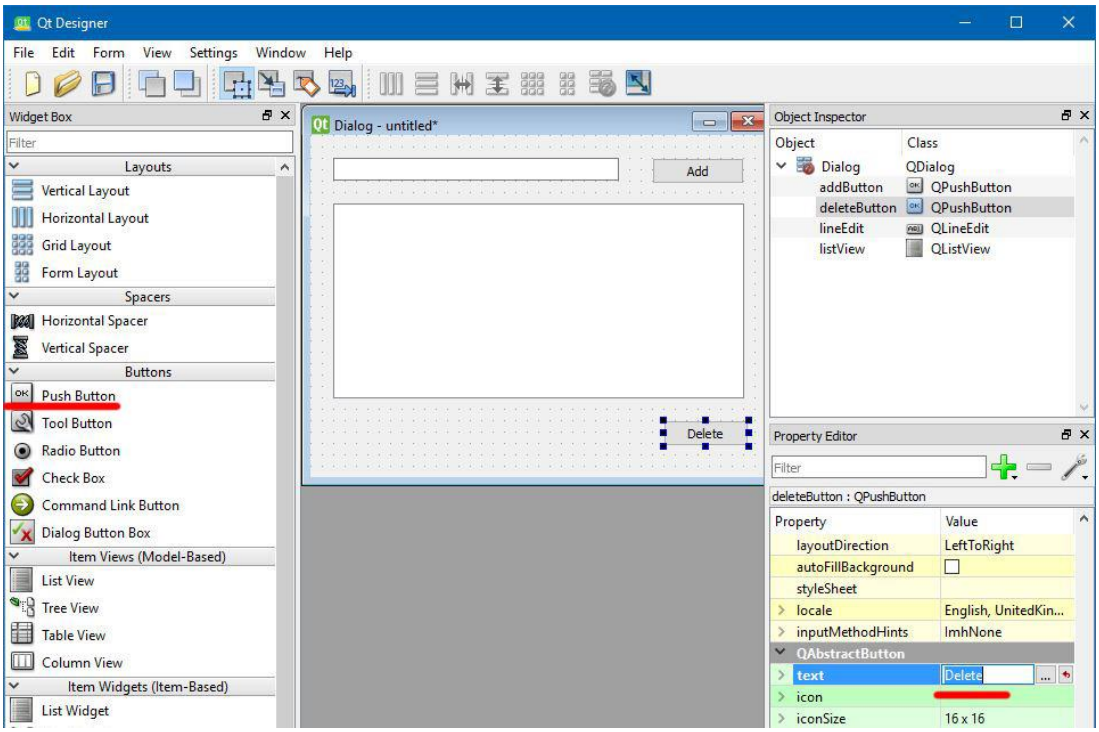
Joonis 9. Nupu lisamine

Lisada vidin **List Widget** akna keskele ja muuta selle suurust, nii et ta näeks atraktiivne välja, ja nimetada see „listWidget“ (Joonis 10).



