Təhsilə dair axtardığınız bir çox kitabın elektron versiyasını “**Telegram**” kanalımızda tapa bilərsiniz (https://t.me/eSources).

Telegram:
Kanal: @eSources
Reklam, təklif və iradlarınız üçün: @n4hkro

- Kitablar ödənişlidir?

✓ Xeyr, təbii ki.

- Paylaşdığınız kitabları öz kanalımda paylaşa bilərəm?

✓ Bəli. Könül istərdi ki, paylaşarkən mənbə bildirəsiniz, amma təbii ki, heç kim sizi buna məcbur etmir.

- Bədii kitablar da paylaşırsınız?

✓ Xeyr, amma həmin kitab sizə dərs üçün lazımdırsa, istisna edərik.

- Azərbaycan Milli Kitabxanasından kitab yükləyirsiniz?

✓ Bəli. Onlayn şəkildə oxunulması mümkün olan istədiyiniz kitabı yükləyirik.

- Sizə kitab göndərsəm qarşılığında nə alacağam?

✓ Kanaldan maddi qazancımız olmadığı üçün, bizə göndərdiyiniz kitablara görə ən yaxşı halda sizin kanalınızı reklam edə bilərik.

Python + Qt = PyQt

PyQt(GUI)
Buraxılış ili – 02.01.2017
Müəllif - Rəşad Qarayev

əlaqə üçün - pythonaz@yahoo.com
- garayevrashad@hotmail.com
Twitter - http://www.twitter.com/rashad_garayev

Platform - Ubuntu 14.04
PyQt version – PyQt4(GUİ- grapfical user interface)
Python (version 2.7)

Mündəricat (Table of Contents)

Giriş

Kitabın mövzuları PyQt kitabxanasının sadə dildə,ana dilimizdə izahından bəhs edilir.Mövzulardan əlavə çeşidli qrafik proqramlarla siz PyQt -i daha yaxşı mənimsəyəcəksiniz.bundan əlavə olaraq PyQt ilə yanaşı və birgə istifadə üçün html və css haqqında nəzəri biliklәriniz olmalıdır.Kitabda daha çox css -atributlarından istifadə olunacaq.Modern qrafik proqramlamanın demək olarki əsasını təşkil edən Qt -(c,c++) ,SİP proqramlama dili ilə binding edilərək ortaq nəticəsi PyQt bizə daha görünüşlü Gui,eləcədə zəngin kitabxanası ilə rahatlıqla ideyalarınızı tətbiq etmək imkanı yaradır.Kitab tamamilə pulsuzdur,istifadəsi hər kəs üçün nəzərdə tutulub.
Kitaba başlamaq üçün,python proqramlama dili- nəzəri bilikləri əldə edin.Kitabı diqqətlə oxuyub kodları tətbiq edin.Çalışın əllə yazmağa üstünlük verin.Metod və funksiyalara dəstək verən editor proqramlardan(Pycharm,Eric,Wing ide,Eclipse + PyDev və s) istifadə etməyin.

Qt və PyQt haqqında

Qt Gui (*grapfical user interface)* ilk dəfə olaraq Trolltech (Norveç) (http://www.troltech.com) şirkəti tərəfindən istifadəyə başlanılmışdır.LGPL ilə lisenziyalaşdırılan Qt Digia daha sonra isə Nokia şirkəti tərəfindən istifadə edilmişdir.Hal-hazırda bir çox şirkətlər Qt Gui(designer) istifadə etməkdə davam edir.Misal olaraq Google,Adobe, Siemens,Skype,Kde Nokia series60(Qt) və s.Demək olarki modern qrafik interfeys dizaynı üçün istifadəyə yararlıdır.Çox geniş kitabxanaya malik Qt c və c++ dilləri ilə yazılmışdır.Kitabxanalarının genişlədilməsi isə SİP (https://riverbankcomputing.com/software/sip/intro) üzərində həyata keçirilir.SİP PyQt üçün nəzərdə tutulmuş binding dildir.Yəni SİP - PyQt üçün kitabxana bazası yaradır.Bu fayllar bazada ui və xml faylları altında saxlanılır.Daha sonra convert(çevirmək) edilərək py fayllarına çevirir. (pyuic4 input.ui -o output.py)SİP -i yükləmək üçün aşağıdakı ünvandan istifadə edə bilərsiniz.
https://riverbankcomputing.com/software/sip/download

PyQt lisenziyalıdır.İstifadəsi deyil,yazdığınız proqramı lisenziyalı olmasına siz qərar verərək rəsmi şirkətlə əlaqə qurub lisenziya tətbiq edə bilərsiniz.
Hal-hazırda PyQt riverbank Ltd şirkəti tərəfindən səhmləşdirilir.
PyQt ,Tkinter kimi Python ilə bərabər gəlmədiyindən yüklənilməsinə ehtiyyac var.Bunun üçün aşağıdakı yükləmə qaydalarına nəzər yetirin.

PyQt ilə fotoları sənədləri,xml və html formatlar,audio video üzərində dəyişikliklər etmək olur.Bununla yanaşı paket formasında gələn hazır dialoglardan(QFileDialog,QFontDialog və s)istifadə etmək olur.
Aşağıda isə şəkil formatlarını dəstəkləyir(QPixmap klası)

BMP	Windows Bitmap
GIF	Graphic Interchange Format (optional)
JPG	Joint Photographic Experts Group
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
PBM	Portable Bitmap
PGM	Portable Graymap
PPM	Portable Pixmap
XBM	X11 Bitmap
XPM	X11 Pixmap

Yükləmə qaydası

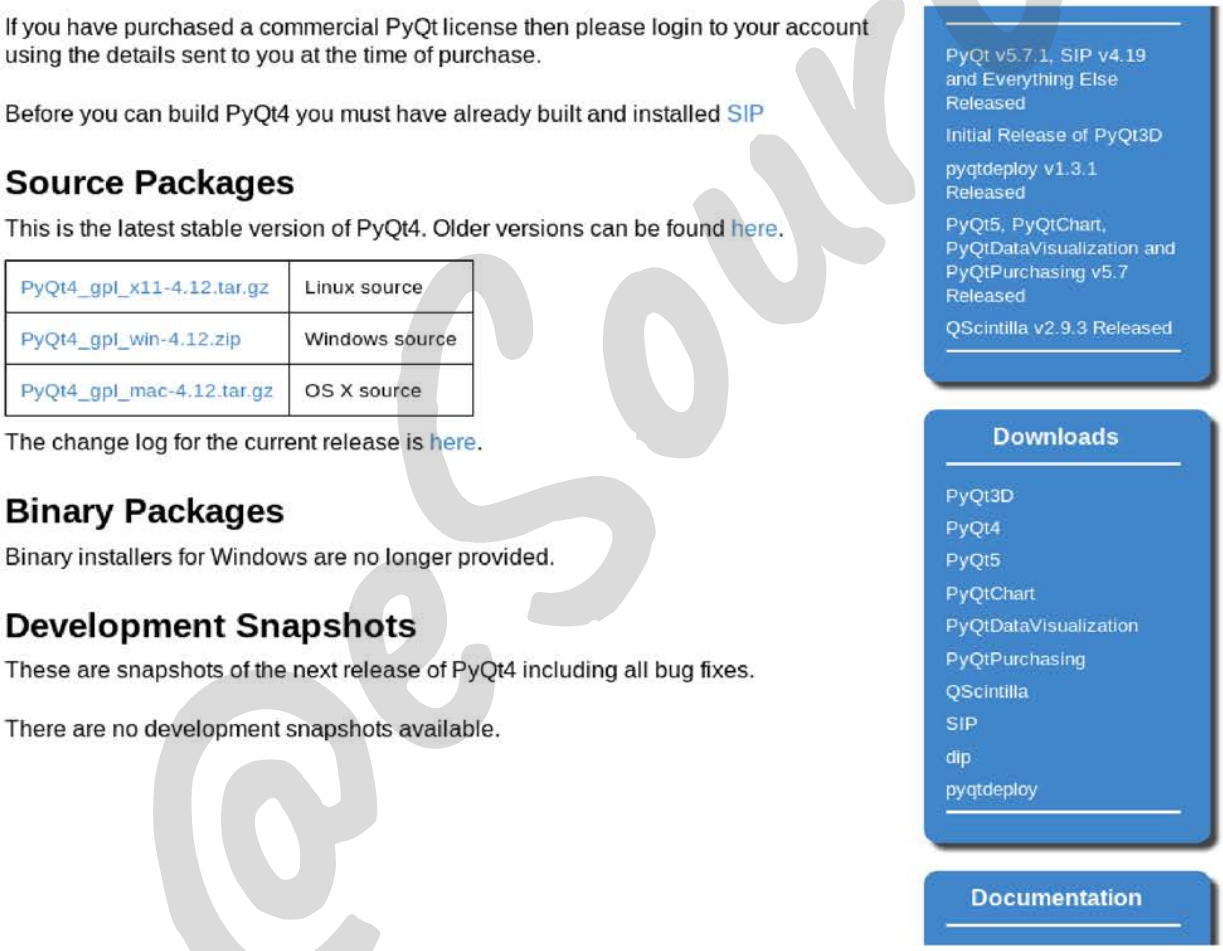
İlk əvvəl sisteminizdə Python yüklü olmalıdır.PyQt python2 və python3 üçün ayrı yüklənilir.Hal-hazırda PyQt , PyQt5xx versiyası üzərindədir.
Pythonu yükləmək qaydalarına dərindən girməyəcəm.

Windows

Windows üçün 32bit və ya 64bit -ə uyğun olaraq python-u(2.7xx və ya 3.4xx) yükləyirik

Daha sonra PyQt(Qt- 'kü te')üçün aşağıdakı ünvandan yükləməni həyata keçiririk

https://www.riverbankcomputing.com/software/pyqt/download



If you have purchased a commercial PyQt license then please login to your account using the details sent to you at the time of purchase.

Before you can build PyQt4 you must have already built and installed SIP

Source Packages

This is the latest stable version of PyQt4. Older versions can be found here.

PyQt4_gpl_x11-4.12.tar.gz	Linux source
PyQt4_gpl_win-4.12.zip	Windows source
PyQt4_gpl_mac-4.12.tar.gz	OS X source

The change log for the current release is here.

Binary Packages

Binary installers for Windows are no longer provided.

Development Snapshots

These are snapshots of the next release of PyQt4 including all bug fixes.

There are no development snapshots available.

PyQt v5.7.1, SIP v4.19 and Everything Else Released
Initial Release of PyQt3D
pyqtdeploy v1.3.1 Released
PyQt5, PyQtChart, PyQtDataVisualization and PyQtPurchasing v5.7 Released
QScintilla v2.9.3 Released

Downloads

PyQt3D
PyQt4
PyQt5
PyQtChart
PyQtDataVisualization
PyQtPurchasing
QScintilla
SIP
dip
pyqtdeploy

Documentation

Linux

Gnu Linux üçün

Bunun üçün terminalı açırıq

```
$ sudo apt-get update ; sudo apt-get dist-upgrade
$ sudo apt-get install python
$ sudo apt-get install python3
```

və python,python3 -ü sistemə yükləyirik.Ardından

```
$ sudo apt-get install python-qt4
$ sudo apt-get install python-qt5
```

Daha sonra aşağıdakı kodu terminala yazaraq enter-ə basın

sudo apt-get install build-essential, qt4-qmake, qt4-dev-tools

əmr aşağıdakı paketləri yükləyir.

dpkg-dev g++ gcc libc-dev make

Əgər Ubuntu istifadəçisinizsə yuxarıdakı kitabxana sisteminizdə kökdən yüklənəcək.Ubuntu istifadəçisi deyilsinizsə dpkg-dev -paketini qurmağa çalışmayın.Sadəcə g++ paketini yükləməniz yetərlidir.

Və ya yuxarıdakı ünvana daxil olub tar.gz faylını sisteminizə yükləyə bilərsiniz.

Mac Os üçün

brew install pyqt

Sistemə xətasız yüklədikdən sonra,versiyasını təyin edək

Terminaldan python-u çağırırıq

```
$ python
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from PyQt4 import QtCore
>>> QtCore.PYQT_VERSION_STR
'4.10.4'
```

və python2 üçün '4.10.4' üzərindəyik.Eyni qaydada python3 üçün tətbiq edə bilərik

```
$ python3
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> from PyQt4 import QtCore
>>> QtCore.PYQT_VERSION_STR
'4.10.4'
```

Yuxarıda _STR yazmaqda məqsədimiz versianı daha aydın şəkildə görünməsini təyin etməkdir.PYQT_VERSION_STR QtCore daxilində bir funksiyadır.Pythonda olduğu kimi PyQt daxilindən QtCore import etdik və PYQT_VERSION_STR funksiyasını istifadə etdik.Bunları əyani görmək üçün 'dir' funksiyasından istifadə etsək

```
>>> dir(QtCore)

['PYQT_CONFIGURATION', 'PYQT_VERSION', 'PYQT_VERSION_STR',
'QAbstractAnimation', 'QAbstractEventDispatcher', 'QAbstractFileEngine',
'QabstractFileEngineHandler'.....................................
.....................................................
```

əldə edə bilərik.

Bundan başqa Qt designer vasitəsilə qrafik görünüşü hazırlayıb daha sonra ui faylını py faylına çevirə bilərsiniz.

PyQt4

PyQt kitabxanası,daxilində 600-dən çox klas və funksiya daşıyır.Bunları tək-tək yaza bilməsəkdə bəzilərindən,eləcədə irəlilədikcə metod və funksiyalarından istifadə edəcəyik.

QtCore - klas qrafik interfeyslə əlaqəsi olmayıb,daxilində digər modulları əhatə edir.
QtGui - klas qrafik interfeyslə birbaşa əlaqəlidir(user interface).Görüntünü təşkil edir.
QtSql - verilənlərin bazası ilə əlaqədar olub,bazanın yaradılması,əlavələri təşkil edir.Misal üçün SQLite,MySQL,Pl/SQL və s.
QtSvg - vektorların təyinində istifadə olunan moduldur.(SVG)
QtOpenGl - OpenGl dəstəklənməsi
QtNetwork - şəbəkə ilə əlaqəli sinifdir.
QtMultimedia - multimedia üzərində dəyişiklərdə istifadə olunur.
QtXml - xml faylların düzəlişində istifadə olunur.
QtScript (QtScriptTools) – Skriptlərin istifadəsi üçün nəzərdə tutulub.
Qt - bu sinif bütün modullara,parametrlərə qoşulur.

Xatırlayırsınızsa pythonda modul bölməsində,modulların çağırılma qaydası ilə tanış olmuşduq.Misal olaraq

1. from os import name

```
>>> from os import name
>>> os.name
'posix'
>>>
```

Linuks üçün

2. import os
3. from os import *

1-ci ifadədə os modulu daxilindən name funksiyasını çağırdıq.Yəni os modulundan bizə yalnız name funksiyasını import et əmri verdik.

Eyni qayda ilə PyQt4 -ün yüklü olduğunu təyin edək

```
>>> from PyQt4 import QtCore
>>>
```

və aşağı sətirə xətasız keçid etdik.Deməli pyqt sistemdə yüklüdür.

Qrafik proqramı yazmaq üçün bizə python-a dəstək verən editor,proqramlardan istifadə edə bilərik.Bunlardan ən sadəsi mətn editorudur.(Notepad++ ,leafpad,getid,vim(gui) sublime text,atom və s istifadə edə bilərsiniz)

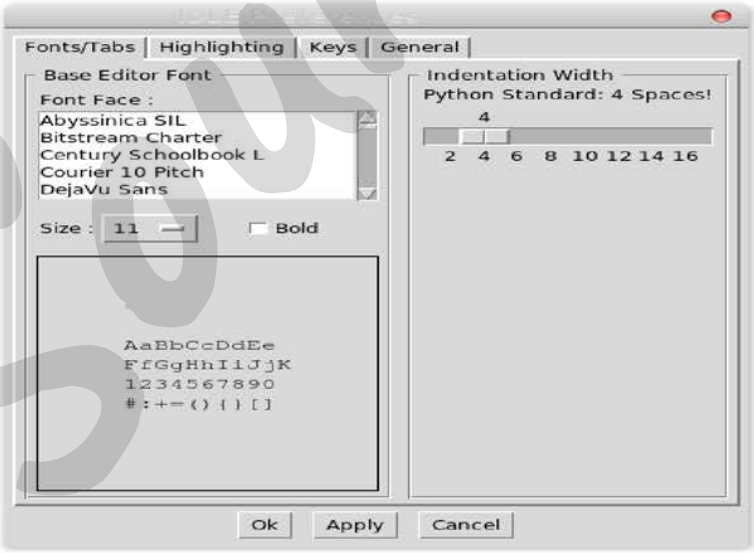
Mən sistemə yüklədiyim idle istifadə edəcəyəm(Python shell).

 $ sudo apt-get install idle (Windows sistemlərində Python2.7(python3.4) yüklədikdə idle(idle3), pythonla bərabər gəlir.)
Davam edərək

İdle → File → New file → yolu ilə yeni fayl açırıq.Mən faylın adını yeni.py adla yaddaşa saxladım.
Bundan əlavə idle üçün bəzi tənzimləmələri başdan edəkki irəlidə xətalarımız olmasın.Yəni davamlı qarşılaşdığınız indentation üidth problemini həll edək.Bunun üçün

Tools → configure idle →
Fonts/Tabs ...

Şəkildəki uyğunluqda tənzimləyin.



Tənzimləmədən sonra Apply və Ok düyməsinə sıxıb python editorunu hazır şəklə gətiririk.

İlk PyQt kodlarını yazaq

```
   # -*-coding: utf-8 -*-
1    from PyQt4 import QtCore,QtGui
2    import sys
3    app=QtGui.QApplication(sys.argv)
4    app.exec_()
```

Kodlarımızı yazıb istənilən bir adla(yeni.py) yaddaşa veririk.Daha sonra python-shell dən run verib çalışdırırıq.Nəticədə qarşımıza boş bir pəncərənin açıldığını görürük.

1-sətirdə yazdığımız kod PyQt4 daxilindən QtCore və QtGui modullarını import etmək deməkdir.
2-sətirdə isə sys modulunu çağırırıqki Qapplication(3-cü sətir) daxilində argument olaraq dəyişkəni qeyd edirik.sys.argv yerinə boş bir list ifadəsi də qeyd etsək heç bir xəta almayacağıq.4-cü sətirdə isə kod blokunu bağlayırıq. Sys.exit ifadəsi pəncərənin təşkili üçün app.exec_() funksiyasını çalışdırır.əgər biz sys.exit ifadəsini istifadə etməsək qarşımıza heç bir pəncərə açılmayacaq. Və ya

```
# -*-coding: utf-8 -*-
from PyQt4 import QtCore,QtGui
import sys
app=QtGui.QApplication([])
w=QtGui.QWidget()
w.show()
sys.exit(app.exec_())
```

Sys.argv list ilə əvəzlədik

ifadəmizi list şəklində göstərdikdə sys.exit ifadəsinə ehtiyyac olmayacaq

```
# -*-coding: utf-8 -*-
from PyQt4 import QtCore,QtGui
import sys
app=QtGui.QApplication([])
w=QtGui.QWidget()
w.show()
app.exec_()
```

Nəticə etibarı ilə kodlarımız iki , baş və köməkçi hissələrdən ibarətdir.

```
app=QtGui.Qapplication([])
ifadələr.....
app.exec_()
```

app.exec() bir modulun bağlandığını düşünün.Yəni biz pythonda bir def funksiyası yazdığımız zaman onun funksionallığı üçün sonda def ifadəsini bağladığımız kimi düşünün.

```
def first():            def first(name):
    pass                    pass
first()                 first('Kant')
```

sys.exit → app.exec_()

Qeyd edimki app -ifadəsini istənilən dəyişkənlə əvəz edə bilərsiniz.Yazdığımız bir application olduğu üçün qısaca bu ifadəni istifadə etdim.exec ifadəsi executive sözünün qısaltması olub dilimizə 'çalışdır' kimi tərcümə olunur.Tərcüməsini nəzərə alaraq app(application) -i çalışdır əmri veririk.

```
# -*-coding: utf-8 -*-
from PyQt4 import QtCore,QtGui
import sys
fi=QtGui.QApplication([])
fi.exec_()
```

Bundan başqa ifadələrimizi bir def(define) altında da qeyd edib çalışdıra bilərik.

```
1.  # -*-coding: utf-8 -*-
2.  from PyQt4 import QtGui
3.  import sys
4.  def pencere():
5.      app=QtGui.QApplication(sys.argv)
6.      window=QtGui.QWidget()
7.      window.show()
8.      sys.exit(app.exec_())
9.  pencere()
```

Kodlarımız arasında yenilik olan 3 və 4-cü sətirdəki ifadələrdir.3-cü sətirdəki ifadə QWidget bir interfeys vidjeti, QGui daxilindəki(və ya digər modullar) metod və funksiyaları pəncərədə görünməsini təşkil edir.Və özüdə boş bir pəncərəni təmsil edir.PyQt -də əsasən iki pəncərə növü mövcuddur.Biri sabit boş pəncərə-hansıki hal-hazırda istifadə edirik,digəri isə QMainWindow -daxilində menu toolbar,statusbar,frame daşıyıcısıdır.4-cü sətirdə isə yazdığımız QWidget modulunu göstərmək üçün window-dəyişkəninə qeyd etdik.Hər halda göstəriləcək digər klas olmasada ilk başdan bunu qeyd edib izah etməyə çalışaq.

Prinsip etibarı ilə PyQt aşağıdakı formada paketlərə mənsubdur.

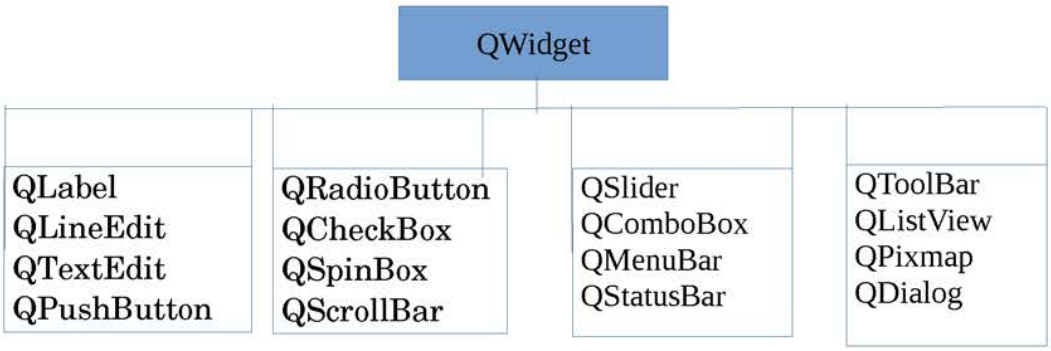
Module → Widget , Class → method → paramethers → function

Kodları yazmağın digər yolu,sinif daxilinə yerləşdirməkdir.Bu haqda irəlidəki bəhslərdə danışacağıq.Aşğıdakı kodlarda öyrənmədiyimiz klaslar olsada,sadəcə sinif daxilinə tətbiqi yoluna baxaq.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
from time import strftime
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('Class')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.button=QPushButton('Oclock',self)
        self.button.setToolTip(u' <b>Saat</b> ')
        self.button.setStyleSheet('QToolTip { font-size: 12pt; font-family:
Sansserif; }')
        self.button.move(50,20)
        self.timer=QTimer(self)
        self.timer.start(1000)
        self.timer.timeout.connect(self.clock)
        self.lcd=QLCDNumber(self)
        self.lcd.setGeometry(80,80,200,60)
        self.show()
    def clock(self):
        self.lcd.display(strftime("%H"+":"+"%M"))
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

QWidget(QMainWindow) -in ala bildiyi klaslar aşağıdakılardır.



Yuxarıdakı klaslara tək-tək nəzər yetirəcəyik.

Boş pəncərəmizə bir yazı ifadəsini göstərək.Bunun üçün QGui klası olan QLabel istifadə edəcəyik.Label sözü dilimizə etiket kimi tərcümə olunur.

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui
import sys
def pencere():
1.      app=QtGui.QApplication(sys.argv)
2.      window=QtGui.QWidget()
3.      label=QtGui.QLabel('Salam Azərbaycan',window)
4.      label.show()
5.      window.show()
6.      sys.exit(app.exec_())
7.  pencere()
```

if __name__ == '__main__':   -ifadəsi

Kodları çalışdırmağın digər yoludur

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    window=QtGui.QWidget()
    label=QtGui.QLabel(u'Salam Azərbaycan',window)
    window.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Kodlarımız daxilində yeni metod olan(3-cü sətir) Qlabel klası ,pəncərə daxilində 'Salam Azərbaycan' -ifadəsini eləcədə bu ifadənin görüntülənməsi üçün QWidget klasından istifadə etdik.əgər biz sadəcə label=QtGui.QLabel('Salam Azərbaycan') kodlarını yazsaq,pəncərədə heç bir dəyişikliyin olduğunu görə bilməzdik.4-cü sətirdə isə label.show() ifadəsindən istifadə etməyə də bilərik.Çünki biz başdan label -ifadəsini Widget ilə təyin etdiyimiz üçün və sonda window.show() yazaraq bunu göstərdiyimizə görə label.show() ifadəsinə ehtiyyac yoxdur.Və proqramımız çalışdığı zaman dil problemi ilə qarşılaşdığımıza görə

QtGui.QLabel(u'Salam Azərbaycan') u-unicode parametrini əlavə etməklə aradan qaldıra bilərik.

Kodlarımız

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    window=QtGui.QWidget()
    label=QtGui.QLabel(u'Salam Azərbaycan',window)
    window.show()
    sys.exit(app.exec_())
pencere()
```

Unicode

Ekran görüntüsü

setText metodu

QLabel daxilində ifadəni aldığı kimi,bu ifadəni kənardan da göstərə bilərik.Bunun üçün setText metodundan istifadə edəcəyik.

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    window=QtGui.QWidget()
    label=QtGui.Qlabel(window)
    #setText parametri
    label.setText(u'Salam Azərbaycan')
    window.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Yazı rəngləri

rənglərin dəyişdirilməsi üçün PyQt daxilindəki modullardan eləcədə html atributları ilə də təyin edə bilərik

```
        # -*-coding: utf-8 -*-
        from PyQt4 import QtGui
        import sys
        def pencere():
1.          app=QtGui.QApplication(sys.argv)
2.          window=QtGui.QWidget()
3.          label=QtGui.QLabel(window)
4.          label.setText("<font color='Red'>Hello Python</font>")
5.          window.show()
6.          sys.exit(app.exec_())

        if __name__ == '__main__':
            pencere()
```

4-cü sətirdə qeyd etdiyimiz html rəng seçimi ilə yanaşı mətn də daxil edə bildik.
Digər alternativ yol setColor və setPalette metodlarıdır.Bunun üçün kodlarımıza bəzi əlavələrimizi edəcəyik.

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    window=QtGui.QWidget()
    reng=QtGui.QPalette()
    reng.setColor(QtGui.QPalette.Foreground,QtCore.Qt.blue)
    label=QtGui.QLabel(window)
    label.setText('Hello world')
    label.setPalette(palette)
    window.show()

    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

QtCore modulunu əlavə edirik

Ön plan rəngi, mavi

və ya QWidget ifadəsini çıxarıb birbaşa label ifadəsini pəncərə daxilində göstərək.

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Foreground,QtCore.Qt.blue)
    label=QtGui.QLabel()
    label.setText('Hello world')
    label.setPalette(palette)
    label.show()
```

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

if __name__ == '__main__':
    pencere()
```

Bu qayda ilə biz pəncərəmizin də arxa plan rəngini dəyişə bilərik.Kodlarımıza əlavələrimizi edərək

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    window=QtGui.QWidget()
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
    window.setPalette(palette)
    label=QtGui.QLabel(window)
    label.setText("<font color='Blue'>'Hello world'</font>")
    label.show()
    window.show()

    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Ön planı(Foreground) arxa plan(Background) olaraq dəyişirik.

Ekran görüntüsü

setFont parametri

Biz bu parametr vasitəsilə yazı tiplərini,qalın,əyri və s dəyişə bilərik.Bunun üçün QFont klasından istifadə edəcəyik.Klas birbaşa QFontDialog vidjetindən parametrləri alır

```
    # -*-coding: utf-8 -*-
    from PyQt4 import QtGui,QtCore
    import sys
    def pencere():
1.      app=QtGui.QApplication(sys.argv)
2.      font=QtGui.QFont()
3.      font.setItalic(True)
4.      window=QtGui.QWidget()
5.      palette=QtGui.QPalette()
6.      palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
7.      window.setPalette(palette)
8.      label=QtGui.QLabel(window)
9.      label.setText("<font color='Blue'>Hello world</font>")
10.     label.setFont(font)
11.     label.show()
12.     window.show()
13.     sys.exit(app.exec_())

    if __name__ == '__main__':
        pencere()
```

2-ci sətirdə QFont klasını istifadə etdik,daha sonra 3-cü sətirdə font tipini italic olaraq ifadə edib bunu yazımıza label.setFont(font) ifadəsi ilə tətbiq etdik.

setWindowTitle

Metod vasitəsilə pəncərə başlığını dəyişmək olur.

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
   app=QtGui.QApplication(sys.argv)
   font=QtGui.QFont()
   font.setItalic(True)
```

```
    window=QtGui.QWidget()
    window.setWindowTitle(u'Bu bizim ilk proqramımız')
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
    window.setPalette(palette)
    label=QtGui.QLabel(window)
    label.setText("<font color='Blue'>Hello world</font>")
    label.setFont(font)
    label.show()
    window.show()

    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Ekran görüntüsü

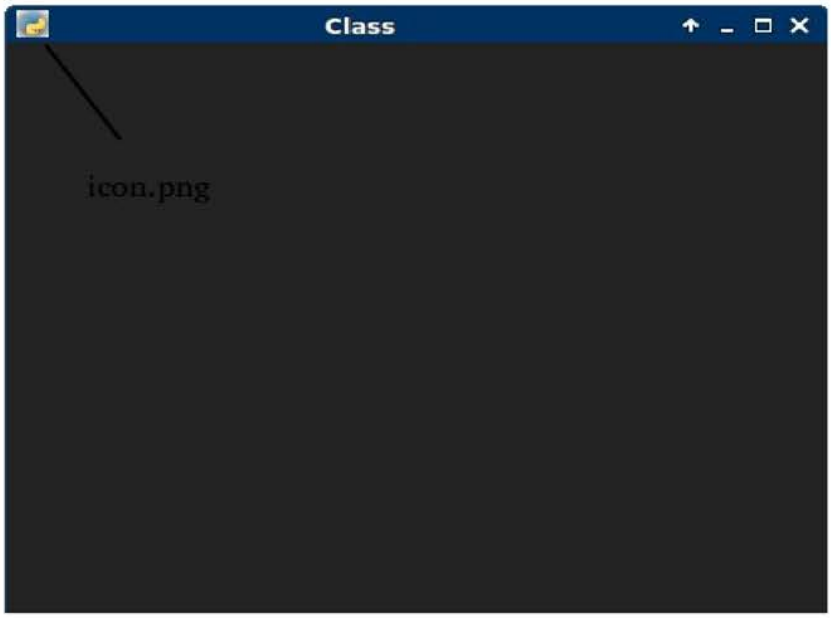
setWindowIcon metodu

metod vasitəsilə pəncərəyə icon əlavə edə bilərik.Bunun üçün kodlarımıza window.setWindowIcon(QtGui.QIcon('../icon.png')) ifadəsini əlavə edəcəyik.Qeyd edimki 'icon' ,py faylını saxladığınız qovluqda olmalıdır.Və ya ünvanı göstərmək lazımdır('/home/user/Picture/icon.png')

Davam edərək

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    font=QtGui.QFont()
    font.setItalic(True)
    window=QtGui.QWidget()
    window.setWindowTitle(u'Bu bizim ilk proqramımız')
    window.setWindowIcon(QtGui.QIcon('icon.png'))
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
    window.setPalette(palette)
    label=QtGui.QLabel(window)
    label.setText("<font color='Blue'>Hello world</font>")
    label.setFont(font)
    label.show()
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Icon əlavəsi



resize() parametri (eni və uzunluğu)

Parametr vasitəsilə pəncərə ölçüsünü tənzimləyə bilərik.Bunun üçün kodlarımız arasına window.resize(width,height) ifadəsini əlavə edəcəyik.Qeyd edimki parametr daxilində ardıcıllıqla width daha sonra height ifadəsi gəlir.

Kodlarımız

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    font=QtGui.QFont()
    font.setItalic(True)
    window=QtGui.QWidget()
    window.setWindowTitle(u'Bu bizim ilk proqramımız')
    window.setWindowIcon(QtGui.Qicon('icon.png'))
                x   y
    window.resize(400,400)
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
    window.setPalette(palette)
    label=QtGui.QLabel(window)
    label.setText("<font color='Blue'>Hello world</font>")
    label.setFont(font)
    label.show()
    window.show()

    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Pəncərə ölçüsü

x – width   y – height

Bura qədər qeyd etdiyimiz bəzi parametrləri digər modullarda da istifadə edəcəyik.Yəni parametr sırf bir modul üçün(bəziləri) nəzərdə tutulmur.

move metodu Koordinasiya (yerləşdirmə)
move(x,y)

Pəncərənin daxilindəki hello world ifadəsinə fikir verirsinizsə,koordinasiyası yuxarı baş hissədə yer alır.Amma biz bir proqram yazdıqda bunu mərkəzdə və ya sağ üst hissədə və s nöqtələrdə göstərmək istəsək,qeyd olunan(move) metodundan istifadə edəcəyik.Yəni label ifadəmizə label.move(x,y) kimi əlavə edəcəyik.
x -koordinat oxu üzrə x – istiqamətində(soldan sağa)
y -koordinat oxu üzrə y – istiqamətində(yuxarıdan aşağı)

Kodlarımız

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    font=QtGui.QFont()
    font.setItalic(True)
    window=QtGui.QWidget()
    window.setWindowTitle(u'Bu bizim ilk proqramımız')
    window.setWindowIcon(QtGui.QIcon('icon.png'))
    window.resize(400,400)
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
    window.setPalette(palette)
    label=QtGui.QLabel(window)
    label.setText("<font color='Blue'>Hello world</font>")
    label.setFont(font)
    label.move(150,20)
    label.show()
    window.show()

    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Etiketin(label) yerləşdirilməsi

Ekran görüntüsü

setGeometry(x,y,width,height) metodu

Metod pəncərəni,lövhədə(Desktop)uyğun parametrlərə görə yerləşdirir.Bu metodun aldığı dörd parametrdən ikisi,resize metodunu əvəzləyə bilir.

Kodlarımızda window.resize(400,400) (metodunu)ifadəsini silib yerinə window.setGeometry(200,200,400,400) -ifadəsini(metodunu) əlavə edək

```
# -*-coding: utf-8 -*-
from PyQt4 import QtGui,QtCore
import sys
def pencere():
    app=QtGui.QApplication(sys.argv)
    font=QtGui.QFont()
    font.setItalic(True)
    window=QtGui.QWidget()
    window.setWindowTitle(u'Bu bizim ilk proqramımız')
    window.setWindowIcon(QtGui.QIcon('icon.png'))
    window.setGeometry(200,200,400,400)
    palette=QtGui.QPalette()
    palette.setColor(QtGui.QPalette.Background,QtCore.Qt.red)
    window.setPalette(palette)
    label=QtGui.QLabel(window)
```

Yeni əlavə etdiyimiz kod

```
    label.setText("<font color='Blue'>Hello world</font>")
    label.setFont(font)
    label.move(150,20)
    label.show()
    window.show()

    sys.exit(app.exec_())

if __name__ == '__main__':
    pencere()
```

Proqramı çalışdırdıqda qarşımıza çıxan pəncərənin yerini dəyişdiyini görürük.Yeni kodlarımızda ilk 200-argumenti pəncərənin x- oxu istiqamətindən məsafəsi(soldan sağa),ikinci 200-argumenti y-oxu istiqamətində(yuxarıdan aşağı) yerləşməsi,digərləri isə ardıcıl olaraq pəncərənin eni və uzunluğunu (width,height) təşkil edir.

Yuxarıda göstərilənlərdən başqa QLabel,aşağıdakı metodları da alır.

setAlignment() metodu

Metod,ifadəni sağ,sol,mərkəz və bir yana əymək kimi parametrlər alır. (AlignLeft,AlignRight,AlignCenter,AlignJustify,AlignBottom)

Parametrə dair yeni kodları yazaq.


```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import *
from PyQt4.QtCore import *
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('Hello PyQt')
    label=QLabel()
    label.setText("<A href='www.techazweb.wordpress.com'>Click me</a>")
    label.setAlignment(Qt.AlignCenter)
```

```
    label.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Kodlarımızda setText daxilində html atributları ilə, link ünvanını göstərib və linkə dəyişən(Click me) verdik.Və linki mərkəzdə(center) göstərmək üçün AlignCenter parametrini verdik.

setPixmap   metodu (QLabel)

Metod, şəklin pəncərədə görünməsini təşkil edir.

Kodlarımız

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setGeometry(400,300,256,256)
    window.setWindowTitle('Hello PyQt')
    label=QLabel(window)
    photo=QPixmap('icon.png')  #və ya ('/home/user/Picture/icon.png')
    label.setPixmap(photo)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Ekran görüntüsü

Text() metodu (QLabel)

Metod ifadənin görüntüsünü əldə edir.Bəzi metod və parametrlərə dair misalların olmaması hələ az bir hissəsini (PyQt) öyrənməyimizdən irəli gəlir.Yəni irəlilədikcə geridə keçdiyimiz metod və parametrlərdən də davamlı istifadə edəcəyik.

linkActivated metodu (QLabel)

Metod pəncərə daxilində qeyd olunmuş bir link -i(ünvanı) aktiv hala gətirmək ,eləcədə digər funksiyalarla ünvanı əlaqələndirmək üçün nəzərdə tutulub.Buna dair misal yazaq

Kodlarımız

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,webbrowser
1.  def clickme(event=None):
2.      webbrowser.open_new(r"https://google.com")
3.  def pencere():
4.      app=QApplication(sys.argv)
```

webbrowser modulu

```
5.              window=QWidget()
6.              window.setGeometry(400,300,256,256)
7.              window.setWindowTitle('Hello PyQt')
8.              label=QLabel(window)
9.              label.setText("<A href='www.google.com'>GOOGLE</A>")
10.             label.linkActivated.connect(clickme)
                window.show()
                sys.exit(app.exec_())
        if __name__ == '__main__':
                pencere()
```

Proqramı çalışdırdığımız zaman qarşımıza çıxan Google ifadəsinə basdıqda qarşımıza google.com brauzer vasitəsilə açılır.Bu funksiyanı 1-ci sətirdə yazdığımız def clickme funksiyası yerinə yetirir.Kodlarımız arasında yeni olan connect(qoşulmaq) metodu isə yeni olduğundan irəlidə buna dair geniş bəhs edəcəyik.Hələlik bunu yazmaqda məqsədimiz linkActivated metodunu izah etmək idi.

linkHovered metodu

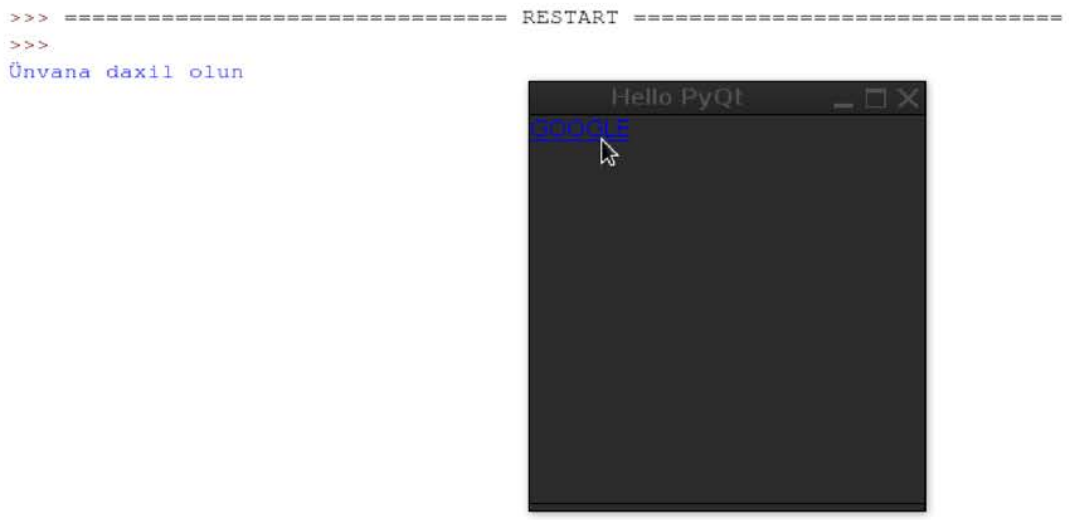
Metod,daxilində digər funksiyaları da çağıra bilir.Və mouse hərəkətini link üzərində təyinatını qeyd edir.əgər biz mausu link-in üstünə gətirdikdə,metoda verdiyimiz istənilən funksiya bunu təyin edə bilər.Bu bir standart mesaj və ya konsolda print vasitəsilə (istifadəçinin şərtlərinə bağlı) verilən ifadəni çap etmək kimi başa düşülə bilər.Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,webbrowser
def clickme(event=None):
    webbrowser.open_new(r"https://google.com")
def info():
    print "Ünvana daxil olun"

def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setGeometry(400,300,256,256)
    window.setWindowTitle('Hello PyQt')
    label=QLabel(window)
    label.setText("<A href='www.google.com'>GOOGLE</A>")
```

```
    label.linkActivated.connect(clickme)
    label.linkHovered.connect(info)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Ekran görüntüsü



ekran görüntüsündən də məlum olduğu kimi GOOGLE ifadəsinə mausu yönəltdikdə shell-də ' Ünvana daxil olun ' ifadəsi çap olundu.
Bura qədər QLabel metodlarına stop deyib QLineEdit klası- haqqında məlumat toplayaq

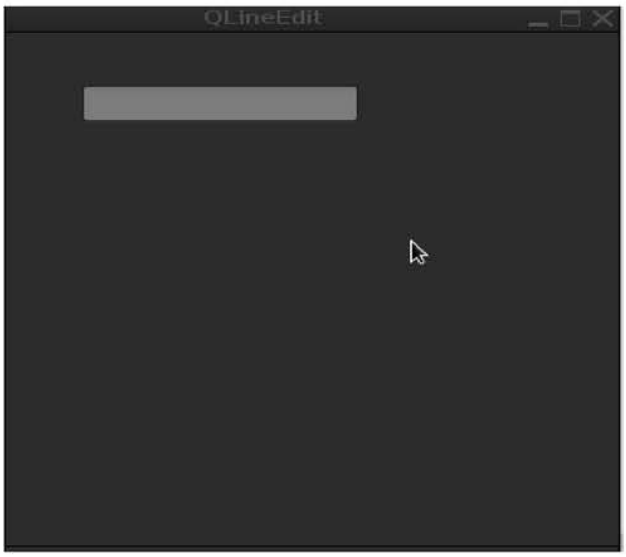
QLineEdit (class,Widget)

Bu sinif tək sətirli mətn girə biləcəyimiz qutudur.
lineedit=QLineEdit()

misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)       #yeni ifadəmiz,pəncərəyə tətbiqi
    entry.move(50,40)             #x və y oxu üzrə yerləşdirilməsi
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

ekran görüntüsü



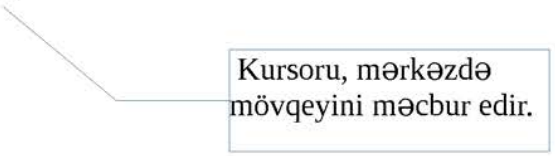
Klasın ala bildiyi metodlara nəzər yetirək

setAlignment() metodu

Metod haqqında Qlabel -dən kifayət qədər məlumatımız var.Eynilə klas daxilində metod parametrlərini (AlignLeft,AlignRight,AlignCenter,AlignJustify) alaraq funksiyasını dəqiqliklə tətbiq edir.Sadəcə olaraq metod qutu daxilində kursorun yerini təyin edir.Biz əgər AlignRight parametrini istifadə etsək,kursorun qutu daxilində sağ -baş hissədə yanıb söndüyünü görəcəyik.