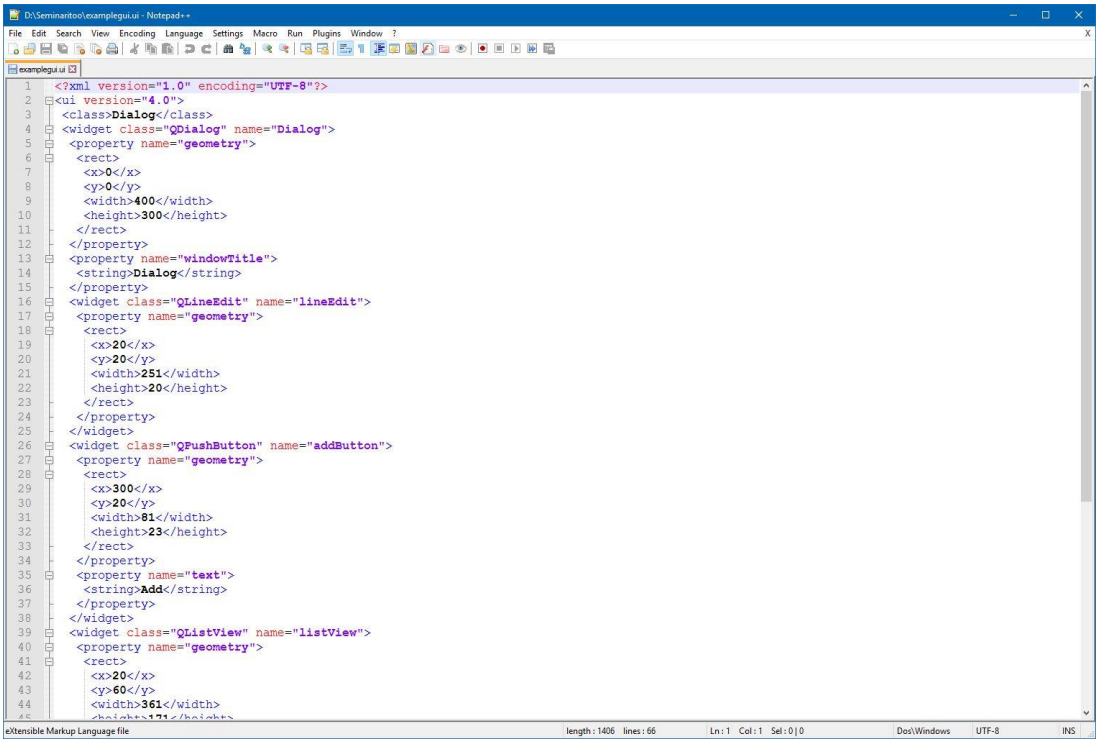
Joonis 10. ListWidget vidina lisamine

Seejärel lisatakse viimane nupp. Selleks tõmmata vidin **Push Button** dialoogiakna alumisse äärde, panna objekti nimeks „deleteButton“ ja nupu tekstiväljale kirjutada „Delete“ (Joonis 11).



Joonis 11. "Delete" nupu lisamine

Nüüd fail salvestatakse, näiteks nimega „examplegui.ui“. Failis on XML-vormingus kirjeldatud kõik loodud kasutajaliidese elemendid ja nende omadused. Tulemus peab välja nägema sarnaselt alltoodud joonisele (Joonis 12). Faili täielikku sisu saab vaadata lisas (Lisa 1).

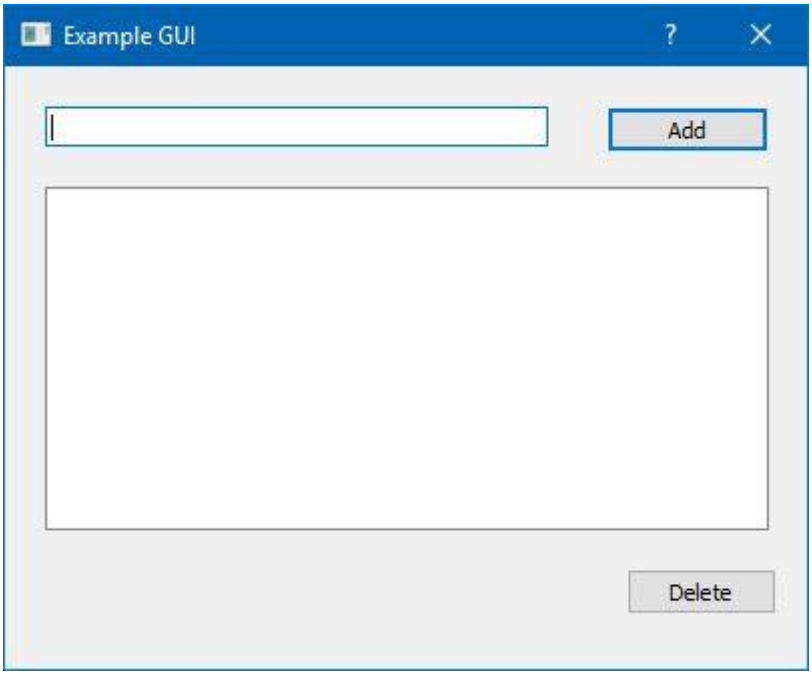


Joonis 12. *.ui faili sisu näide

Järgmise sammuna tuleb examplegui.ui fail konverteerida Python koodiks. Käsureaprogrammis või terminalis tuleb muuta aktiivseks see kataloog, kuhu fail salvestati, ja sisestada järgmine käsk (muuta faili nimi kui see salvestati teise nimega):

```
> pyuic5 -x examplegui.ui -o examplegui.py
```

Selle käsuga luuakse Pythoni fail (Lisa 2) ja kui nüüd see fail käivitada, peab avanema aken, mis peaks välja nägema sarnaselt allolevaga (Joonis 13):



Joonis 13. Aken, mis tekib kasutajaliidese kujundusega Pythoni faili käivitamisel

Hetkel on loodud vaid puhas kasutajaliides ilma mingi funktsionaalsuseta. Selleks, et see programm teeks midagi kasulikku, tuleb lisada sellele loogikat.