Məsələn

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setAlignment(Qt.AlignCenter)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```



Programı çalışdırdıqda,kursorun mərkəzdə olduğunu gördük.Çünki parametr olaraq AlignCenter yazdıq.Siz digər parametrləri də yazaraq test edə bilərsiniz.

setEchoMode()   metodu

Metod daxilində dörd parametr alır.
1.QLineEdit.Normal
2.QLineEdit.NoEcho
3.QLineEdit.Password
4.QLineEdit.PasswordEchoOnEdit

QLineEdit.Normal

1ci parametr Metod daxilində normal vəziyyətidir,yəni parametri verməsəkdə metod yazı yazılacaq tipdədir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setAlignment(Qt.AlignLeft)
    entry.setEchoMode(QLineEdit.Normal)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Yeni parametrimiz

QlineEdit.NoEcho parametri

Parametr qutunu fəaliyyətsiz vəziyyətə gətirir.

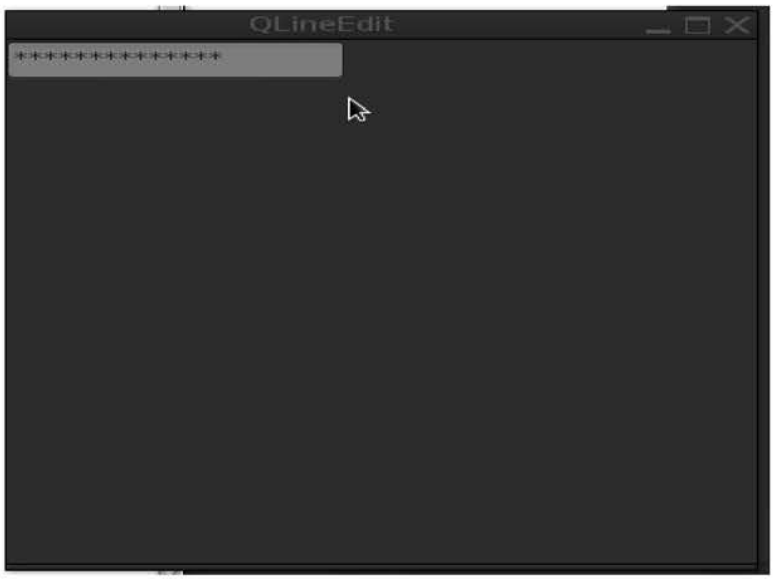
```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setAlignment(Qt.AlignLeft)
    entry.setEchoMode(QLineEdit.NoEcho)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

QlineEdit.Password parametri

Parametrdən şifrələrin soruşulması və kənardan görünməsinin qarşısının alınması üçün istifadə olunur.Qutuya daxil olan ifadəni ulduz işarələri altında gizlədir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setAlignment(Qt.AlignLeft)
    entry.setEchoMode(QLineEdit.Password)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Ekran görüntüsü



QLineEdit.PasswordEchoOnEdit parametri

Parametrdən şifrə qutusunun açıq şəkildə,ifadənin görünməsi üçün istifadə olunur.Bundan əvvəlki parametr ifadənin qapalı(ulduz işarələri) olmasını tənzimləyirdisə,bu parametr isə tamamilə açıq şəkildə tənzimləyir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setAlignment(Qt.AlignLeft)
    entry.setEchoMode(QLineEdit.PasswordEchoOnEdit)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

setMaxLength() metodu

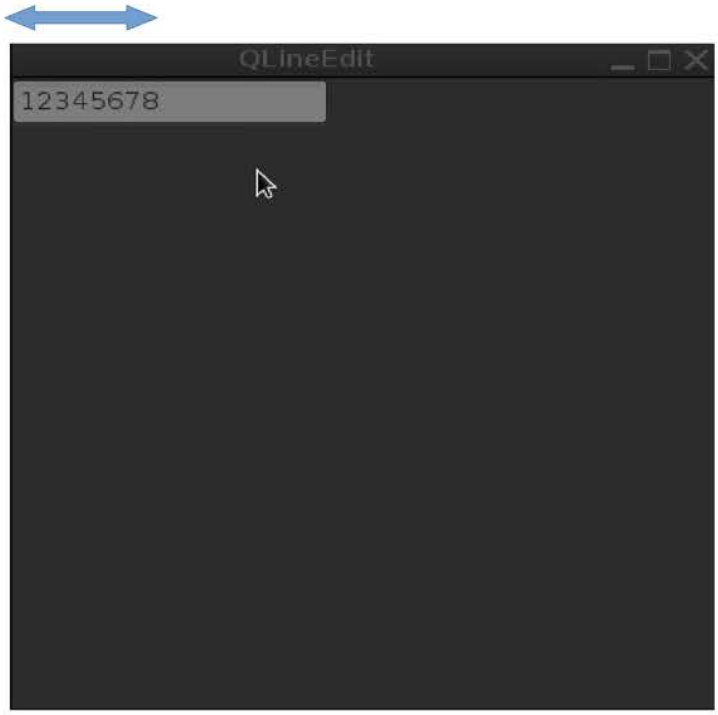
Metod qutuya daxil olan maksimum ifadə sayını tənzimləyir.Metod integer (tam ədədlər) cinsində ədədləri-parametr olaraq alır.Qutunun ifadə sayını 8 sayda tənzimləyək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setAlignment(Qt.AlignLeft)
    entry.setMaxLength(8)
    window.show()
```

Metod və 8 sayla tənzimləmə

```
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

ekran görüntüsü



setReadOnly() metodu

Metod iki parametr alır; True və False
Qutuda olan ifadənin sadəcə görünməsini təşkil etmək üçün metoda True paramterini veririk.
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
```

```
    entry.setText(u'mətni oxuyun')
    entry.setReadOnly(True)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

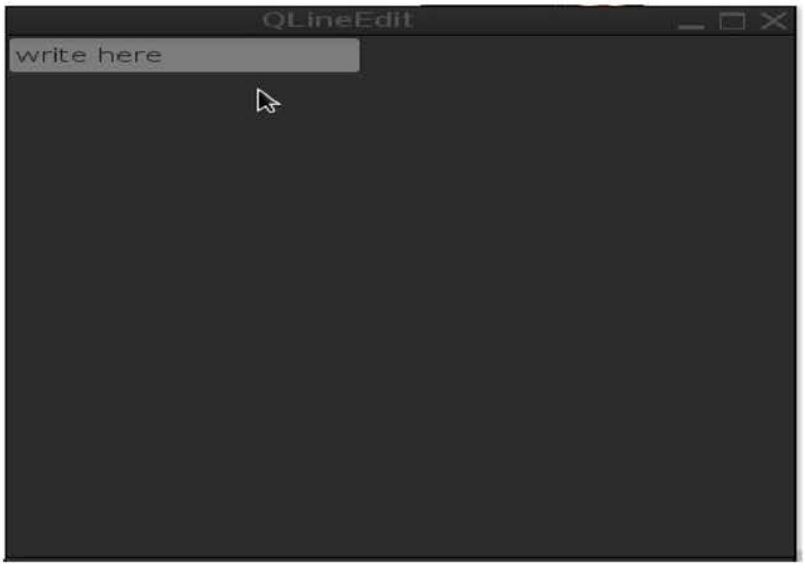
setText() metodu

metoda dair bundan əvvəlki bəhslərdə danışmışdıq.Metod,qutuya açılış zamanı ifadə daxil etmək üçün istifadə olunur

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setText(u'write here')
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Qutuda mətni göstərmək

ekran görüntüsü



text() metodu

Metod qutu daxilində olan ifadəni alır,kopyalayır.Bunu ifadə etmək biraz çətin olsada misallardan aydın olacaq.Metoda label klasında da rast gəlmişdik.Aşağıdakı kodlarda metodun funksiyası anlaşılacaq.
Misal olaraq

kodlarımız

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setText(u'write here')
    i=entry.text()
    print 'qutuda olan mətn ->',i
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Kodlarımız daxilində metodu bir i-dəyişəninə tətbiq etdik,daha sonra print funksiyası vasitəsilə qutudan mətni alıb çap etdik.Python shell-də 'qutuda olan mətn -> write here' ifadəsinin çap olunduğunu gördük.əlimizdə çox az məlumatların olmasından daha dolğun proqramların yazılmasını təşkil edə bilmirik.İrəlidəki bəhslərdə daha irəli səviyyədə proqramları təşkil edə biləcəyik.
Sadəcə başlanğıc üçün klas,metod və parametrlərdən istifadə qaydaları ilə tanış oluruq.

setValidator() metodu

Metod daxilind 2 ədəd parametr alır.metodun qutuda rolu,qutunu integer və kəsirli ədədlər üçün tənzimləyir.Normal halda bu metoddan istifadə

etməsək,qutu integer,string və decimal cinslərində ifadələri qəbul edir.Amma bəzən sırf bir cins üçün tətbiq etmək istəsək o zaman bu metoddan istifadə edəcəyik.Metodun ala bildiyi parametrlər
1.QIntValidator -tam ədədlər üçün
2. QDoubleValidator -kəsirli ədədlər üçün

Misallara baxaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    entry.setValidator(QIntValidator())
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Tam ədədlər üçün təşkili

Proqramı çalışdırdıqda klaviaturadan hərfləri daxil etməyə çalışsaqda,qutu tam ədədlər üçün limitləndiyindən istəyimiz baş tutmadı.Digər parametri daxil edib nə kimi nəticə əldə edəcəyinizi test edə bilərsiniz.

setInputMask() metodu

Metoddan ,qutu daxilində ifadəni müəyyən formatda tənzimləmək üçün istifadə olunur.

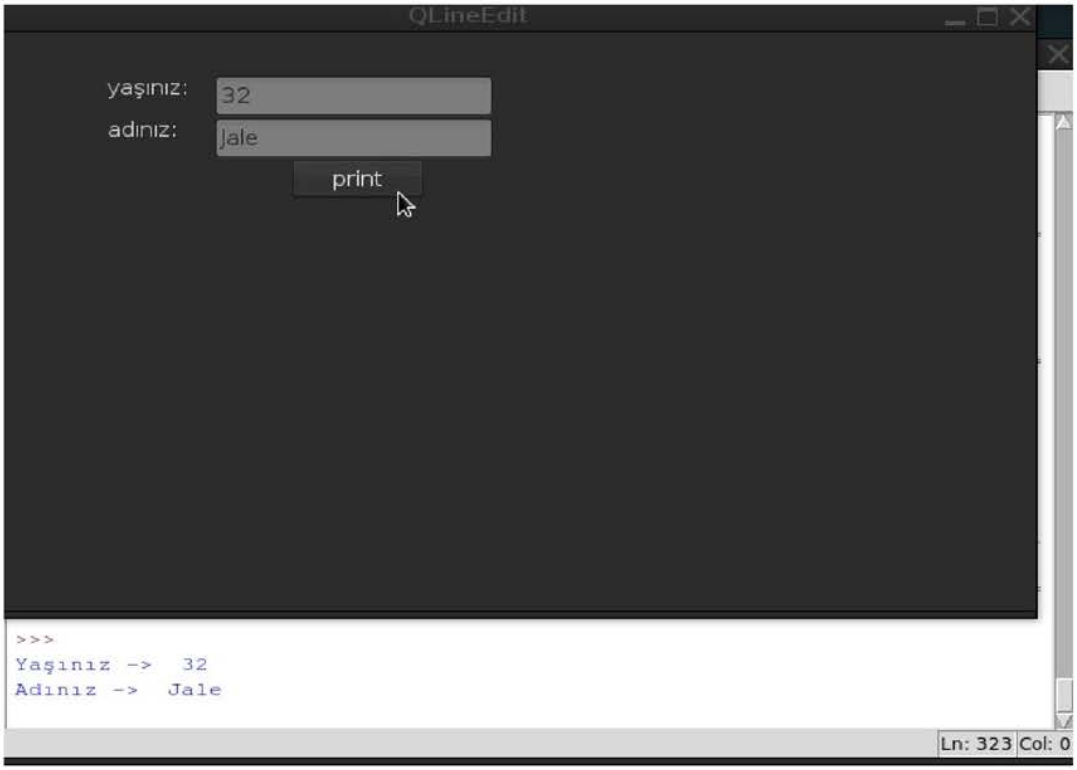
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
```

```
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    #etiket(label)-----------------------
    label=QLabel(window)
    label.setText(u'yaşınız:')
    label.move(70,30)
    #etiket(label)
    label_ad=QLabel(window)
    label_ad.setText(u'adınız:')
    label_ad.move(70,60)
    #ad-----------------------------------
    entry_yas=QLineEdit(window)
    entry_yas.setInputMask('99')
    entry_yas.setAlignment(Qt.AlignLeft)
    entry_yas.move(140,30)
    #yas ---------------------------------
    entry_ad=QLineEdit(window)
    entry_ad.move(140,60)
    #button
    def goster():
        print 'Yaşınız -> ' ,entry_yas.text()
        print 'Adınız -> ', entry_ad.text()
    button=QPushButton(window)
    button.setText('print')
    button.move(190,90)
    button.clicked.connect(goster)
    #----------+----------
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Yuxarıda kodlarımız arasına yeni olan buton metodunu əlavə etdik.Sadəcə olaraq yuxarıdakı metod üçün bunu tənzimlədik.Bundan sonrakı bəhsdə Button mövzusuna keçəcəyik.Haşiyədən kənara çıxmayaraq kodlarımıza izah verək.entry_yas.setInputMask('99') ifadəsini yazmaqla qutu daxilində yaş həddini eləcədə onluq say sistemdə tənzimlədik.Yəni istifadəçinin 100 kimi bir ədəd girməsinin qarşısını aldıq.Daha sonra buttona bir funksiya əlavə edərək qutu daxilində yazdığımız yaş və adımızı python-shell də görünməsini def goster(): - funksiyası vasitəsilə tənzimlədik.

Proqramın ekran görüntüsü



Bu yerdə qarşılaşacağınız xətalardan biri də dil kodlamasıdır.əgər ad bölməsinə ə,ğ,ö və s əlavə etsək unicode xətası alacağıq.

UnicodeEncodeError: 'ascii' codec can't encode character u'\u018f' in position 0: ordinal not in range(128)

Xətanın aradan qaldırılması ,unicode funksiyası

Buna görədə biz print 'Adınız -> ', unicode(entry_ad.text()) ifadəsinə unicode funksiyasını əlavə edərək xətadan çıxırıq.

Bütünlükdə kodlar deyil dəyişiklik edilən hissəni bura qeyd edəcəm

```
def goster():
    print 'Yaşınız -> ',entry_yas.text()
    print 'Adınız -> ', unicode(entry_ad.text())
```

sadəcə def goster funksiyası daxilində dəyişiklik etdik(unicode əlavəsi)

setFont metodu

Metod etiketdə(QLabel) olduğu kimi mətnin yazı tipini(bold,italic və s) təşkil edir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    entry=QLineEdit(window)
    font=QFont('Times',20,Qfont.Bold)
    #və ya entry.setFont(QFont(' ',20,QFont.Bold))  kimi yaza    bilərsiniz
    # entry.setFont(QFont("Arial",20))
    entry.setFont(QFont(font))
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Yazı tipini və ölçüsünü dəyişdikdə ,qutunun ölçüsü də dəyişdi.

Ekran görüntüsü

metodu html atributları ilə də yaza bilərik.Bunu etiketlərdə istifadə etdiyimizə görə təkrar yazmayacam.

editingFinished() metodu

Metod qutu daxilində mausu ,eləcədə enter düyməsini basdıqda funksiya vasitəsilə konsola və ya hər hansı bir dialoqla ifadələri çap edə bilərik.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    def printing():
        print 'print [+]'
    entry=QLineEdit(window)
    entry.editingFinished.connect(printing)
    entry.move(50,10)
    entry1=QLineEdit(window)
    entry1.editingFinished.connect(printing)
    entry1.move(50,40)
    entry2=QLineEdit(window)
    entry2.editingFinished.connect(printing)
    entry2.move(50,70)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

returnPressed() metodu

Metod qutu daxilində Enter duyməsini basmaqla funksiyanı fəaliyyətini təyin ed bilərik.Buna misal olaraq brauzerlərdə link daxil edib enter -ə basdıqda linki aktiv etməsidir.
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,webbrowser
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QLineEdit')
    window.setGeometry(400,400,400,400)
    def web():
        webbrowser.open('http://www.'+ entry.text())

    entry=QLineEdit(window)
    entry.returnPressed.connect(web)
    entry.move(50,10)

    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

google.com yazmaq kifayətdir

Yuxarıda ünvanı sadəcə xxxx.com yazmaqla sistemdə olan brauzer vasitəsilə ünvanı açırdıq.QWebkit modulundan istifadə edərək bir brauzer yaradaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
from PyQt4.QtWebKit import *
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
```

WebKit modulu

```
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,600,600)
    def web():
        web=QWebView(window)
        url='http://www.'+entry.text()
        web.load(QUrl(url))
        web.setGeometry(0,50,600,600)
        web.show()
    label=QLabel(window)
    label.setText('write url here')
    label.move(30,15)
    entry=QLineEdit(window)
    entry.setGeometry(180,10,400,30)
    entry.returnPressed.connect(web)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Kodlarımız arasında def web funksiyası daxilində web=QWebView(window) ana pəncərəyə tətbiq edirik.Daha sonra ünvanı istifadəçidən alacaq şəkildə yazırıq.Bunun üçün text() metodundan istifadə
edirik.url='http://www.'+entry.text()
ünvana qoşulmasını load metodu ilə tətbiq edərək ünvanın ölçüsünü setGeometry metodu vasitəsilə pəncərə daxilində görünüşünü təmin edirik.Və son olaraq ünvanın göstərilməsini web.show() vasitəsilə tənzimləyirik.entry.returnPressed.connect(web) ifadəsi qutuya ünvanı daxil edib enter düyməsini basdıqda web -funksiyasını çağırır.

selectionChanged() metodu

Metod,qutuya mausu basdığınızda(və ya qutudakı ifadəni seçdiyinizdə) funksiya vasitəsilə mesaj verə bilərik.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
```

```
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,600,600)
    def dep():
        print 'text changed'
    entry=QLineEdit(window)
    entry.setGeometry(180,10,400,30)
    entry.selectionChanged.connect(dep)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

textChanged() metodu

Bu metod əvvəlki metoddan fərqi,biz qutuda yazı yazmağa başladıqda mesajın verilməsini təmin edir.
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,600,600)


def dep():
        print 'text changed'
    entry=QLineEdit(window)
    entry.setGeometry(180,10,400,30)
    entry.textChanged.connect(dep)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Dəyişikliyi çap edir

Metod dep funksiyasını çağırır

textEdited() metodu

Metod,qutuya daxil etdiyimiz hər bir verilənə reaksiya verərək funksiya vasitəsilə verilənin fəaliyyətini bildirir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,600,600)
    def dep():
        print 'text edited'
    entry=QLineEdit(window)
    entry.setGeometry(180,10,400,30)
    entry.textEdited.connect(dep)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

cursorPositionChanged() metodu

Metod maus və kursora reaksiya verir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,600,600)
    def dep():
        print 'cursor changed'
    entry=QLineEdit(window)
    entry.setGeometry(180,10,400,30)
    entry.cursorPositionChanged.connect(dep)
```

```
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Klası yekunlaşdırıb növbəti klas,QPushButton haqqında məlumat toplayaq.

QPushButton

Klasa ,demək olarki hər gün rast gəlirik.Button-yəni düymə mənasını ifadə edir.Bəsit bir misal verək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    button=QPushButton(window)
    button.setText('start')
    button.move(120,30)
    #və ya button=QPushButton('start',window)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Pəncərəyə tətbiq edirik

Button mətni

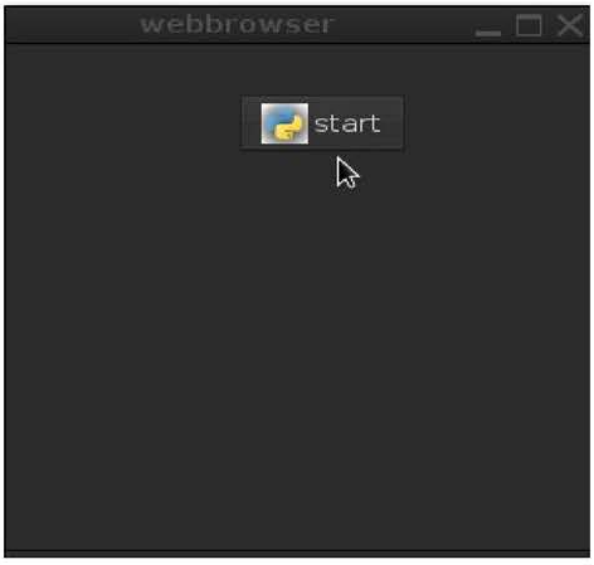
Kodlarımıza Button -u pəncərəyə tətbiq etdik.Pəncərəyə tətbiq etməni daha aydın şəkildə desəm,yəni irəlidə pəncərə daxilində pəncərələr yaradsaq bunu mütləq tətbiq etməliyik.Beləki button label lineedit klasların hansı pəncərədə yerləşməsini tətbiq etmək deməkdir.Digər klaslar kimi Button klası da metodlar alır.Bunlara nəzər yetirək

setIcon() metodu

Metod button daxilində icon və ya böyük ölçülü şəkillərin göstərilməsini təmin edir.
Metoda dair misala baxaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    button=QPushButton(window)
    button.setText('start')
    button.move(120,30)
    #və ya button=QPushButton('start',window)
    #icon tətbiqi
    button.setIcon(QIcon('icon.png'))
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Ekran görüntüsü



Düyməyə həm ifadə həmdə icon əlavə etdik.start -ifadəsini silsək sadəcə düymə daxilində icon görünəcəkdir.

Düymə fəaliyyətsiz olduğu üçün bir funksionallıq əlavə edək

Bunun üçün

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    button=QPushButton(window)
    button.setText('Time')
    button.move(0,10)

    #və ya button=QPushButton('start',window)
    #icon tətbiqi
    def oclock():
        label=QLabel(window)
        label.move(50,50)
        time1=''
        time2= time.strftime(('%H:%M:%S'))
        label.setText(time2)
        label.show()
    button.setIcon(QIcon('icon.png'))
    button.clicked.connect(oclock)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

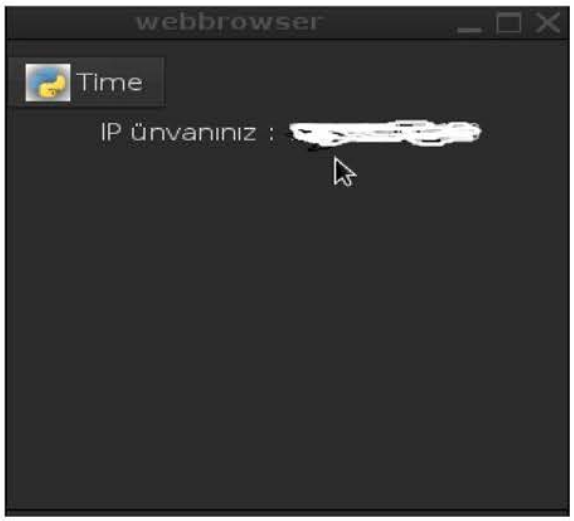
Düyməyə basdıqda pəncərədə kompyutermizdəki saat saniyə ilə bərabər çap olundu.

Və ya ip ünvanını təyin edək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
from json import load
from urllib2 import urlopen
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    button=QPushButton(window)
    button.setText('Time')
    button.move(0,10)
    #və ya button=QPushButton('start',window)
    #icon tətbiqi
    def oclock():
        label=QLabel(window)
        label.move(50,50)
        ip = load(urlopen('http://httpbin.org/ip'))['origin']
        label.setText(u'IP ünvanınız : '+ip)
        label.show()
    button.setIcon(QIcon('icon.png'))
    button.clicked.connect(oclock)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Yeni modullarımız

Ekran görüntüsü



Biz icon ölçüsünü də tənzimləyə bilərik.Kodlarımızı yazaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
from json import load
from urllib2 import urlopen
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    button=QPushButton(window)
    button.setText('Ip')
    button.move(0,10)
    #və ya button=QPushButton('start',window)
    #icon tətbiqi
    def oclock():
        label=QLabel(window)
        label.move(50,50)
        ip = load(urlopen('http://httpbin.org/ip'))['origin']
        label.setText(u'IP ünvanınız : '+ip)
        label.show()
    button.setIcon(QIcon('icon.png'))
    button.setIconSize(QSize(30,30))
    button.clicked.connect(oclock)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Yeni əlavəmiz

Ölçüsünü dəyişdikcə düymənin ölçüsü də dəyişir.

setCheckable() metodu

Metod iki parametr alır; True və False.Metod vasitəsilə düyməni aktiv və passiv hala gətirmək olur.

Metoddan istifadə etmədikdə düymə setCheckable(True) ifadəsi altında çalışır.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    button=QPushButton(window)
    button.setText('Time')
    button.move(0,10)
    #və ya button=QPushButton('start',window)
    button.setIcon(QIcon('icon.png'))
    button.setEnabled(False)  #düyməni passiv hala gətiririk
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Button -a dair daha bir misal yazaq

```
# -*-coding: utf-8 -*-
import pygame.camera
import pygame.image
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('webbrowser')
    window.setGeometry(50,10,300,300)
    def camera():
        pygame.camera.init()
        cam = pygame.camera.Camera(pygame.camera.list_cameras()[0])
```

pygame modulu

```
        cam.start()
        img = cam.get_image()
        pygame.image.save(img, 'foto.jpeg')
        pygame.camera.quit()
    button=QPushButton(window)
    button.setText('Time')
    button.move(0,10)
    #və ya button=QPushButton('start',window)
    button.setIcon(QIcon('icon.png'))
    button.clicked.connect(camera)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Fayl tipini png,bmp və s olaraq dəyişə bilərsiniz

əgər pygame modulu xətası alırsanızsa o zaman modul sistemdə yüklü deyil yükləmək üçün(Gnu/Linux) $ **sudo apt-get install python-pygame** yazaraq yükləyə bilərsiniz.

Pəncərəyə bir progresbar əlavə edək.Bunun üçün QProgressBar modulundan istifadə edəcəyik.

Kodlarımıza əlavələrimizi edək

```
def camera():
        pygame.camera.init()
        cam = pygame.camera.Camera(pygame.camera.list_cameras()[0])
        cam.start()
        img = cam.get_image()
        pygame.image.save(img, 'foto.jpeg')
        pygame.camera.quit()
        basla=0
        print'please wait....'
        while basla<100:
            basla += 0.0001
            progres.setValue(basla)
        print 'completed'
```

Düymənin çağırdığı camera funksiyasına bəzi əlavələrimizi etdik.

Bütünlükdə kodlarımız

```
# -*-coding: utf-8 -*-
import pygame.camera
import pygame.image
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QPushButton')
    window.setGeometry(400,300,300,300)
    progres=QProgressBar(window)
    progres.setGeometry(80,40,130,15)
    def camera():
        pygame.camera.init()
        cam = pygame.camera.Camera(pygame.camera.list_cameras()[0])
        cam.start()
        img = cam.get_image()
        pygame.image.save(img, 'foto.jpeg')
        pygame.camera.quit()
        basla=0
        print'please wait....'
        while basla<100:
            basla += 0.0001
            progres.setValue(basla)
        print 'completed'
    button=QPushButton(window)
    button.setText('Capture')
    button.move(100,80)
    #və ya button=QPushButton('start',window)
    button.setIcon(QIcon('icon.png'))
    button.clicked.connect(camera)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

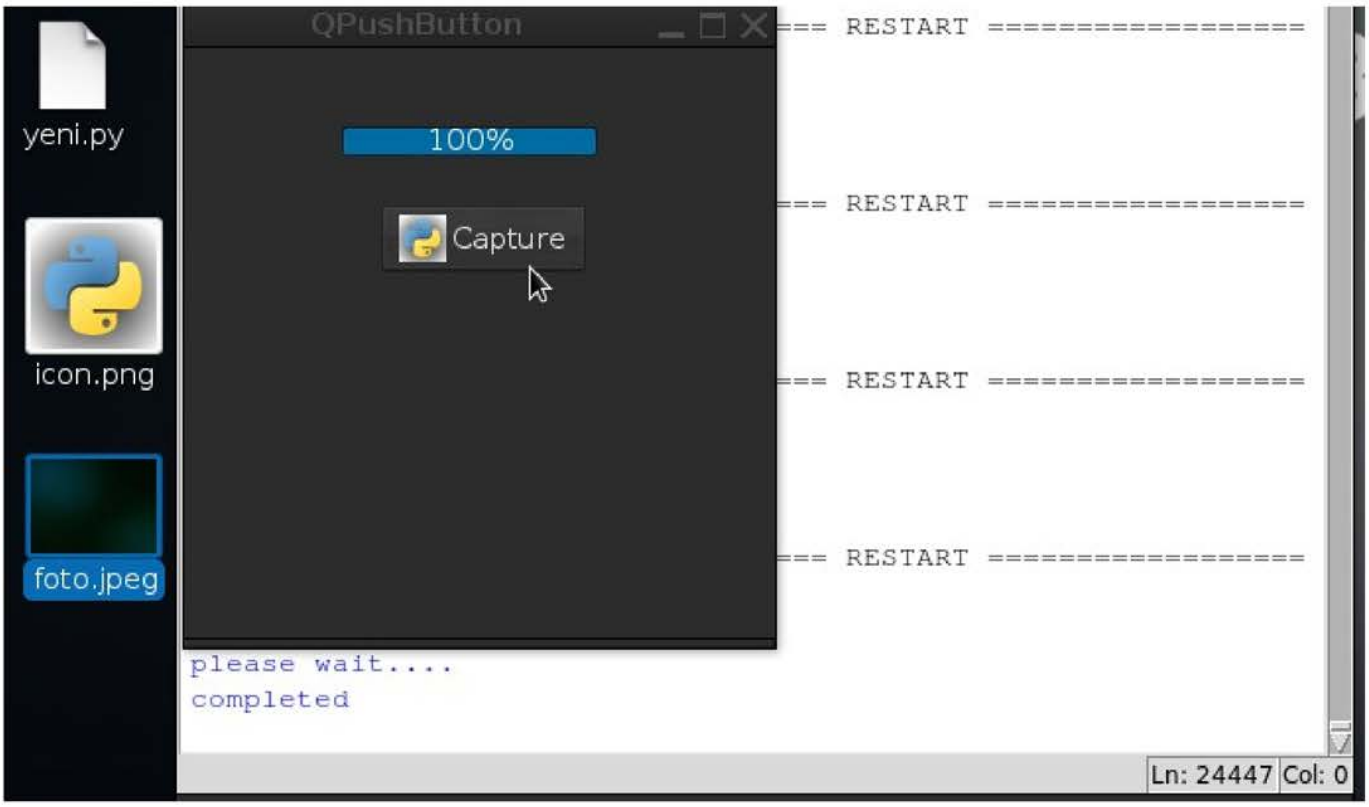
ProgressBar pəncərəyə tətbiq edirik

Başlanğıc 0% üzrə

Həcmini100% üzrə tənzimləyə bilərsiniz.

Vaxtı millis ilə tənzimləyə bilərsiniz

ekran görüntüsü



Kodlarımıza izahat verək.Deməli basla=0 yazdığımız ifadə progress qutusunun ədəd faizini tənzimləmək üçündür,15 -əgər yazsaq qutuda(progress) 15% -ifadəsini görəcəyik.while basla<100: -ifadəsi əgər basla ifadəsinə verdiyimiz say 100-dən kiçikdirsə basla üzərinə 100 ü əlavə et əmrini veririk.Yəni basla+100=basla(0)

0.0001 -sayı isə vaxtı tənzimləyir.Yəni 100 -ü millisaniyə ilə tənzimləyirik,əgər daha bir sıfr artırsaq progress qutusunun daha çox vaxta bittiyinin şahidi olacağıq.
Digər print funksiyaları isə nəyi ifadə etdiyi bizə məlumdur.

setCheckable metodu

Metod True və False parametrlərini alır.əgər True versək metod düymənin basıldığını təyin edib istifadəçi tərəfindən verilən funksiyanı yerinə yetirər.

Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QPushButton')
    window.setGeometry(400,300,300,300)
    def proces():
        if button.isChecked():
            print'button pressed'
            progres=QProgressBar(window)
            progres.move(120,5)
            progres.show()
        else:
            print'button default'
            progres=QProgressBar(window)
            progres.move(120,5)
            i=0
            while i<100:
                i+=0.00001
                progres.setValue(i)
            progres.show()
    button=QPushButton(window)
    button.setText('Home')
    button.setCheckable(True)
    button.move(0,30)
    button.clicked.connect(proces)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

setEnabled() metodu
Metod iki parametr alır.True və False.Düyməni aktiv halda tutmaq üçün metodu True parametri verilir.Passiv hal üçün isə False.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QPushButton')
    window.setGeometry(400,300,300,300)
    button=QPushButton(window)
    button.setText('Home')
    button.setEnabled(False)
    button.move(0,30)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

Düymə daxilində 'Drop down' menu

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
def pencere():
    app=QApplication(sys.argv)
    window=QWidget()
    window.setWindowTitle('QPushButton')
    window.setGeometry(400,300,300,300)
    button=QPushButton(window)
    button.setText(u'Daxil ol')
    button.move(0,30)
    def show():
        print'hello guys!'
        window.close()
    menu=QMenu()
```

Proqramdan çıxış üçün

```
    menu_1=menu.addAction('Open')
    menu_2=menu.addAction('Quit')
    menu_3=menu.addAction('Save')
    menu1=QMenu()
    menu2=menu1.addAction('Save as')
    menu_2.triggered.connect(show)
    menu_3.setMenu(menu1)
    button.setMenu(menu)
    window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    pencere()
```

show funksiyasını çağırmaq üçün

QMenu klasına(sinif) dərindən, klas bəhsində öyrənəcəyimiz üçün burada izah etməyə çalışmayacam.Sadəcə olaraq button(düymə) daxilində bu menuları tətbiq etmək eləcədə bunları digər funksiyalarla əlaqələndirib bir əməliyyatı yerinə yetirməyi tətbiq edə bilərik.

Siniflər(Class)

Bura qədər sadəcə olaraq kodlarımızı funksiya(def) daxilində yazırdıq.Bundan başqa pythondan bilirikki class-siniflər də mövcuddur.Və kodlarımızı siniflər daxilinə tətbiq edərək istifadəsini daha rahat şəkildə təmin edə bilərik.Siniflər daxilində kodların toplanması,bizə kodlarımızı həm rahat,həmdə dağınıqlığı aradan qaldırılmasına kömək edir.Ciddi bir fərq olmasada class daxilinə kodların toplanması daha məqsədəuyğundur.

İlk klas daxilinə kodlarımızı tətbiq edək
Bunun üçün biz class xxx() -kimi ifadə yazacağıq.Bundan başqa biz ana pəncərəni bir dəyişənə atıb və o dəyişənə metodları tətbiq edirdiksə klaslarda isə ardıq sadəcə self ifadəsini istifadə edərək yerinə yetirəcəyik.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self):
        self.gui()
    def gui(self):
        self.pencere=QWidget()
        self.pencere.setWindowTitle('hello')
        self.pencere.setWindowIcon(QIcon('icon.png'))
        self.pencere.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Yuxarıdakı kodlarımızda class daxilində -class window(QWidget) – ifadəsini yazdıq.Biz mötərizə daxilində Qwidget qeyd etməyimin səbəbi ümumi kod bloku Qwidget üzərində başladığını tənzimləyirəm.Və ya QDialog QMainWindow və s olaraq klasları əvəz edə bilərsiniz.Biz yuxarıda qeyd etdiyim klaslara keçmədiyim üçün hal-hazırda QVidget üçün bunu tətbiq etdim.Bir klas üzərində öyrənməyiniz,digərlərinin istifadəsinə zəmin yaradacaq.Bu metodun daha da alternativ variantı super funksiyası ilə tətbiqetmədir.super funksiyasının daxilində klas adını daxil etməklə həm klas adının təkrar-təkrar yazılmasının qarşısını alırıq,həm də ana pəncərəni bir sinif içərisində tutaraq qrafik hissəsini funksiyalardan(və ya digər siniflərdən) ayırmış oluruq.
Misal olaraq

```
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
```

Yuxarıdakı kodlarımızda   QWidget klasını çağırdıq və klası super-funksiyasına tətbiq edərək,self ifadəsi ilə əvəz etdik.Yəni

```
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setGeometry(700,300, 800, 800)
        self.setWindowTitle('Panda')
       self.setWindowIcon(QIcon('icon.png'))
```

yazaraq pəncərəni self ifadəsilə əvəz etdik və dəfələrlə pəncərəyə tətbiq edəcəyimiz dəyişkəni tətkrar etmədik.
Davam edərək ilk sinif daxilində proqramımızı yazaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.resize(400, 400)
        self.setWindowTitle('Class')
        self.setWindowIcon(QIcon('icon.png'))
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

kodlarımızda QWidget klasını sinif daxilə tətbiq etdik.Və klası bir dəyişənə deyil,sadəcə self ifadəsi ilə metodları tətbiq etdik.Növbəti klaslardan istifadə üçün əgər ana pəncərəyə tətbiq etsək,mötərizə daxilində sadəcə self(yəni Qwidget) yazacağıq.

Pəncərəyə button əlavə etsək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.resize(400, 400)
        self.setWindowTitle('Class')
        self.setWindowIcon(QIcon('icon.png'))
        self.button=QPushButton('Start',self)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Düymənin yerini ana pəncərədə olmasını təyin edirik

Və sadəcə self yazaraq düyməni ana pəncərəyə tətbiq etdik.

setToolTip() metodu

Metod,ifadələrə rəyin verilməsində istifadə olunur.İzahına çətinlik çəksəmdə aşağıdakı kodlarımızdan hər şey aydın olacaq
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.resize(400, 400)
        self.setWindowTitle('Class')
        self.setWindowIcon(QIcon('icon.png'))
        self.button=QPushButton('Start',self)
        self.button.setToolTip(u'Düyməyə sıxın')
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Mausu düyməyə yaxınlaşdırdıqda kiçik ölçülü məlumat pəncərəsi açıldı.
Html kodları ilə göstərmək istəsək
button.setToolTip(u' məlumat pəncərəsi ') kimi yaza bilərik.

Yazını dəyişmək üçün(QToolTip)

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
```

```
        self.resize(400, 400)
        self.setWindowTitle('Class')
        self.setWindowIcon(QIcon('icon.png'))
        self.button=QPushButton('Start',self)
        self.button.setToolTip(u' <b>məlumat pəncərəsi</b> ')
        self.button.setStyleSheet('QToolTip { font-size: 12pt; font-family:
Sansserif; }')
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Signal və slotlar

Signal və slotlar ayrı ayrılıqda bir funksiyadır.Signal və slotlara dair misalları bura qədər dəfələrlə kodlarımız daxilində istifadə etmişik.Model olaraq QtCore.QObject.connect(widget, QtCore.SIGNAL('signalname'), slot_function)

şəklindədir.Yuxarıda mötərizə daxilində ilk vidget -aid olduğu klas,SİGNAL adı(clicked,pressed,return_pressed və s) daha sonra çağırılan pyhon funksiyası və ya

button.clicked.connect(funksiya) kimi də yazılır.

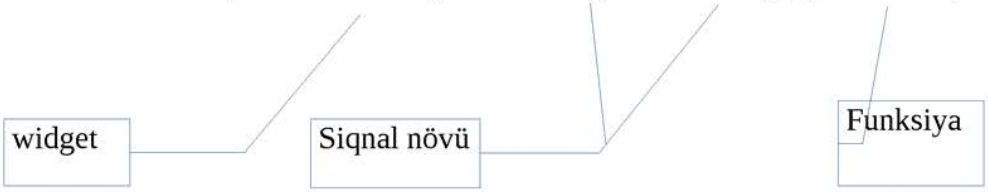
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('Class')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.show()
        self.klas()
```

```
    def klas(self):
        self.entry=QLineEdit(self)
        self.entry.setToolTip(u'Adınızı yazın')
        self.entry.move(20,10)
        self.entry.show()
        self.button=QPushButton('Click',self)
        self.button.move(20,40)
        self.button.clicked.connect(self.ad)
        self.button.show()
    def ad(self):
        print u'Adınız : '+unicode(self.entry.text())
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

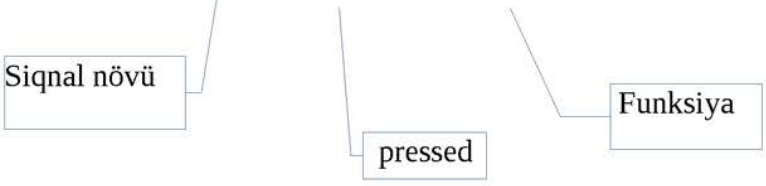
Kodlarımız arasında düşünürəm yenilik yoxdur.Çünki əvvəlki bəhslərdə button.clicked.connect(function) metodundan istifadə etmişik.Burada signal tipi clicked slot -isə function ifadələridir.Digər komanda metodu isə self.button.connect(self.button,SIGNAL('clicked()'), self.ad) yazaraq self.ad funksiyasını çağırırıq

```
self.button.connect(self.button,SIGNAL('clicked()'), self.ad)
```



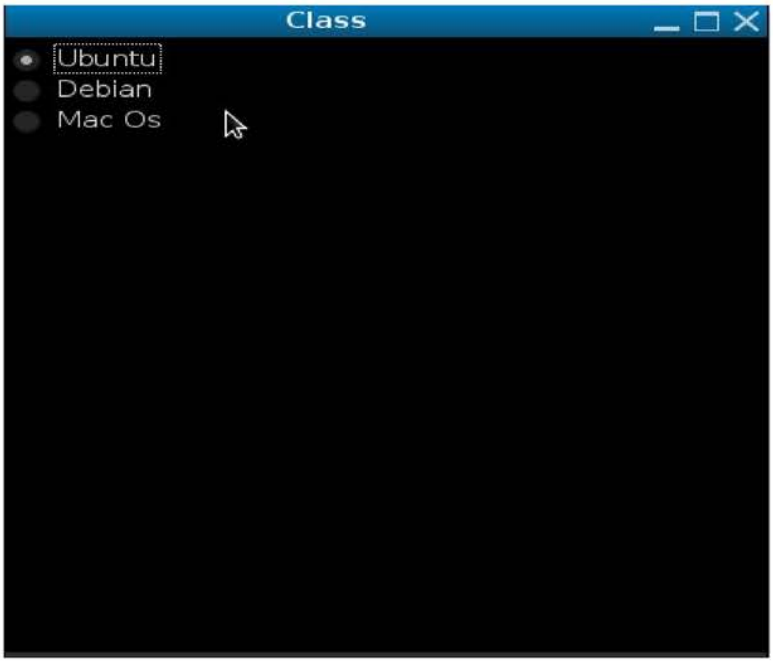
Siqnal növündə clicked əvəzinə pressed -də yazsaq xətasız çalışacaq.Və ya self.button.clicked.connect(self.ad) əvəzinə self.button.pressed.connect(self.ad) yaza bilərik.

```
self.button.clicked.connect(self.ad)
```



QRadioButton klası

Ekran görünüşü



radiobutonun daimi istifadə olunan metodlarını nəzər yetirək.