Selle eesmärgi saavutamiseks luuakse veel üks Pythoni fail ja nimetatakse see näiteks nimega „exampleprog.py“.

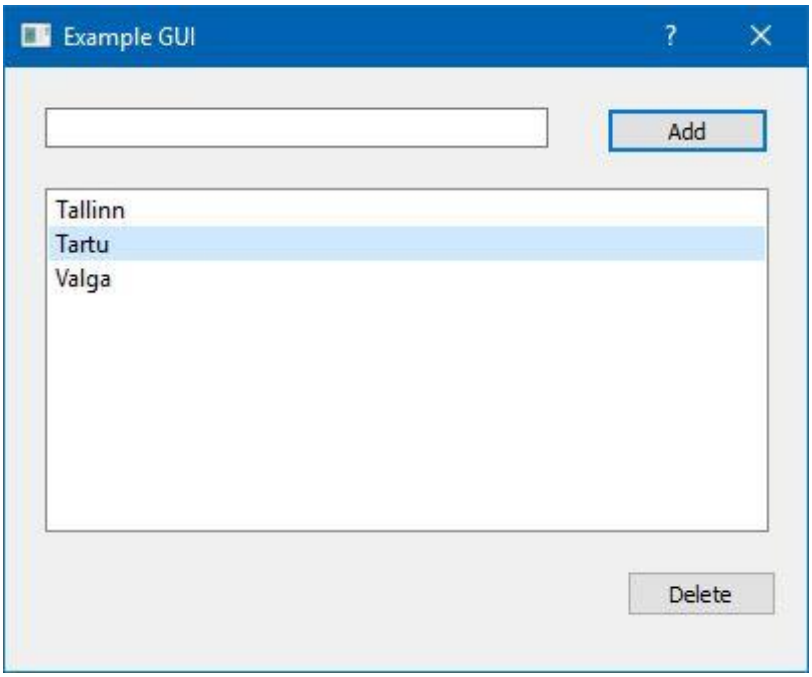
Uuele loodud failile lisatakse järgmine kood (Koodinäide 5):

```
import sys
from PyQt5 import QtWidgets
from examplegui import Ui_Dialog

class ExampleGuiProgram(QtWidgets.QDialog, Ui_Dialog):
    def __init__(self):
        QtWidgets.QDialog.__init__(self)
        Ui_Dialog.__init__(self)
        self.setupUi(self)

        self.addButton.clicked.connect(self.addButtonClicked)
        self.deleteButton.clicked.connect(self.deleteButtonClicked)
        self.deleteButton.setEnabled(False)

    def addButtonClicked(self):
        text = self.lineEdit.text()
        if len(text):
            self.listWidget.addItem(text)
            self.lineEdit.clear()
            self.deleteButton.setEnabled(True)

    def deleteButtonClicked(self):
        self.listWidget.takeItem(self.listWidget.currentRow())
```

```
        if self.listWidget.count() == 0:
            self.deleteButton.setEnabled(False)

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = ExampleGuiProgram()
    window.show()
    sys.exit(app.exec_())
```

Koodinäide 5. Dialoogiakna rakenduse kood

Koodi (Koodinäide 5) käivitamise tulemus peab nägema välja sarnaselt alltoodule (Joonis 14):



Joonis 14. Valmis dialoogiaken sisestusvormiga.

Programmeerimiskoodi (Koodinäide 5) lahtiseletus.

```
from examplegui import Ui_Dialog
```

Selle lausega imporditakse Qt Designer-i abil genereeritud kasutajaliidese moodul.

```
class ExampleGuiProgram(QtWidgets.QDialog, Ui_Dialog):
    def __init__(self):
        QtWidgets.QDialog.__init__(self)
        Ui_Dialog.__init__(self)
        self.setupUi(self)
```

Luuakse uus klass nimega ExampleGuiProgram, mis pärineb klassidest QDialoog ja Ui_Dialog. See klass on põhimõtteliselt kahe klassi ühend ja seetõttu võib pöörduda mõlema klassi atribuutide poole otse, kasutades selleks prefiksit **self.**

Seejärel initsialiseeritakse ülemklasside konstruktorid.