1.isChecked()

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('RadioButton')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.gray)
        self.setPalette(self.palette)
        self.radiobutton=QRadioButton('Ubuntu',self)
        self.radiobutton.move(2,2)
```

```
        self.radiobutton.clicked.connect(self.click)
        self.radiobutton1=QRadioButton('Debian',self)
        self.radiobutton1.move(2,22)
        self.radiobutton1.clicked.connect(self.click)
        self.radiobutton2=QRadioButton('Mac Os',self)
        self.radiobutton2.move(2,42)
        self.radiobutton2.clicked.connect(self.click)
        self.show()
    def click(self):
        if self.radiobutton.isChecked():
            print 'Ubuntu selected: Ubuntu'
        elif self.radiobutton1.isChecked():
            print 'Debian selected: Debian'
        elif self.radiobutton2.isChecked():
            print 'Mac Os selected: Mac Os'
        else:
            print 'please select radiobutton'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

click funksiyası daxilində isChecked metodundan istifadə edərək,istifadəçinin radiobutonlara uyğun olaraq seçdikdə print funksiyası vasitəsilə mesaj verdik.

text() metodu
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('RadioButton')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
```

```
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.gray)
        self.setPalette(self.palette)
        self.radiobutton=QRadioButton('Ubuntu',self)
        self.radiobutton.move(2,2)
        self.radiobutton.clicked.connect(self.click)
        self.radiobutton1=QRadioButton('Debian',self)
        self.radiobutton1.move(2,22)
        self.radiobutton1.clicked.connect(self.click)
        self.radiobutton2=QRadioButton('Mac Os',self)
        self.radiobutton2.move(2,42)
        self.radiobutton2.clicked.connect(self.click)
        self.show()
    def click(self):
        if self.radiobutton.isChecked():
            self.radiobutton.setText('Android')
            print self.radiobutton.text()
        elif self.radiobutton1.isChecked():
            print 'Debian selected: Debian'
        elif self.radiobutton2.isChecked():
            print 'Mac Os selected: Mac Os'
        else:
            print 'please select radiobutton'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

text metodu

Yuxarıda kodlarımız daxilində istifadəçinin Ubuntu seçməsi ilə,daha sonra radiobuttona verdiyimiz ifadənin dəyişilməsi,daha sonra bu dəyişilən ifadənin çap olunmasını təşkil etdik.

Radiobutton- un aktiv və passiv etmək üçün setDisabled metodundan istifadə edəcəyik.parametr olaraq metod True və False alır.Sadəcə dəyişiklik olan kodları bura yazıram

```
self.radiobutton.setDisabled(True)
```

ifadəni kodlarınız daxilinə əlavə edib nəticəni test edə bilərsiniz.

setChecked metodu

metod program açılışı zamanı düymənin seçili olmasını (və ya olmamasını) təmin edə bilərik.Parametr olaraq True və False alır.

Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('RadioButton')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.gray)
        self.setPalette(self.palette)
        self.radiobutton=QRadioButton('Ubuntu',self)
        self.radiobutton.move(2,2)
        self.radiobutton.clicked.connect(self.click)
        self.radiobutton1=QRadioButton('Debian',self)
        self.radiobutton1.move(2,22)
        self.radiobutton1.setChecked(True)
        self.radiobutton1.clicked.connect(self.click)
        self.show()
    def click(self):
        if self.radiobutton.isChecked():
            self.radiobutton.setText('Android')
            print self.radiobutton.text()
        elif self.radiobutton1.isChecked():
            print 'Debian selected: Debian'
        else:
            pass
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

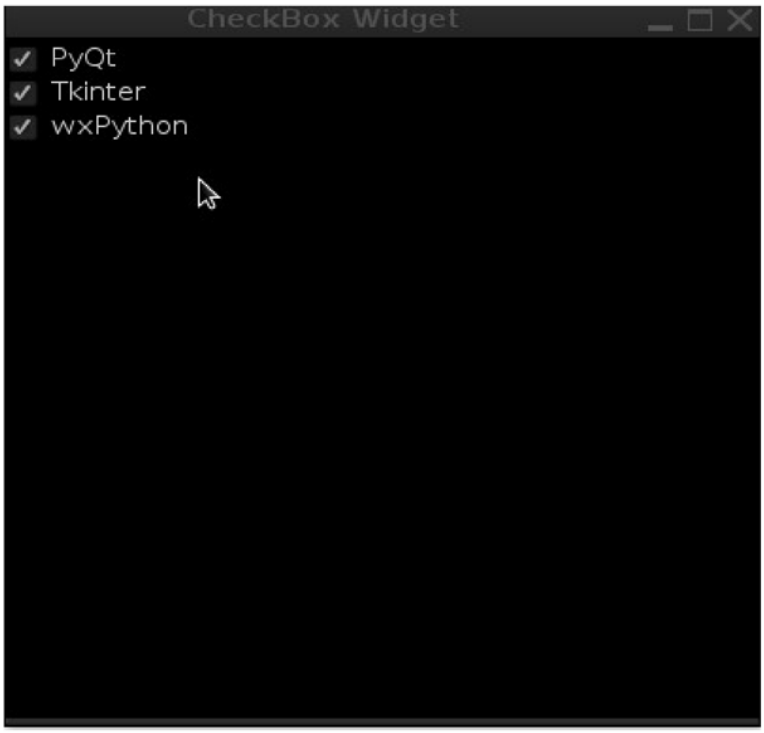
Programın açılışında self.radiobutton1.setChecked(True) ifadəsi sayəsində radiobutton1 -in seçili olduğunu gördük.

QCheckBox Widget

Klas radiobuttona bənzəyir

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('CheckBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.checkbox1=QCheckBox('Tkinter',self)
        self.checkbox1.move(0,20)
        self.checkbox2=QCheckBox('PyQt',self)
        self.checkbox2.move(0,0)
        self.checkbox3=QCheckBox('wxPython',self)
        self.checkbox3.move(0,40)
        self.checkbox3.setTristate(False)
        self.show()
    def i(self):
        print 'hello'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



klas üçün state metodları

checked və unchecked

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('CheckBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.checkbox=QCheckBox('Tkinter',self)
        self.checkbox.move(0,20)
```

```
        self.checkbox.stateChanged.connect(self.i)
        self.show()
    def i(self):
        if self.checkbox.text()=='Tkinter':
            if self.checkbox.isChecked()==True:
                print u'Siz qutunu seçdiniz'
            else:
                print 'qutu secili deyil'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

def i funksiyası daxilində istifadəçinin qutunu seçməklə 'Siz qutunu seçdiniz' mesajı,əksinə isə 'qutu secili deyil' mesajı verdik.İstifadə olunan stateChanged metodunu clicked və ya pressed metodları ilə dəyişə bilərsiniz.metodumuz sadəcə mausdan gələn məlumata cavab verir.əgər setChecked() metoduna True versək proqram açılışında hər hansı bir mesajın olmadığını görəcəyik,amma qutu seçili olacaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('CheckBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.checkbox=QCheckBox('Tkinter',self)
        #və ya self.checkbox.setText('Tkinter')
        self.checkbox.move(0,20)
        self.checkbox.clicked.connect(self.i)
        self.checkbox.setChecked(True)
        self.show()
    def i(self):
        if self.checkbox.text()=='Tkinter':
            if self.checkbox.isChecked()==True:
                print u'Siz qutunu seçdiniz'
```

Qutunun seçili olmasını təmin etdik

```
        else:
            print 'qutu secili deyil'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız arasında Demək olarki metodların bir neçəsini istifadə etdik

Bunlardan başqa setTristate metodu da varki qutunu istifadəsiz halda (passiv deyil) görünməsini təmin edir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('CheckBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.checkbox=QCheckBox('Tkinter',self)
        self.checkbox.move(0,20)
        self.checkbox.clicked.connect(self.i)
        self.checkbox.setChecked(True)
        self.checkbox.setTristate(True)
        self.show()
    def i(self):
      if self.checkbox.text()=='Tkinter':
        if self.checkbox.isChecked()==True:
            print u'Siz qutunu seçdiniz'
        else:
            print 'qutu secili deyil'
if __name__ == '__main__':
    app = QApplication([])
```

```
    gui = window()
    app.exec_()
```

QComboBox Widget

qutu daxilində ifadələri list şəklində görünməsini,eləcədə hər bir ifadəyə bir funksionallıq verilməsini təmin edir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.move(150,50)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız arasına klası əlavə etdik.Açılan pəncərədə qutunun boş olduğuna görə bəzi ifadələr əlavə edək.Bunun üçün klas,bizə iki yolla ifadə əlavə etməyi təklif edir.
addItem və addItems.Aralarında fərq isə biri sadəcə tək ifadəni,digəri list şəklində əlavə edir.Metodlar misallardan aydın olacaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItem('Python')
        self.combobox.addItems(['Java','C++','HTML5'])
        self.combobox.move(150,50)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

Kodlarımız arasında hər iki metoddan istifadə etdik.metodlar ardıcıllığa uyğun olaraq ifadələri qutuya əlavə etdi.

Bundan əlavə addİtem metodu listlərə aid olduğundan biz sorted funksiyasından istifadə edə bilərik.Funksiya ifadələri əlifba sırası ilə düzülüşünü təşkil edir.

self.combobox.addItems(sorted(['Rubby Rails','JQuerry','Java','C++','Avira']))

clear() metodu

Metod qutu daxilində ifadələrin bütünlükdə silinməsini təşkil edir.Pəncərəyə bir dümə əlavə edib və düyməni sıxdıqda qutu daxilində ifadələrin silinməsini(Clear() metodu vasitəsilə) tətbiq edək.Bunun üçün ayrıca bir funksiya daxilində kodlarımızı yazaq.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
1.  self.combobox.addItem('Python')
2.  self.combobox.addItems(['Java','C++','HTML5'])
3.  self.combobox.move(150,50)
4.  self.button=QPushButton('delete ',self)
5.  self.button.move(250,50)
```

```
6.      self.button.connect(self.button,SIGNAL('clicked()'),self.clean)

                    self.show()
    def clean(self):
        self.combobox.clear()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

1-ci və 2-ci sətridə addİtem və addİtems metodlarından istifadə edərək ifadələr əlavə etdik.6-cı sətirdə isə pəncərə daxilində olan düyməni def clean funksiyası ilə əlaqələndirdik.Funksiya daxilində isə clear() metdoundan istifadə edərək qutu daxilindəki(combobox) ifadələri sildik.

Digər metodları bərabər işlədib açıqlamalar verək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItem('Python')
        self.combobox.addItems(['Java','C++','HTML5','CSS'])
        self.combobox.move(150,50)
        self.combobox.currentIndexChanged.connect(self.choose)
        self.show()
    def choose(self):
```

```
        print u'Qutuda ümumi %s dil var.'%self.combobox.count()
        print u'Siz sıra nömrəsi {} olan {} dilini
seçdiniz.'.format(self.combobox.currentIndex(),self.combobox.currentText())
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Python shell- də görünüşü

```
>>>
Qutuda ümumi 5 dil var.
Siz sıra nömrəsi 3 olan HTML5 dilini seçdiniz.
Qutuda ümumi 5 dil var.
Siz sıra nömrəsi 4 olan CSS dilini seçdiniz.
Qutuda ümumi 5 dil var.
Siz sıra nömrəsi 0 olan Python dilini seçdiniz.
```

.....

self.combobox.currentIndexChanged.connect ifadəsi Sİgnal metodu sayılır.Qutu daxilindəki hər hansı bir ifadəyə sıxdıqda bizə def choose funksiyasını çağırır.
Ayrıca currentText metodu bizə seçdiyimiz ifadəni yalnızca göstərir.Digəri currentİndex isə sadəcə olaraq seçdiyimiz ifadənin sıra sayıdır.Python bir list daxilində ifadələri 0-dan başlayaraq saydığından sıralamada 0-ədədini də görəcəyik.count() metodu isə qutu daxilində ümumi ifadə sayını təyin edir.

Mövzuya dair daha bir metoddan istifadə edərək proqram yazaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
```

```
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItem('Python')
        self.combobox.addItems(['Java','C++','HTML5','CSS'])
        self.combobox.move(150,50)
        self.combobox.highlighted.connect(self.choose)
        self.show()
    def choose(self,index):
        if index == self.combobox.currentIndex():
            print 'Siz {} dilini seçdiniz'.format(self.combobox.currentText())
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız arasında yeni Signal metodu highlighted istifadə etdik.Metod,mausu ifadələrin üzərinə gətirdikdə İfadənin index nömrəsini tutur.Və bizdə həmin index nömrəsinə görə ifadənin sıra nömrəsi deyil currentText metodu ilə adını yazmağı əmr etdik.

Və ya sadəcə index nömrəsini çap edək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,400,400)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.black)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItem('Python')
        self.combobox.addItems(['Java','C++','HTML5','CSS'])
        self.combobox.move(150,50)
```

```
        self.combobox.highlighted.connect(self.choose)
        self.show()
    def choose(self,index):
        print 'sıra nömrəsi {} olan dili seçdiniz'.format(index)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Klasa dair daha bir proqram yazaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
from pyqterm import TerminalWidget
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,700,700)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.gray)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItem('Css')
        self.combobox.addItems(['Java','C++','HTML5','python'])
        self.combobox.move(150,50)
        self.combobox.currentIndexChanged.connect(self.choose)
        self.show()
    def choose(self):
        if  self.combobox.currentText()=='python':
            self.term=TerminalWidget(self)
            self.term.setGeometry(30,90,650,500)
            self.term.zoom_out()
            self.term.zoom_out()
            self.term.execute(command='python')
            self.term.show()
if __name__ == '__main__':
    app = QApplication([])
```

```
    gui = window()
    app.exec_()
```

Qutu daxilində əgər istifadəçi python ifadəsini seçərsə,ana pəncərə daxilində terminaldan python2-ni çağıracaq.Funksiya daxilində pyqterm modulundan istifadə etdim.İnternetdə araşdırdığım qədəriylə normal bir paket deyil,hətda normal çalışmır,metod və funksiyaları çox azdır.fronted faylına daxil olub özünüz də araşdıra bilərsiniz.Hər halda məqsədim sizə seçilmiş ifadə vastəsilə funksiyanın çağırılması idi.Bu arada pyqterm modulunu yükləmək üçün https://pypi.python.org/pypi/pyqterm ünvanından istifadə edə bilərsiniz.Modul yüklü olmasa təbiki yuxarıdakı proqram xəta verəcəkdir.Hal-hazırda 2 versiyası üzərindədir.

Css atributları ilə qutunun rənglərini tənzimləmək olar

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,700,700)
        self.palette=QPalette()
        self.palette.setColor(QPalette.Background,Qt.gray)
        self.setPalette(self.palette)
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItems(sorted(['Eclipse','C++','Rubby Rails','terminal']))
        self.combobox.setStyleSheet('QComboBox{background-color: darkgray;\
                                selection-background-color: gray;\
                                color: black;}')
        self.combobox.move(150,50)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
```

```
    app.exec_()
```

Kodlarımız arasına setStyleSheet əlavə etməklə qutunu css kodlama dili ilə bəzədik.
Css kodları ilə pəncərənin də arxa plan rəngini dəyişə bilərik

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('ComboBox Widget')
        self.setWindowIcon(QIcon('icon.png'))
```

Arxa plan rəngi.Və ya red,black,gray yazaraq da dəyişə bilərsiniz

```
            self.setGeometry(400,400,700,700)
        self.setStyleSheet("background-color:#1A2C3E;")
        self.label=QLabel('Combobox',self)
        self.label.move(50,50)
        self.combobox=QComboBox(self)
        self.combobox.addItems(sorted(['Eclipse','C++','Rubby Rails','terminal']))
        self.combobox.setStyleSheet('QComboBox{background-color: darkgray;\
                                            selection-background-color: gray;\
                                            color: black;}')
        self.combobox.move(150,50)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QBoxLayout Class -klası

Klas horizontal və vertikal istiqamətdə yerləşdirməni təşkil edir.klası tətbiq etmək üçün addWidget addStrech və addLayout metodlarından istifadə olunur

Aşağıda yazacağımız kodlarda iki düyməni vertikal istiqamətdə yerləşdirək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QBoxLayout class')
        self.setWindowIcon(QIcon('icon.png'))
        self.move(100,200)
        self.setStyleSheet("background-color:gray;")
        self.button=QPushButton('vbox',self)
        self.button1=QPushButton('vbox1',self)
        self.vbox=QVBoxLayout() # vertikal istiqamətdə V -ifadəsi
        self.vbox.addWidget(self.button)
        self.vbox.addWidget(self.button1)
        self.setLayout(self.vbox)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ilk əvvəl iki ədəd düymə təyin etdik,daha sonra vbox dəyişəninə QVBoxLayout () klasını tətbiq etdik.Qeyd edimki QboxLayout iki parametr alır V -vertikal və H-horizontal.əgər bu baş hərfləri qeyd etməzsəniz xəta alacaqsınız.Və sonra addWidget metodu ilə düymələri klasa tətbiq etdik.Son olaraq yazdığımız setLayout ifadəsi ilə ana pəncərəyə hər iki düyməni əlavə edirik.Proqramın açılışı zamanı gördüyümüz kimi iki ədəd düymə vertikal istiqamətdə yerləşdirilmişdir.Biz kodlarımızda biz yerləşdirmə metodlarından düymə üçün istifadə etmədik.Bu üsul tək düymə üçün deyil,QLabel ,QLineEdit və s istifadə edə bilərsiniz.Düymələri horizontal istiqamətdə düzmək üçün isə

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QBoxLayout class')
```

```
        self.setWindowIcon(QIcon('icon.png'))
        self.move(100,200)
        self.setStyleSheet("background-color:gray;")
        self.button=QPushButton('hbox',self)
        self.button1=QPushButton('hbox1',self)
        self.hbox=QHBoxLayout() # horizontal istiqamətdə H -ifadəsi
        self.hbox.addWidget(self.button)
        self.hbox.addWidget(self.button1)
        self.setLayout(self.hbox)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Bundan başqa addStretch() metodu da varki,düymələr arasında məsafə yaradır.Və bu məsafə,pəncərənin dinamik ölçüsünü dəyişdikdə düymələr arasında məsafələr də dəyişir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QBoxLayout class')
        self.setWindowIcon(QIcon('icon.png'))
        self.move(100,200)
        self.setStyleSheet("background-color:gray;")
        self.button=QPushButton('hbox',self)
        self.button1=QPushButton('hbox1',self)
        self.hbox=QHBoxLayout() # vertikal istiqamətdə V -ifadəsi
        self.hbox.addWidget(self.button)
        self.hbox.addStretch()
        self.hbox.addWidget(self.button1)
        self.setLayout(self.hbox)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

ilkin görünüş

ölçünün dəyişilməsi

Görüntüdən də aydın olurki pəncərə ölçüsünü dəyişdikdə addStrech metodu məsafəni qorumur.

QGridLayout Class -klası

klas sətir və sütunlarla verilənləri yerləşdirir.

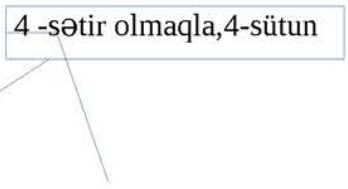
```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QBoxLayout class')
        self.setWindowIcon(QIcon('icon.png'))
        self.move(100,200)
        self.setStyleSheet("background-color:gray;")
        self.grid=QGridLayout(self)
        for i in range(3):
            for v in range(1,6):
                self.grid.addWidget(QPushButton(b),i,v)
        self.setLayout(self.grid)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız arasında for i in range(3) ifadəsi ilə düymələri 3-sətirdən ibarət olmasını,for v in range(1,6): ifadəsilə isə hər sətirdə 5 ədəd düymənin olmasını tənzimlədik.klasın ümumi modeli addWidget(QWidget, int r, int c) şəklindədir

int r -rowspan(sətir)
int c-columnspan(sıra(sütun))

və ya

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QBoxLayout class')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.grid=QGridLayout()
        self.setLayout(self.grid)
        list=[\
            '1','2','3','/','<-',
            '4','5','6','*','C',
            '7','8','9','+','i',
            '0','.','=','-','exit']
        post=[(i,j) for i in range(5) for j in range(5)]
        for show, number in zip(post, list):
            if number == ' ':
                continue
            self.button = QPushButton(number)
            self.grid.addWidget(self.button, *show)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```



kimi yazaraq hesab maşını üçün qrafik görünüşü hazırlaya bilərik.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QBoxLayout class')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.grid=QGridLayout()
        self.setLayout(self.grid)
        self.monitor=QLCDNumber()
        self.monitor.setDigitCount(30)
        self.monitor.setStyleSheet("background-color:white;")
        #self.line=QLineEdit()
        list=[\
            '1','2','3','/','<-',
            '4','5','6','*','C',
            '7','8','9','+','i',
            '0','.','=','-','Quit']
        post=[(i,j) for i in range(5) for j in range(5)]
        for show, number in zip(post, list):
            if number == '':
                continue
            self.grid.addWidget(self.monitor,0,5)
            self.button=QPushButton(number)
            self.grid.addWidget(self.button,*show)
            self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

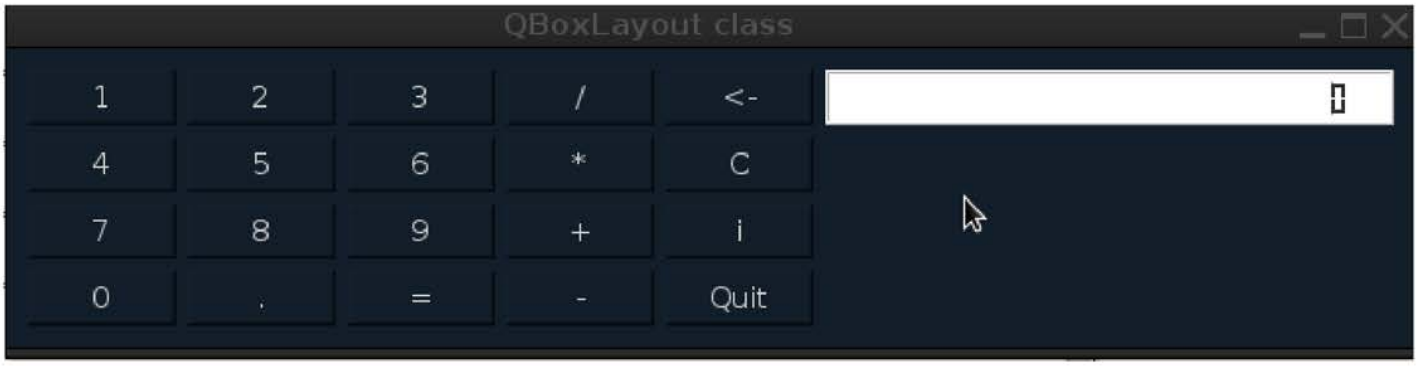
Rəqəm sayını 30-sayda tənzimləyirik

LCD monitorun 0-cı sətirdən 5-ci sırada yer alır

QLCDNumber əvəzinə QLineEdit klasını da istifadə edə bilərdik.Və LCD monitoru 30 digit sayda təmin etdik.Arxa plan rəngini isə white-ağ olaraq dəyişdik.QGridLayout- addWidget metodundan gördüyümüz kimi sıra və sütunlarda verilənləri yerləşdirir.Buttonların yerini isə dəyişə bilmərik,çünki range funksiyası daxilində artıq sətir və sütunları qeyd etmişik.və Başlanğıc

kimi klas,verilənləri 0-dan tənzimləyir.

Ekran görünüşü



Hesab maşını üçün digər alternativ yola baxaq
Bunun üçün kodlarımızı yazaq

```
# -*-coding: utf-8 -*-
from __future__ import division
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window  setting-------------------------
        self.setWindowTitle('QBoxLayout class')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.grid=QGridLayout()
        self.setLayout(self.grid)
        self.line=QLineEdit()
        self.line.setMaxLength(17)
        self.line.setStyleSheet("background-color:darkgray;")
        self.line.setAlignment(Qt.AlignRight)
        #-----------------------------------------------------
```

Yuxarıdakı kodlarımızda pəncərənin görünüşü,eləcədə başlıq və arxa plan

rəngini dəyişdik.Kodlarımız arasında bizə yad olan yeni metod və funksiya yoxdur.İndidə pəncərəyə bəzi ehtiyac düymələri tətbiq edək

```
        button_plus=QPushButton('+')
        button_minus=QPushButton('-')
        button_mult=QPushButton('*')
        button_div=QPushButton('/')
        button_equal=QPushButton('=')
        button_equal.clicked.connect(self.result)
        button_clear=QPushButton('CE')
        button_clear.clicked.connect(self.clean)
        button_backsps=QPushButton('Backspace')
        button_backsps.setStyleSheet("background-color:black;")
        button_backsps.clicked.connect(self.back)
        button_exit=QPushButton('Quit')
        button_exit.clicked.connect(self.quiting)
```

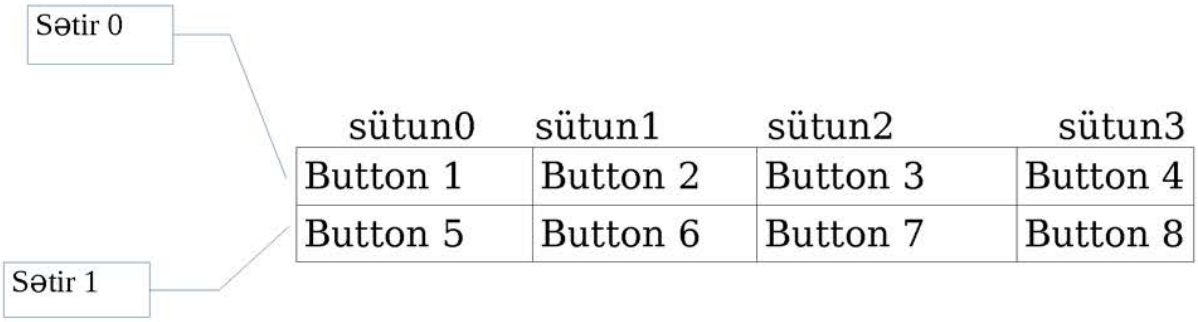
Düymələri,funksiyalarına görə adlandırdıq.Daha sonra plus minus vurma bölmə işarələrini bir list daxilinə tətbiq edib,addWidget metodu vasitəsilə pəncərəyə yerləşdirək(sıra və sütun üzrə)

```
list_operator=[\
            button_plus,button_minus,
            button_mult,button_div,]
        for i in list_operator:
            i.clicked.connect(self.opert)

        list=[\
            '1','2','3',
            '4','5','6',
            '7','8','9',
            '0','.']
        post=[(i,v) for i in range(5) for v in range(5)]
        for show, number in zip(post, list):
            if number == '':
                continue
            self.grid.addWidget(self.line,0,5)
            self.button=QPushButton(number)
            self.button.setStyleSheet("background-color:#1A2938;")
            self.button.clicked.connect(self.operation)
            self.grid.addWidget(self.button,*show)
            self.grid.addWidget(button_plus,2,1)
            self.grid.addWidget(button_minus,2,2)
```

```
        self.grid.addWidget(button_mult,2,3)
        self.grid.addWidget(button_div,2,4)
        self.grid.addWidget(button_equal,3,4)
        self.grid.addWidget(button_clear,1,5)
        self.grid.addWidget(button_backsps,2,5)
        self.grid.addWidget(button_exit,3,5)
      self.show()
```

self.grid.addWidget(button_plus,2,1) ifadəsi daxilində yazdığımız 2 və 1 ədədləri sıra və sütunları göstərir.model olaraq addWidget(widget,int row,int column) formasındadırYəni 2,ikinci sətir 1-isə birinci sütunu təmsil edir.Daha aydın şəkildə
cədvəllə ifadə edək.



sütun0	sütun1	sütun2	sütun3
Button 1	Button 2	Button 3	Button 4
Button 5	Button 6	Button 7	Button 8

Proqramın ekran görünüşü aşağıdakı şəkildədir

Aşağıdakı kodlarımızda isə hər düyməyə sıxanda ekranda görünməsini təşkil etdik

Bunun üçün sender metodundan istifadə edəcəyik.Sender metodu düymənin

sıxıldığında,düyməyə verdiyiniz ifadəni oxuyur.Biz self.button.clicked.connect(self.operation) ifadəsini yazmaqla operation funksiyasını çağırırır.İndidə baxaq bu operation funksiyası nəyi yerinə yetirir.

```
def operation(self):
        sender=self.sender()
        if False==False:
            self.line.setText(self.line.text()+sender.text())
```

kodlarımız daxilində sender metodunu global verilən kimi self.sender() şəklində yazdıq.Daha sonra monitorun mətn görünüşünü sender metodunun text() parametrindən istifadə edərək üzərinə əvvəlki ifadəni gəlməklə tətbiq etdik.Çünki biz self.line.text ifadəsini istifadə etməsəydik.hər dəfə düymələrə sıxdıqda özündən əvvəlki ifadələr ayrı formada qalacaqdır.
Və nəticədə result funksiyası daxilinə pythonun eval funksiyasını tətbiq edirik.Ən başda bildiyimiz kimi from __future__ import division metodunu istifadə etməklə istifadəçiyə dəqiq cavabları verməyi təmin edirik.

```
def result(self):
        try:
            i=self.line.text()
            self.line.clear()
            self.line.setText(str(eval(str(i))))
        except NameError:
            self.line.clear()
            self.line.setText('please write only integer')
        except SyntaxError:
            self.line.clear()

        else:
            self.line.setText(sender.text())
```

Və kodları xətalarla birgə istifadə etmək üçün try blokuna alaraq NameError SyntaxError kimi xətalardan qaçırıq.Bu arada qeyd edimki funksiyaları self.show() metodundan sonra yazın əks halda xəta alacaqsınız.Və aşağıdakı funksiyaların da nəyi təyin etdiyi bizə qaranlıq qalmayacaq

```
    def opert(self):
        sender=self.sender()
        i=sender.text()
        if False==False:
```

```
            self.line.setText(self.line.text()+i)
```

opert funksiyasının çalışması üçün list_operator=[\
button_plus,button_minus,
button_mult,button_div,]
daxilindəki ifadələri bir i dəyişəninə atıb `for i in list_operator:`
`i.clicked.connect(self.opert)`
düymə kimi fəaliyyətini tənzimləyirik.

```
    def clean(self):
        self.line.clear()
    def back(self):
        self.line.backspace()
    def quiting(self):
        self.close()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Monitoru tam silmək üçün

Ifadələri tək tək silmək üçün

Bu arada xəta bloklarını artıra bilərsiniz.(ZeroDivisionError və s)Eləcədə pəncərədə görünən boşluqlara əlavə düymələr əlavə edə bilərsiniz.Bütövlükdə kodları aşağıdakı ünvandan
https://pastebin.ubuntu.com/23910108/
Və ya github hesabıma yerləşdiriyim hesab maşınını yükləyib istifadə edə,eləcədə kodlara nəzər yetirə bilərsiniz.
https://github.com/RashadGarayev/pyq-calculator

QFormLayout Class -1

```
# -*-coding: utf-8 -*-
from __future__ import division
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
```

```
        #window setting
        self.setWindowTitle('QForm')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.label=QLabel('Name')
        self.line=QLineEdit()
        self.form=QFormLayout(self)
        self.form.addRow(self.label,self.line)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

Kodlarımız daxilində Qlabel və QlineEdit klaslarını dəyişənə atıb təyin etdik.Daha sonra QForm klasını dəyişənə qeyd edib,addRow metodundan istifadə edərək self.line və self.label -i pəncərəyə tətbiq etdik.Biz yerləşdirilməsinə müdaxilə etmədən klas verilənləri pəncərəyə ardıcıllıqla yerləşdirdi.əgər vertikal istiqamətdə,alt-alta tətbiq etmək üçün QVBoxLayout klasını istifadə etməliyik.
Misal üçün

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting----------------------------
        self.setWindowTitle('QForm')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.label=QLabel('Name')
        self.line=QLineEdit()
        self.form=QFormLayout(self)
        self.form.addRow(self.label,self.line)
```

```
        self.vbox=QVBoxLayout(self)
        self.vbox.addWidget(self.label)
        self.vbox.addWidget(self.line)
        self.form.addRow(self.vbox)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

Qeyd edimki ifadələri pəncərəyə yerləşdirilməsində ardıcıllığa fikir verin.Biz ilk öncə self.label daha sonra self.line yazdığımız üçün qanunauyğunluqla verilənlər pəncərəyə tətbiq olundu.
Və ya QVBoxLayout daxilinə tətbiq edib daha sonra QFormLayout klası vastəsilə yerləşdirək

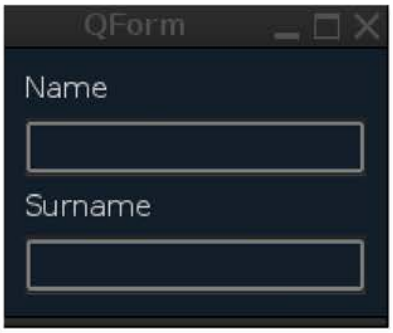
```
# -*-coding: utf-8 -*-
from __future__ import division
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QForm')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.label=QLabel('Name')
        self.line=QLineEdit()
        self.form=QFormLayout(self)
        self.vbox=QVBoxLayout(self)
        self.vbox.addWidget(self.label)
        self.vbox.addWidget(self.line)
        self.form.addRow(self.vbox)
```

```
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Və form üçün növbəti kodlarımızı yazaq

```
# -*-coding: utf-8 -*-
from __future__ import division
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QForm')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:#121E2A;")
        self.label=QLabel('Name')
        self.label1=QLabel('Surname')
        self.line=QLineEdit()
        self.line1=QLineEdit()
        self.form=QFormLayout(self)
        self.vbox=QVBoxLayout(self)
        self.tobox=QVBoxLayout(self)
        self.vbox.addWidget(self.label)
        self.vbox.addWidget(self.line)
        self.tobox.addWidget(self.label1)
        self.tobox.addWidget(self.line1)
        self.form.addRow(self.vbox)
        self.form.addRow(self.tobox)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

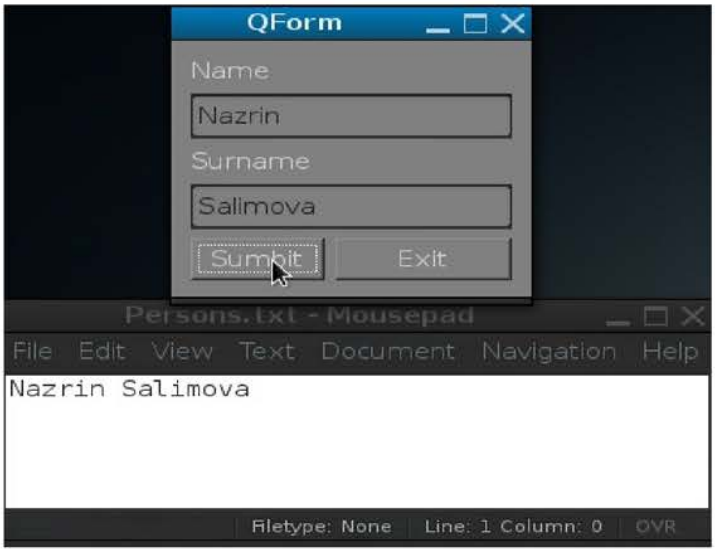


Bildiyimiz kimi addRow metodu soldan sağa doğru sətir üzrə veriləni yerləşdirdiyi üçün düymə tətbiq etsək ardıcıl olaraq bizə əlverişli üsulda yerləşdirəcəkdir.Və move metodunu istifadə etməyəcəyik.

```
# -*-coding: utf-8 -*-
from __future__ import division
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QForm')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
        self.label=QLabel('Name')
        self.label1=QLabel('Surname')
        self.line=QLineEdit()
        self.line1=QLineEdit()
        self.button=QPushButton('Sumbit')
        self.button1=QPushButton('Exit')
        self.button.clicked.connect(self.ok)
        self.button1.clicked.connect(self.cancel)
        self.form=QFormLayout(self)
        self.vbox=QVBoxLayout(self)
        self.tobox=QVBoxLayout(self)
```

```
        self.vbox.addWidget(self.label)
        self.vbox.addWidget(self.line)
        self.tobox.addWidget(self.label1)
        self.tobox.addWidget(self.line1)
        self.form.addRow(self.vbox)
        self.form.addRow(self.tobox)
        self.form.addRow(self.button,self.button1)
        self.show()
    def ok(self):
        f=open('Persons.txt','aw+')
        f.write(self.line.text()+' '+self.line1.text())
        f.close()
        print'Name of the file:',f.name
        print 'Closed or not:',f.closed
        print'Opening mode:',f.mode
        print 'Softspace flag :',f.softspace
    def cancel(self):
        self.close()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

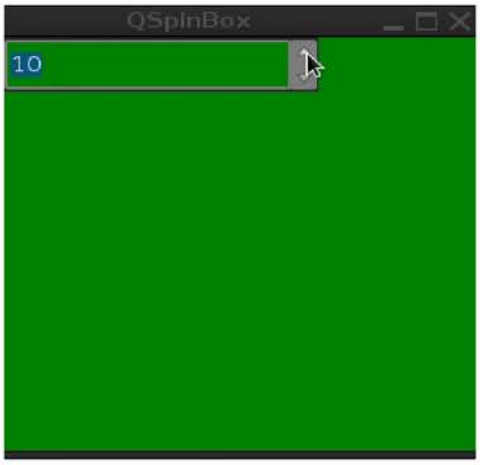
Ekran görünüşü




```
>>>
Name of the file: Persons.txt
Closed or not: True
Opening mode: aw+
Softspace flag : 0
>>>
```

QSpinBox Widget

Widget,qutu daxilində ifadənin görünüşünü təmin edir.




```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSpinBox')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:green;")
        self.setGeometry(400,400,300,300)
        self.spinbox=QSpinBox(self)
        self.spinbox.resize(200,40)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımızda sadəcə olaraq spinbox-u ana pəncərəyə tətbiq etdik.Başlanğıc etibarı ilə qutu 0-dəyərindən başlayır.əgər minimum dəyər tətbiq etmək istəsək,setMinimum(int) metodunu istifadə edəcəyik.Bundan başqa qutunu maksimum dəyərlə də təmin etmək olur.Bu zaman setMaximum(int) metodunu istifadə edəcəyik.

Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSpinBox')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:green;")
        self.setGeometry(400,400,300,300)
        self.spinbox=QSpinBox(self)
        self.spinbox.resize(200,40)
        self.spinbox.setMinimum(-10)
        self.spinbox.setMaximum(30)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımızda metodlardan istifadə edərək,self.spinbox.setMinimum(-10) ifadəsilə minimal dəyəri -10, self.spinbox.setMaximum(30) ifadəsilə isə maksimal dəyəri 30 olaraq qeyd etdik.

setRange(int,int) metodu

Metod,minimal və maksimal dəyərləri mötərizə daxilində bərabər qəbul edir.Yəni yuxarıdakı metodların əvəzinə birbaşa bu metodu istifadə edə bilərik.

```
self.spinbox.setRange(-5,10)
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
```

```
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting----------------------------
        self.setWindowTitle('QSpinBox')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:green;")
        self.setGeometry(400,400,300,300)
        self.spinbox=QSpinBox(self)
        self.spinbox.resize(200,40)
        self.spinbox.setRange(-5,10)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

self.spinbox.setRange(-5,10) ifadəsilə minimal və maksimal dəyərləri qeyd etdik

Digər metodlara nəzər yetirək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSpinBox')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
        self.setGeometry(400,400,400,400)
        self.spinbox=QSpinBox()
        self.label=QLabel('Spinbox value:')
        self.spinbox.resize(100,25)
        self.spinbox.setRange(1994,2009)
        self.spinbox.setValue(200)
        self.grid=QGridLayout(self)
        self.grid.addWidget(self.spinbox)
        self.grid.addWidget(self.label)
        self.spinbox.valueChanged.connect(self.create)
        self.show()
```

```
    def create(self):
        self.label.setText('Value:'+str(self.spinbox.value()))
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

self.spinbox.valueChanged.connect(self.create) ifadəsində valueChanged -signaldır.Və self.create funksiyasını çağıraraq istifadəçisinin qutudakı verilənləri dəyişməklə pəncərədə görünməsini təşkil etdik.value() ifadəsi isə digər klaslarda text() metodunu əvəz edir.Yəni qutu daxilində seçdiyiniz tam ədədi oxuyur.

Klasa dair daha bir proqram yazaq.

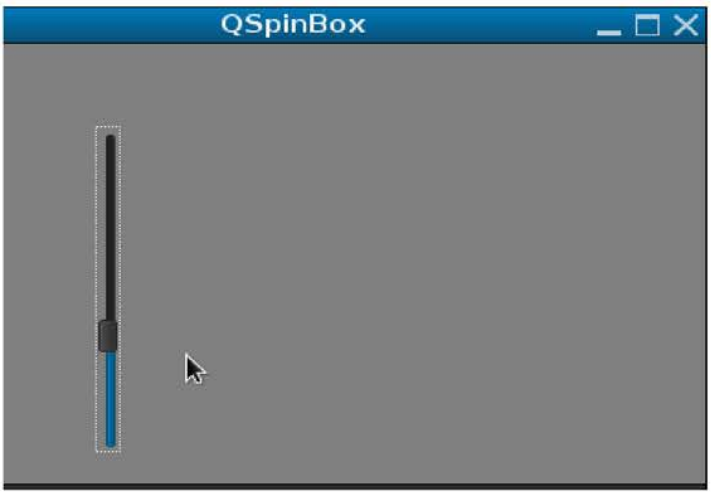
```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSpinBox')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
        self.setGeometry(400,400,400,400)
        self.lcd=QLCDNumber()
        self.lcd1=QLCDNumber()
        self.spinbox=QSpinBox()
        self.spinbox1=QSpinBox()
        self.grid=QGridLayout(self)
        self.grid.addWidget(self.lcd,0,0)
        self.grid.addWidget(self.lcd1,0,1)
        self.grid.addWidget(self.spinbox,1,0)
        self.grid.addWidget(self.spinbox1,1,1)
        self.connect(self.spinbox,SIGNAL('valueChanged(int)'),self.change)
        self.connect(self.spinbox1,SIGNAL('valueChanged(int)'),self.change1)
        self.show()
    def change(self):
        if self.spinbox.value() !=self.spinbox1.value():
            self.spinbox1.setValue(self.spinbox.value())
        self.lcd.display(self.spinbox.value())
```

```
    def change1(self):
        if self.spinbox1.value() !=self.spinbox.value():
            self.spinbox.setValue(self.spinbox1.value())
        self.lcd1.display(self.spinbox1.value())
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QSlider – Widget

vidjet,pəncərədə həcmin dəyişilməsini təmin edir. QSlider kənar verilənlərə müdaxilə edir.vidjetin modeli

self.sp=QSlider(Qt.Horizontal)
self.sp=QSlider(Qt.Vertical) şəklindədir.Həm vertikal ,həm də horizontal şəkildə yerləşə bilir.Digər klaslar kimi metod və funksiyalara malikdir.Aşağıdakı pəncərəmizdə QSlider-in bəsit görünüşü verilmişdir.




```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
```

```
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSlider')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
        self.setGeometry(400,400,400,400)
        self.slider=QSlider(self)
        self.slider.setGeometry(50,50,20,200)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

width=20
height=200

slider -i pəncərəyə tətbiq etdikdə bir qayda olaraq vertical formada yerləşir.setGeometry metodu daxilindəki 20 və 200 ifadələri slider-in width və height ölçüləridir.Biz vertical və horizontal formada tətbiq etmək istəsək sadəcə olaraq self.slider=QSlider(Qt.Horizontal,self) və self.slider=QSlider(Qt.Vertical,self) formada yazacağıq.Digər üsul isə QboxLayout klası vasitəsilə horizontal və vertical tətbiq etməkdir.
setMininum(int)
setMaximum(int) metodları

Vidget həcmini tam ədədlər olmaq şərtilə limitləndirə bilərik.Yəni self.slider.setMaximum(100)

spinbox-da istifadə etdiyimiz setRange(int,int) metodunu tətbiq edə bilərik.Bu zaman setMinimum və setMaximum metodlarına ehtiyyac olmayacaq.

Proqram açılışında vidgetin seçili olduğunu görürük,bunu aradan qaldırmaq üçün setFocusPolicy(Qt.NoFocus) metodundan istifadə edəcəyik.