Järgmiseks kutsutakse välja meetod `setupUi()`. See on eelnevalt Qt Designer-i abiga genereeritud mooduli meetod. See meetod loob kõik kasutajaliidese failis olevad vidinad ja paigutab neid vastavalt failis kirjeldatud kujundusele. Teiste sõnadega – see meetod taasloob Qt Designer-i abiga tehtud dialoogiakna.

```
self.addButton.clicked.connect(self.addButtonClicked)
self.deleteButton.clicked.connect(self.deleteButtonClicked)
```

Siin ühendatakse nupud neile vastavatesse pesadesse (meeldetuletus: pesad on meetodid, mis reageerivad sündmustele).

```
self.deleteButton.setEnabled(False)
```

Kuna loodud nimekiri on hetkel veel tühi, deaktiveeritakse „Delete“ nupp.

```
def addButtonClicked(self):
    text = self.lineEdit.text()
    if len(text):
        self.listWidget.addItem(text)
        self.lineEdit.clear()
        self.deleteButton.setEnabled(True)
```

Siin defineeritakse meetod, mis ühendati nupuga „Add“. Kui nuppu vajutatakse, saadakse tekst vidinast „lineEdit“ kätte. Kontrollitakse, kas tekstiväli sisaldas teksti ja kui jah, siis lisatakse tekst vidinale „listWidget“, misjärel tehakse teksti sisestusväli tühjaks ja aktiveeritakse nupp „Delete“.

```
def deleteButtonClicked(self):
    self.listWidget.takeItem(self.listWidget.currentRow())
    if self.listWidget.count() == 0:
        self.deleteButton.setEnabled(False)
```

„Delete“ nupuga ühendatud meetodi defineerimine. Kui vajutatakse „Delete“ nuppu, siis eemaldatakse valitud element vidinast „listWidget“, kasutades selleks meetodit `takeItem()`, mis võtab argumendiks reanumbri, mis on omakorda saadud kätte kasutades meetodit `currentRow()` (mõlemad meetodid on klassi „QlistWidget“ public meetodid). Kontrollitakse, kas listWidget sai tühjaks peale rea kustutamist, ja kui on tühi, siis deaktiveeritakse uuesti „Delete“ nupp.

Sellega lõppeb tutvumine Qt Designer-iga. Siin kasutati kahte erinevat graafilise kasutajaliidese loomise lähenemisviisi: 1) käsitsi programmeerimiskoodi kirjutades ja 2) kasutades visuaalset töövahendit, nagu Qt Designer. Erinevate lahenduste võrdlusena saab öelda, et redaktori kasutamine on oluliselt mugavam ja säästab palju aega. Arendaja saab kontsentreeruda rakenduse loogikale ja arendusprotsess muutub efektiivsemaks.

Kokkuvõte

Käesoleva seminaritöö eesmärgiks oli tutvustada võimalusi, mida pakub Qt raamistik Python keeles rakenduste loomisel ja luua lühike õpetus sellel teemal.

Eesmärgi saavutamiseks kirjutas autor teoreetilisest taustast, sealhulgas lühidalt GUI olemusest ja selle töötamise põhimõttest, tutvustas võimalusi GUI arendamiseks Python keeles ja Qt raamistikku. Loodi lühike juhend, kus autor tutvustas PyQt seoste kogumit, mille abil saab luua Qt raamistiku võimalusi kasutavaid rakendusi Python keeles. Õpetuses antakse mõnede graafiliste komponentide rakendamise näited koos koodiseletustega ja illustratsioonidega. Töö lõpuosas kirjutas autor Qt Designer-ist – tööriistast, mille abil saab kergesti luua GUI elementidega rakendusi kasutades Qt raamistiku võimalusi.

Tagasiside saamiseks jagas autor antud seminaritööd tudengitega, kes olid juba varem Python keelega tutvunud. Tudengid tutvusid siinses töös olevate juhenditega ja proovisid toodud näiteid läbi teha enda arvutitel. Tagasiside ja kommentaarid anti autorile vabas vormis. Saadud tagasiside põhjal on tehtud mõned muudatused antud töös – muudeti töö struktuuri, parandati kirjastiili ja kirjavigu. Juhendi kohta negatiivseid kommentaare ei olnud ja süsteemi seadistusega ning näidete läbiproovimisega probleeme ei tekkinud.