```
 self.slider.setFocusPolicy(Qt.NoFocus)
```

setTickPosition()   metodu

Metod slider-i şkalalara bölür.Bu şkalaları metoda parametr verərək yerləşməsini təmin edə bilərik.Metodun ala bildiyi parametrlər

QSlider.NoTicks -şkalanın görünməməsi üçün
QSlider.TicksBothSides -şkalanı hər iki tərəfə tətbiq etmək üçün

QSlider.TicksBelow -şkalanı aşağı hissəyə tətbiq etmək üçün(horizontal)
QSlider.TicksAbove- şkalanı yuxarı hissəyə tətbiq etmək üçün(Horizontal)
QSlider.TicksLeft -şkalanı sol hissəyə tətbiqi
QSlider.TicksRight -şkalanı sağ hissəyə tətbiq etmək üçün
İstifadə modeli slider.setTickPosition(QSlider.NoTicks) formasındadır.

Vidgetin Signal və slotları

valueChanged()
sliderPressed() Signal-pressed
sliderMoved()
sliderReleased()

Yuxarıdakı şkalaları bölgü üzrə həcmini setTickInterval(int) metodu ilə tənzimləyirik.Yəni abstrak düşünərək 0-300 aralığını 100-şkalalı bir bölgü ilə tənzimləyə bilərik.

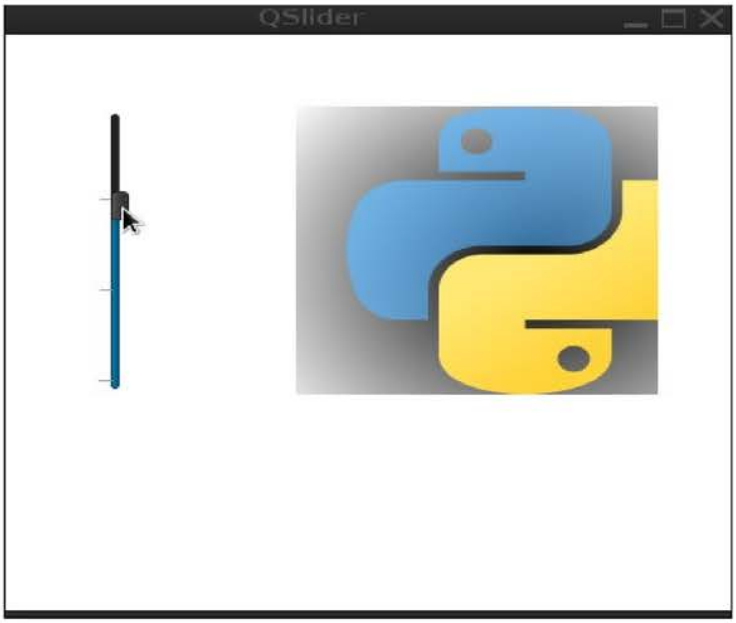
Misallara baxaq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSlider')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:white;")
        self.setGeometry(400,400,400,400)
        self.label=QLabel(self)
        self.label.setPixmap(QPixmap('icon.png'))
        self.label.setGeometry(160,50,80,50)
        self.slider=QSlider(self)
        self.slider.setGeometry(50,50,20,200)
        self.slider.setRange(0,200)
        self.slider.setTickPosition(QSlider.TicksLeft)
        self.slider.setFocusPolicy(Qt.NoFocus)
        self.slider.setTickInterval(70)
        self.slider.valueChanged.connect(self.change)
        self.show()
```

```
    def change(self,value):
        if value==0:
            self.label.setGeometry(160,50,80,50)
        elif value>0 and value<=100:
            self.label.setGeometry(160,50,100,100)
        elif value>=100 and value<=200:
            self.label.setGeometry(160,50,200,200)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız arasında self.slider.setRange(0,200) ifadəsi sliderin ümumi həcmini minimal və maksimal olaraq 0-200 aralığında qeyd etdik.Daha sonra slider oxuna şkala tətbiq etmək üçün self.slider.setTickPosition(QSlider.TicksLeft) ifadəsini yazdıq,və sol tərəfə yönəltdik(Qslider.TicksLeft).Və proqramın açılışı zamanı slider slider oxunun seçimini aradan qaldırdıq.self.slider.setFocusPolicy(Qt.NoFocus) ifadəsi ilə.şkala intervalını isə self.slider.setTickInterval(70) ifadəsilə 70 tam ədədi verməklə 0-200 aralığını 3 yerə böldük.Və self.slider.valueChanged.connect(self.change) signalı ilə def change funksiyasını çağırdıq.
Funksiya daxilində aralıqları if və elif operatoru vasitəsilə təyin edib labelin-ölçüsünü dəyişdik.

Ekran görünüşü



```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSlider')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
        self.setGeometry(400,400,400,400)
        self.label=QLabel('QSlider',self)
        self.label.move(150,20)
        self.slider=QSlider(Qt.Horizontal,self)
        self.slider.move(150,60)
        self.slider.setRange(0,30)
        self.slider.setTickPosition(QSlider.TicksLeft)
        self.slider.setTickInterval(5)
        self.slider.setValue(20)
        self.slider.valueChanged.connect(self.change)
        self.show()
    def change(self):
            self.label.setFont(QFont('Italic',self.slider.value()))
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()


# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSlider')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
```

```
        self.setGeometry(400,400,400,400)
        self.label=QLabel('QSlider',self)
        self.label.move(150,20)
        self.slider=QSlider(Qt.Horizontal,self)
        self.slider.move(150,60)
        self.slider.setRange(0,30)
        self.slider.setTickPosition(QSlider.TicksLeft)
        self.slider.setTickInterval(5)
        self.slider.setValue(20)
        self.slider.sliderMoved.connect(self.change)
        self.slider.sliderPressed.connect(self.change1)
        self.show()
    def change(self):
            print 'Moved'
    def change1(self):
            print 'Pressed'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

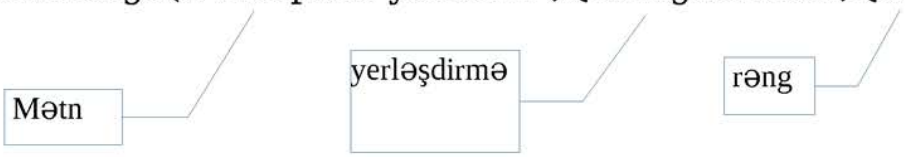
Proqram başlıqları
Tez-tez rast gəldiyimiz proqram açılışları əvvəlində loading və ya fotoların,videoların gəlməsini tənzimləmə üçün QSplash klasından istifadə edəcəyik.İlk öncə metodlarına nəzər salaq

finish() açılış pəncərəsinin bağlanmasını təmin edir.
show() klasa daxil olan bütün verilənlərin göstərilməsi
showMessage('mesaj və ya mətn','koordinasiya,yerləşdiriləsi','yazı rəngi')
metod açılışa tətbiq etdiyimiz foto üzərinə yazı əlavə etmək üçün istifadə olunur.
Yerləşdirmə üçün ala bildiyi parametrlər
1.Qt.AlignCenter
2.Qt.AlignBottom
3.Qt.AlignHCenter (Horizontal istiqamətdə mərkəzdə)
4.Qt.AlignVCenter (Vertical istiqamətdə mərkəzdə)
5.Qt.AlignLeft
6.Qt.AlignRight

yazı rəngi isə aşağıdakı qaydada tənzimlənir

Qt.red (və ya black,green,blue,gray və s)
QColor istifadə etdikdə isə QColor('red') və ya QColor("#ff0000") formasındadır
Ümumi modeli

self.splash.showMessage(u'Proqram yüklənir',Qt.AlignCenter,Qt.black)



İnternet üzərindən istənilən ölçüdə foto yükləyək daha sonra kodlarımızı yazıb skripti çalışdıraq

İlk əvvək modulları,ana pəncərəni ölçüsünü arxa plan rəngini hazırlayırıq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QSplash')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:black;")
        self.setGeometry(400,400,400,400)
```

İstəsəniz pəncərəyə düymə label və s əlavə edə bilərsiniz.məqsədim proqram açılışını hazırlamaq olduğundan pəncərəyə əlavələr məsələsinə girməyəcəm.

Daha sonra QSplash klasını tərtib edirik

```
        self.splash=QSplashScreen(QPixmap('icon.jpg')
        self.splash.show()
        self.loading(self.splash)
        self.splash.finish(self)
        self.show()
```

Kodlarımız daxilinə splash klasına bir jpg faylı əlavə edirik.Daha sonra Bu faylın görünməsini splash.show() ifadəsi ilə təmin edirik.Və ana pəncərənin loading funksiyasını çalışdırmaq əmri veririk.self.loading(self.splash)
Daha sonra funksiyanın bitməsi ilə çıxış verib əsas pəncərəyə keçid edirik.

Yuxarıda qeyd etdiyimiz loading funksiyasını yazaq

```
 def loading(self,s):
            for i in range(1,11):
                time.sleep(2)
                s.showMessage(u'Proqram yüklənir,zəhmət olmasa gözləyin...{}
%'.format(i*10),Qt.AlignCenter,Qt.green)
```

Funksiya daxilində 1,11-yəni 10 vahidlik ədəd təyin edirik.icon.jpg faylın görünməsini 2 saniyə ilə tənzimləyirik.Daha sonra icon foto üzərinə mesaj yazırıq.format (i*10) ifadəsi ilə 10 vahidlik ədəddi vururuq.Yəni format (10*10) və yazını mərkəzə yerləşdiririk,rəngini də yaşıl olaraq qeyd edirik.Sonda isə

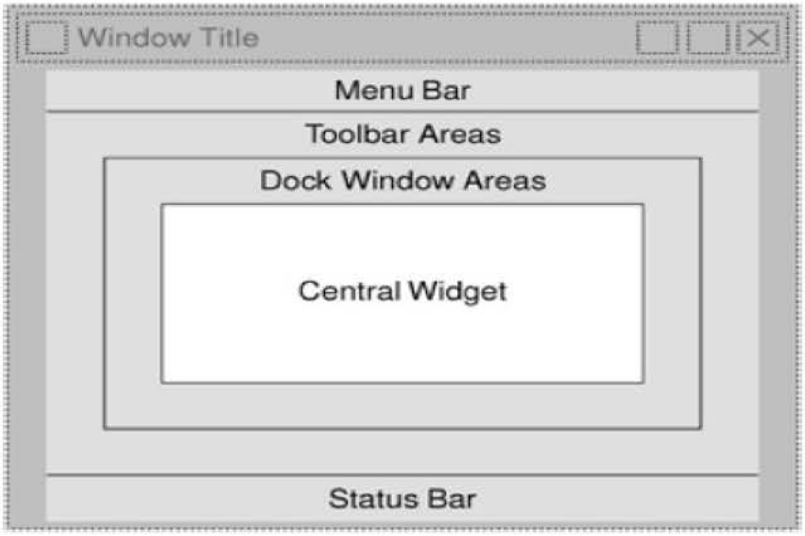
```
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodları yazıb skripti hazır vəziyyətə gətiririk

Kodları,bütövlükdə https://github.com/RashadGarayev/PyQt.QSplash ünvanından əldə edə bilərsiniz.

QMenuBar, QMenu Widget

Kompyuterdə tezt-tez rastlaşdığımız menular,əsasən proqram daxilində yuxarı hissədə yerləşir.File,Edit,View və bu menuların da alt menuları mövcuddur.Bu menular title bar altında toolbar arasında yerləşir.QMenu klasın metodlarıdır.Və QMenu bir vidgetdir.Bunlarda əsas pəncərə adı ilə QMainWindow -nun yuxarı hissəsində yerləşir. Bura qədər sinif daxilində yazdığımız QWidget
(class window(QMainWindow)) bir QMainWindow vidgetidir. qrafik görünüşünə nəzər yetirək.



Ümumi modeli ifadə=menubar()

QMenuBar

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:black;")
```

Vidget -i dəyişirik

```
        self.setGeometry(400,400,600,600)
        self.menu=QMenuBar(self)
        self.menu.addMenu('File')
        self.menu.addMenu('Edit')
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

Yuxarıdakı metoddan savayı digər yol aşağıdakı kimidir

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setStyleSheet("background-color:gray;")
        self.setGeometry(400,400,600,600)
        menubar=self.menuBar()
        f=menubar.addMenu('&File')
        f.addAction('New')
```

```
        f.addAction('Open')
        f.addAction('Save')
        f.addAction('Close')
        e=menubar.addMenu('Edit')
        e.addAction('view')
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

Menubar hər əməliyyat sistemində və desktoplar üçün fərqli çubuqda görünə bilir.

Klasın metodlarına nəzər yetirək

menuBar() - klassik menu çubuğu
addMenu - menu çubuğuna əlavələr
addAction - alt menuların yerləşdirilməsi(eləcədə icon və mətnlər üçün)
setEnabled(True or False) - menu çubuğunda olan əlavələri aktiv və pasiv etmək.
addSeparator() - alt menuları bir xətli ayırmaq
clear() - menu çubuğunu götürmək,silmək
setShortcut - menu-nu klaviatura ilə əlaqələndirmək

setText() - ifadə əlavə etmək
text()   - ifadələri götürmək
title()   - QMenu üçün başlıq

Signal
triggered

Menu çubuğunun arxa plan rəngini dəyişdirmək üçün

menubar.setStyleSheet('background-color:gray;') ifadəsini əlavə edirik.

Menu çubuğunu ümumilikdə css ilə dizayn etmək üçün

```
menubar.setStyleSheet("color:#365471;"
                      "background-color: black;"
                      "selection-color: #172533;"
                      "selection-background-color: #BFBFBF;")
```

kodları əlavə edirik

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,600,600)
        self.setStyleSheet("background-color:#121D28;")
        menubar=self.menuBar()
        menubar.setStyleSheet("color: #365471;"
                              "background-color: black;"
                               "selection-color: #172533;"
                              "selection-background-color: #BFBFBF;")
```

```
        f=menubar.addMenu('File')
        f.addAction('New')
        f.addAction('Open')
        f.addAction('Save')
        f.addAction('Close')
        e=menubar.addMenu('Edit')
        e.addAction('view')
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

setShorcut('string',vidjet) metodu

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
```

```
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,600,600)
        self.setStyleSheet("background-color:#121D28;")
        menubar=self.menuBar()

        menubar.setStyleSheet("color: #365471;"
                              "background-color: black;"
                              "selection-color: #172533;"
                              "selection-background-color: #BFBFBF;")
        f=menubar.addMenu('File')
        f.addAction('New')
        f.addAction('Open')
        f.addAction('Save')
        close=QAction('Close',self)
        close.setShortcut("Esc")
        close.triggered.connect(self.closeing)
        f.addAction(close)
        e=menubar.addMenu('Edit')
        e.addAction('view')
        self.show()
    def closeing(self):
        self.close()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız arasına metodu əlavə edərək istifadəçinin istifadəçinin yalnız Esc düyməsini basdıqda belə pəncərənin bağlanmasını tənzimlədik.Alt menu görünüşünü css kodları ilə tənzimləyək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
```

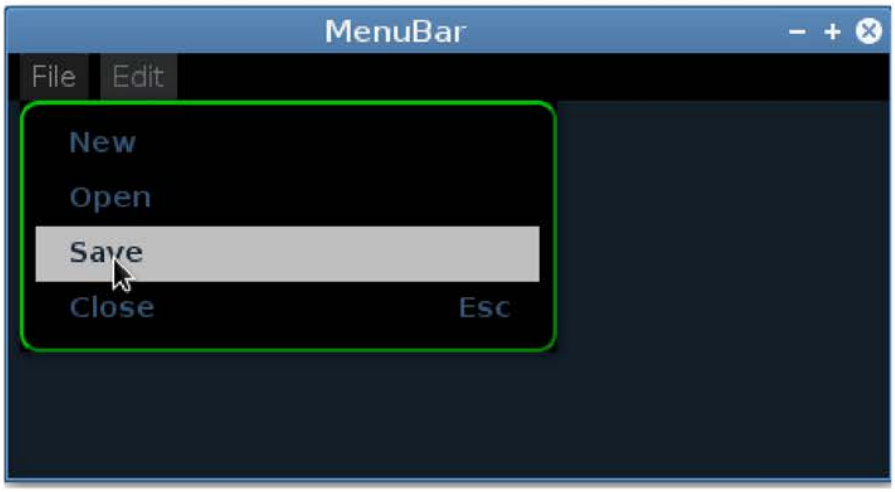
```
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(400,400,600,600)
        self.setStyleSheet("background-color:#121D28;")
        menubar=self.menuBar()

        menubar.setStyleSheet("color: #365471;"
                                    "background-color: black;"
                                    "selection-color: #172533;"
                                     "selection-background-color: #BFBFBF;")


        f=menubar.addMenu('File')
        f.setStyleSheet(" background-color: black;"
                            "border-style: outset;"
                            "border-width: 2px;"
                            "border-radius: 10px;"
                             "border-color: green;"
                             "font: bold 14px;"
                             "min-width: 10em;" )


        f.addAction('New')
        f.addAction('Open')
        f.addAction('Save')
        close=QAction('Close',self)
        close.setShortcut("Esc")
        close.triggered.connect(self.closeing)
        f.addAction(close)
        e=menubar.addMenu('Edit')
        e.addAction('view')
        self.show()
    def closeing(self):
        self.close()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



Kodlarımızda css kodlama dili ilə alt menulara bəzək verdik.
Alt menulara digər menuların əlavə edilməsi üçün

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 500, 300)
        self.setStyleSheet("background-color:#121D28;")
        menubar=self.menuBar()

        menubar.setStyleSheet("color: #365471;"
                              "background-color: black;"
                              "selection-color: #172533;"
                               "selection-background-color: #BFBFBF;")
        f=menubar.addMenu('File')
        f.setStyleSheet(" background-color: black;"
                        "border-style: outset;"
```

```
                            "border-width: 2px;"
                            "border-radius: 10px;"
                            "border-color: green;"
                             "font: bold 14px;"
                             "min-width: 10em;"
                              )
        f.addAction('New')
        f.addAction('Open')
        f.addAction('Save')
        close=QAction('Close',self)
        close.setShortcut("Esc")
        close.triggered.connect(self.closeing)
        f.addAction(close)
        e=menubar.addMenu('Edit')
        e.setStyleSheet("background-color:black;"
                            "border-style: outset;"
                            "border-width: 2px;"
                            "border-radius: 10px;"
                            "border-color: red;"
                            "font: bold 14px;"
                              )
        edit=e.addMenu('View')
        edit.addAction('Style')
        edit.addAction('Toolbar')
        color=QAction('Color',self)
        e.addAction(color)
        color.triggered.connect(self.colorscheme)
        self.show()
    def closeing(self):
        self.close()
    def colorscheme(self):
        color=QColorDialog.getColor()
        self.setStyleSheet("background-color:%s"%color.name())
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

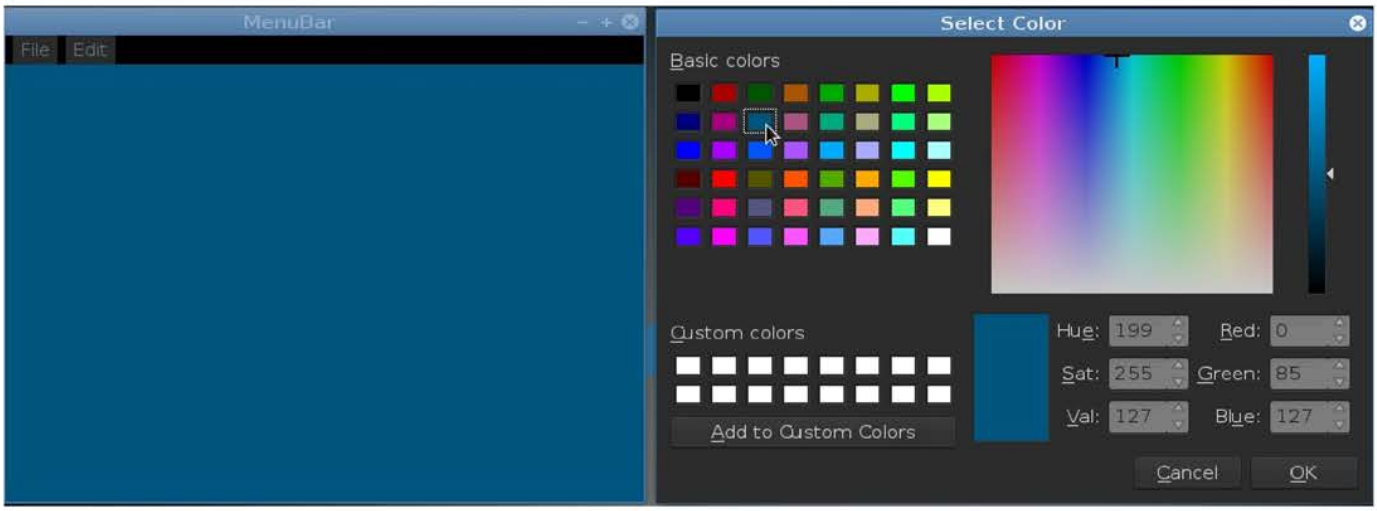
Kodlarımız daxilində e=menubar.addMenu('Edit')yazaraq menu çubuğuna Edit əlavə etdik.Daha sonra Css kodları ilə bəzədik.Və Edit alt menu üçün edit=e.addMenu('View') ifadəsi ilə yeni menu əlavə etdik.Qeyd edimki addMenu ifadəsi alt menularda yanında açılış işarəsi ilə əlavə olunur,yəni onun da alt menuları əlavə olunacaq şəkildə görünür.Amma tək bir alt menu üçün

addAction ifadəsini istifadə edə bilərik.Biz View alt menusuna digər menular əlavə etmək üçün edit.addAction('Style') ifadəsindən istifadə etdik.Daha sonra sərbəst bir alt menu üçün color=QAction('Color',self) ifadəsini yazdıq.

```
e.addAction(color)
```

color=QAction('Color',self) ifadəsini yazmaqda məqsədimiz bu alt menuya funksiya əlavə etməkdir.Bu color menunu Edit daxilinə əlavə etmək üçün e.addAction(color) kodunu əlavə etdik.Daha sonra menuların Signalı olan triggered istifadə edərək color.triggered.connect(self.colorscheme) ifadəsi ilə self.colorscheme funksiyasını çağırırıq.Bu çağırış yalnız color alt menusunu basdıqda reallaşır.İndidə funksiyamızı izah edək.Funksiya daxilində QColorDialog -hazır klasından istifadə etdik. color=QColorDialog.getColor() Daha sonra Ümumi ana pəncərəmizin arxa plan rəngini dəyişmək üçün Css kodları ilə yazıb,pythonun bizə alternativ yol göstərdiyi %s əmri ilə arxa plan rəngini color.name() seçili rəng ilə tənzimlədik.