Autori hinnangul on seminaritööks püstitatud eesmärgid saavutatud.

Enne selle töö alustamist puudusid autoril piisavad teadmised, kuidas luua graafilise kasutajaliidesega rakendusi Python keeles. Töö kirjutamise käigus sai autor uusi teadmisi Qt raamistikust ja selle võimalustest. Huvitav oli teada saada, et GUI rakenduste loomine Python keeles ei ole nii raske kui esmapilgul tundus.

Tuleb mainida, et autor sai teada ka kui raske on leida eestikeelseid vasteid ingliskeelsetele IT-alastele terminitele. Paljudele sõnadele täpsemaid ja kinnitatud eestikeelseid vasteid lihtsalt ei eksisteeri.

Kasutatud kirjandus

Bodnar, J. (2016). *PyQt5 tutorial*. Loetud aadressil http://zetcode.com/gui/pyqt5/

Jaganmohan, G., & Loganathan, V. (2016). *PySide GUI Application.* Birmingham: Packt Publishing.

Riverbank Computing Ltd. (2016). *Riverbank Computing*. Loetud aadressil https://www.riverbankcomputing.com/

Shantnu. (2015). *Your first GUI app with Python and PyQt*. Loetud aadressil http://pythonforengineers.com/your-first-gui-app-with-python-and-pyqt/

Summerfield, M. (2008). *Rapid GUI Programming with Python and Qt.* Michigan: Prentice Hall.

The Qt Company. (2016). *Qt Documentation*. Loetud aadressil http://doc.qt.io/

TutorialsPoint. (n.d.). *Learn PYQT*. Loetud aadressil https://www.tutorialspoint.com/pyqt/index.htm

Цилюрик, О. (2014). *Тонкости использования языка Python: Часть 9. Разработка GUI-приложений.* Loetud aadressil https://www.ibm.com/developerworks/ru/library/l-python_details_09/

LISAD

Lisa 1. examplegui.ui faili sisu

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
  <class>Dialog</class>
  <widget class="QDialog" name="Dialog">
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>400</width>
        <height>300</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>Example GUI</string>
    </property>
    <widget class="QLineEdit" name="lineEdit">
      <property name="geometry">
        <rect>
          <x>20</x>
          <y>20</y>
          <width>251</width>
          <height>20</height>
        </rect>
      </property>
    </widget>
    <widget class="QPushButton" name="addButton">
      <property name="geometry">
        <rect>
          <x>300</x>
          <y>20</y>
          <width>81</width>
          <height>23</height>
        </rect>
      </property>
      <property name="text">
        <string>Add</string>
      </property>
    </widget>
    <widget class="QPushButton" name="deleteButton">
      <property name="geometry">
        <rect>
          <x>310</x>
          <y>250</y>
          <width>75</width>
          <height>23</height>
        </rect>
      </property>
      <property name="text">
        <string>Delete</string>
      </property>
    </widget>
    <widget class="QListWidget" name="listWidget">
      <property name="geometry">
```

```
        <rect>
          <x>20</x>
          <y>60</y>
          <width>361</width>
          <height>171</height>
        </rect>
      </property>
    </widget>
  </widget>
  <resources/>
  <connections/>
</ui>
```

Lisa 2. examplegui.py faili sisu

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

# Form implementation generated from reading ui file
'examplegui.ui'
#
# Created by: PyQt5 UI code generator 5.7

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(400, 300)
        self.lineEdit = QtWidgets.QLineEdit(Dialog)
        self.lineEdit.setGeometry(QtCore.QRect(20, 20, 251,
20))
        self.lineEdit.setObjectName("lineEdit")
        self.addButton = QtWidgets.QPushButton(Dialog)
        self.addButton.setGeometry(QtCore.QRect(300, 20, 81,
23))
        self.addButton.setObjectName("addButton")
        self.deleteButton = QtWidgets.QPushButton(Dialog)
        self.deleteButton.setGeometry(QtCore.QRect(310, 250,
75, 23))
        self.deleteButton.setObjectName("deleteButton")
        self.listWidget = QtWidgets.QListWidget(Dialog)
        self.listWidget.setGeometry(QtCore.QRect(20, 60, 361,
171))
        self.listWidget.setObjectName("listWidget")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Example
GUI"))
        self.addButton.setText(_translate("Dialog", "Add"))
        self.deleteButton.setText(_translate("Dialog",
"Delete"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```