Ekran görünüşü



Alt menuları və ya menuları aktiv və pasiv etmək üçün setEnabled() metodundan istifadə olunur.Metod iki parametr alır.True və False. color.setEnabled(False) ifadəsini kodlarımıza əlavə edib Qcolor dialogunu passiv edə bilərik və ya digərlərini.

```
edit=e.addMenu('View')
edit.addAction('Style')
edit.addAction('Toolbar')
color=QAction('Color',self)
color.setEnabled(False)    #al menu passiv edirik
```

```
        e.addAction(color)
        color.triggered.connect(self.colorscheme)
        self.show()
```

və ya metoda əks olan color.setDisabled(True) metodundan da istifadə edə bilərik

```
        edit=e.addMenu('View')
        edit.addAction('Style')
        edit.addAction('Toolbar')
        color=QAction('Color',self)
        color.setDisabled(True)
        e.addAction(color)
        color.triggered.connect(self.colorscheme)
        self.show()
```

QToolBar Widget

vidjet,Menu çubuğunun alt hissəsində qərarlaşmaqla daxilində,icon,mətn və düymələrin yaradılmasını təmin edir.

toolbar=self.addToolBar('verilən')

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 500, 300)
        self.setStyleSheet("background-color:#121D28;")
        toolbar=self.addToolBar('')
        new=QAction(QIcon('new.jpeg'),'New',self)
        open=QAction(QIcon('open.png'),'Open file',self)
        exit=QAction(QIcon('exit.png'),'Exit',self)
        toolbar.addAction(new)
        toolbar.addAction(open)
```

```
        toolbar.addAction(exit)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

Kodlarımızda toolbar=self.addToolBar('') ifadəsi ilə toolbar yaratdıq.QmainWindow vidgeti yerləşmə nöqtəsinə görə menuları,toolbar statusbar -ı avtomatik yerləşdirir.

Daha sonra internətdən 3 ədəd yeni open və exit iconlarını yüklədim.Bu iconların toolBar da görünməsi üçün

```
new=QAction(QIcon('new.jpeg'),'New',self)
open=QAction(QIcon('open.png'),'Open file',self)
exit=QAction(QIcon('exit.png'),'Exit',self)
```

hər birini bir dəyişənə atıb ardıcıl olaraq

```
toolbar.addAction(new)
toolbar.addAction(open)
toolbar.addAction(exit)
```

-a əlavə etdim.
Bu vidjetdə setToolTip metodunu istifadə etməyə ehtiyyac yoxdur.QAction

daxilində bunu yerinə yetirdik.iconların üzərinə gəldikdə tooltip mətnləri görünür.

Və bu üç dəyişənə funksiyaları əlavə etmək üçün bilirikki dəyişnlərin hər birinə triggered.connect ifadəsini verməklə edə bilərik.

Misal üçün exit toolbar menusunu çıxış kimi tərtib edək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 500, 300)
        self.setStyleSheet("background-color:#121D28;")
        toolbar=self.addToolBar('')
        new=QAction(QIcon('new.jpeg'),'New',self)
        open=QAction(QIcon('open.png'),'Open file',self)
        exit=QAction(QIcon('exit.png'),'Exit',self)
        exit.triggered.connect(self.closing)
        toolbar.addAction(new)
        toolbar.addAction(open)
        toolbar.addAction(exit)
        self.show()
    def closing(self):
        self.close()
if __name__ == '__main__':
    app = QApplication([])
```

Closing funksiyasını çağırırıq

closing funksiyası daxilində əsas pəncərənin bağlanmasını tərtib etdik.

Biz icon icon ölçülərini böyüdüb kiçiltməklə toolbarın həcmini dəyişə bilərik toolbar.setIconSize(QSize(15,15)) ifadəsini əlavə etməklə

ToolBar -ı horizontal və vertical istiqamətdə tutmaq üçün setOrientation metodundan istifadə edəcəyik.Metod Qt.Horizontal və Qt.Vertical

parametrlərini alır .İstifadə modeli toolbar.setOrientation(Qt.Vertical) formasındadır

addWidget() metodu

metod vasitəsilə toolbar a qutular mətnlər əlavə etmək olur.Bundan başqa toolbarı buttonstyle formasında da göstərmək olur.əgər fikir verirsinizsə toolbarın sol hissəsindən tutub hara istəsək yönəldə bilirik.Bunu disable etmək üçün bizə setMovable() metodunu təklif edir.Metod bool-True və False parametrlərini alır.

toolbar.setToolButtonStyle(Qt.ToolButtonTextBesideIcon|Qt.AlignLeading) -Düymə formasında təşkili üçün

Kodlarımıza əlavələrimizi edib yekun ekran görünüşünə baxaq.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 500, 300)
        self.setStyleSheet("background-color:#121D28;")
        toolbar=self.addToolBar('')
        toolbar.setIconSize(QSize(15,15))
        toolbar.setStyleSheet("background-color:green;")
        new=QAction(QIcon('new.jpeg'),'New',self)
        open=QAction(QIcon('open.png'),'Open file',self)
        exit=QAction(QIcon('exit.png'),'Exit',self)
        exit.triggered.connect(self.closing)
        toolbar.addAction(new)
        toolbar.addAction(open)
        toolbar.addAction(exit)
        toolbar.setToolButtonStyle(Qt.ToolButtonTextBesideIcon|Qt.AlignLeading)
        line=QLineEdit()
        line.setStyleSheet("background-color:white;")
        toolbar.addWidget(line)
        toolbar.setMovable(False)
```

```
        self.show()
    def closing(self):
        self.close()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QDialog Class

Bəsit dialoq klasıdır.Çox istifadə olunan klas deyil

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('MenuBar')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 800,800)
        toolbar=self.addToolBar('')
        toolbar.setIconSize(QSize(20,20))
        toolbar.setStyleSheet("background-color:green;")
        new=QAction(QIcon('new.jpeg'),'New',self)
        open=QAction(QIcon('open.png'),'Open file',self)
        exit=QAction(QIcon('exit.png'),'Exit',self)
        toolbar.addAction(new)
        toolbar.addAction(open)
        toolbar.addAction(exit)
        new.triggered.connect(self.dialogue)
        self.show()
    def dialogue(self):
        self.dialog=QDialog()
        self.button=QPushButton('Ok',self.dialog)
        self.button.move(50,50)
        self.button.clicked.connect(self.dialog.accept)
        self.dialog.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
```

app.exec_()

QMessageBox
klas bir məlumat pəncərəsidir.Ümumi dörd formada məlumat verir.

1.Sual Question(QMessageBox.question) 

2.Məlumat İnformation(QMessageBox.information) 

3.Həyəcan Warning(QMessageBox.warning) 

4.Kəskin həyəcan(kritik)
Critical(QMessageBox.critical) 

5.Haqqında About (QMessageBox.about) icon -no..

QMessageBox düymə formatları

QMessageBox.Ok	QMessageBox.Close
QMessageBox.Open	QMessageBox.Yes
QMessageBox.Save	QMessageBox.No
QMessageBox.Cancel	QMessageBox.Abort
QMessageBox.Retry	QMessageBox.Ignore

Metodlar

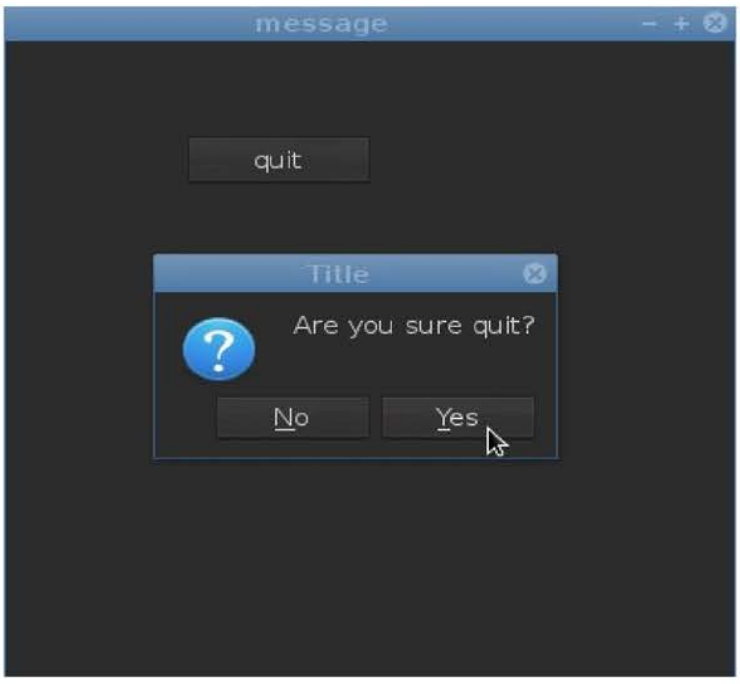
setInformativeText('ifadə')
setDetailText('ifadə')
setTitle('ifadə')

Signal
messaje=buttonClicked.connect(Slot_funksiya)

Bütövlükdə bu məlumatlar mesaj pəncərəsi üçün icon -dur.Yəni informasiya növünü öncədən bəlli etsək,mesaj pəncərəsi ona uyğun icon yerləşdirəcəkdir. Bütün bu parametrlər,setIcon metodu daxilində yer alır.Və ya dəyişənə atıb ifadə etmək olur.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('message')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 400,400)
        self.button=QPushButton('quit',self)
        self.button.move(100,60)
        self.button.clicked.connect(self.messaje)
        self.show()
    def messaje(self):
        message=QMessageBox()
        message.setWindowTitle('QMessageBox')
        message=QMessageBox.question(self,'Title','Are you sure
quit?',QMessageBox.Yes | QMessageBox.No)
        if message==QMessageBox.Yes:
            self.close()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



message=QMessageBox.question(self,'Title','Are you sure quit?',QMessageBox.Yes QmessageBox.No)

ifadəsi ilə pəncərəni sual formatıda,mötərizə daxilində ana pəncərəyə self ifadəsilə tətbiq edərək daha sonra mətn yazırıq ardından düymə tərtib edərək mesajı veririk.

QInputDialog Widget

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys,time
import pygame, random, sys
from pygame.locals import *
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('message')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 300,100)
        self.button=QPushButton('Dalogue',self)
        self.connect(self.button,SIGNAL('clicked()'),self.dialog)
        self.setFocus()
        self.line=QLineEdit(self)
        self.line.move(130,0)
        self.show()
    def dialog(self):
        text,ok=QInputDialog.getText(self,'Input Dialog','Enter you name')
        if ok:
            self.line.setText(unicode(text))
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Dialog spinbox və combobox daxil olmaqla məlumatları toplayıb funksiya vasitəsilə görünməsini təmin edir.Vidjetin ala bildiyi bir neçə metoda nəzər salaq

getInt()
getDouble()
getText()
getItem()

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QİnputDialog')
        layout=QFormLayout()
        self.button=QPushButton('Choose from list')
        self.button.clicked.connect(self.i)
        self.line=QLineEdit()
        layout.addRow(self.button,self.line)
        self.button1=QPushButton('name')
        self.button1.connect(self.button1,SIGNAL('clicked()'),self.t)
        self.line1=QLineEdit()
        layout.addRow(self.button1,self.line1)
        self.setLayout(layout)
        self.show()
    def i(self):
        items=('Python','Rubby Rails','Perl','Pascal','Css')
        item,ok=QInputDialog.getItem(self,'select input dialog','list of
language',items,0,False)
        if ok and item:
            self.line.setText(item)
    def t(self):
        text,ok=QInputDialog.getText(self,'Text Input Dialog','Enter you name:')
        if ok:
            self.line1.setText(str(text))
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

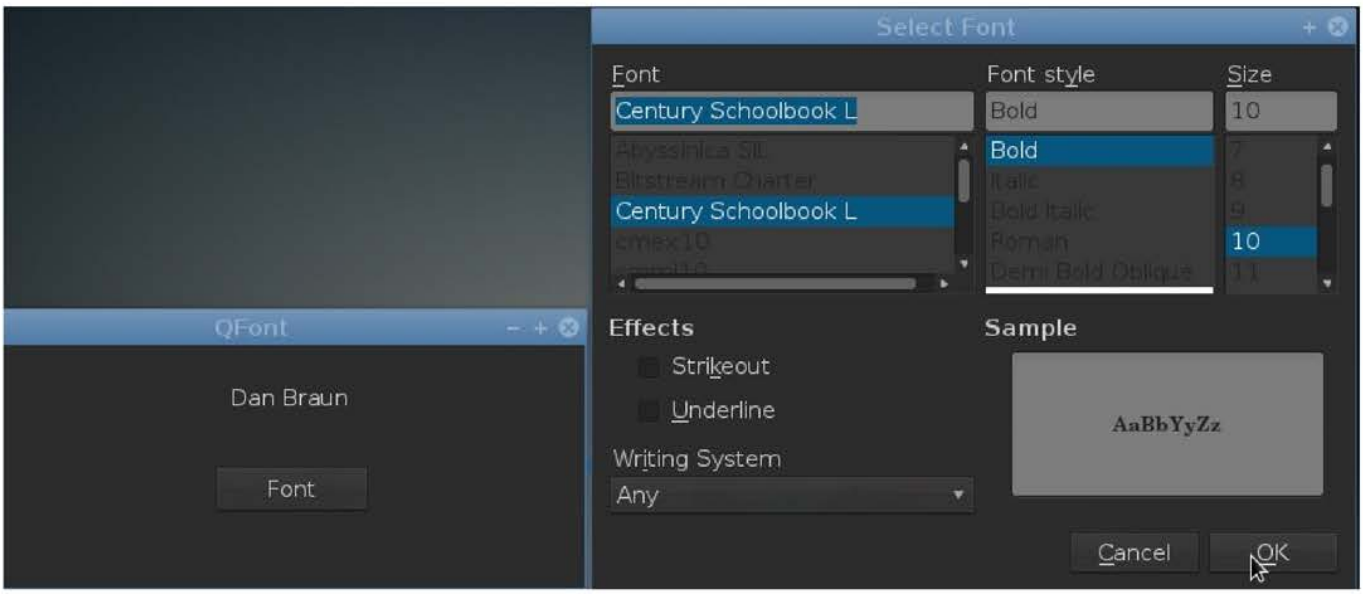
Ekran görünüşü

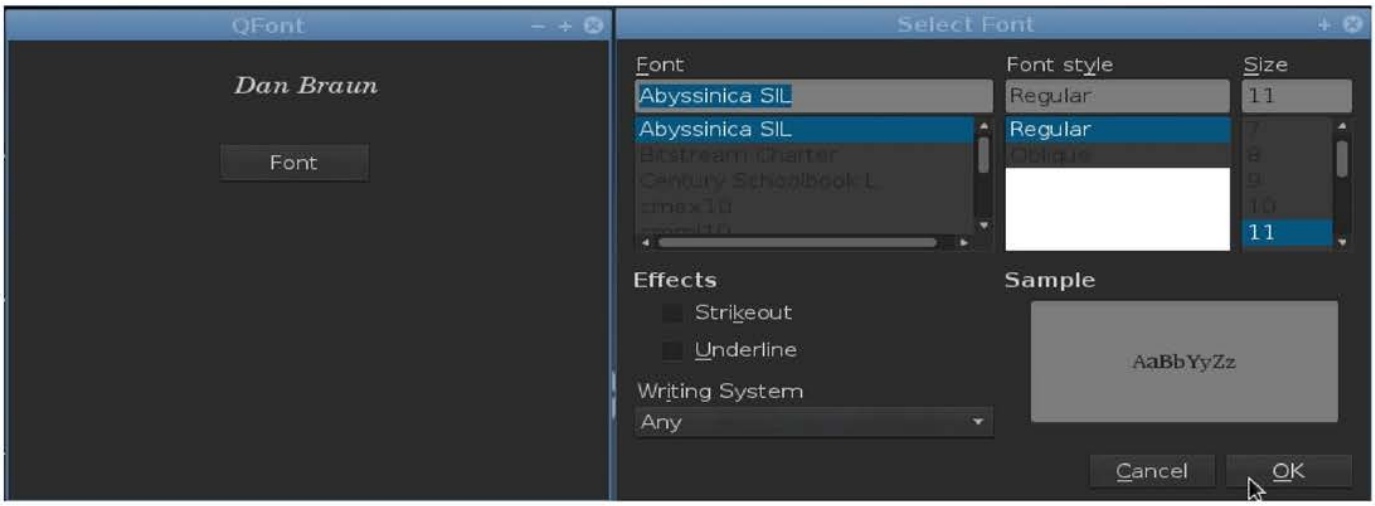
QFontDialog Widget

vidjet funksiya vasitəsilə QfontDialog pəncərəsini çağırır

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QFont')
        self.setGeometry(400,400,400,400)
        self.label=QLabel(u'Dan Braun',self)
        self.label.move(150,20)
        self.button=QPushButton('Font',self)
        self.button.move(140,80)
        self.button.clicked.connect(self.q)
        self.show()
    def q(self):
        okey,font=QFontDialog.getFont()
        if font:
            self.label.setFont(okey)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü





QFileDialog widget

Vidjet fayl seçmək üçün pəncərədir.Daxilində metod və funksiya ilə iki əsas metodla getOpenFileName() və getSaveFileName() tərtib olunub.İstifadə modeli aşağıdakı kimidir

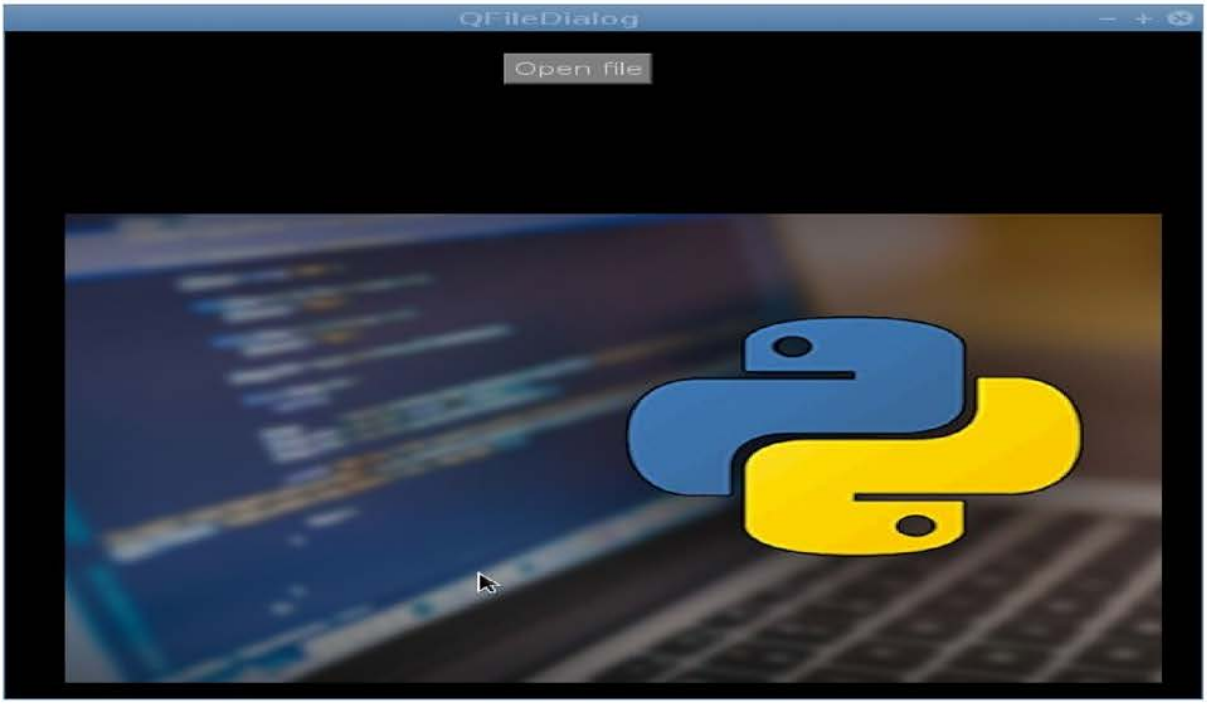
f= QFileDialog.getOpenFileName(self, 'Open file', '/home',"Image files (*.jpg *.gif)")

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QFileDialog')
        self.setGeometry(400,400,600,600)
        self.setStyleSheet('background-color:black;')
        self.button=QPushButton('Open file',self)
        self.button.setStyleSheet('background-color:gray;')
        self.button.setFocusPolicy(False)
        self.button.move(250,20)
        self.button.clicked.connect(self.getfile)
        self.label=QLabel(self)
        self.label.setGeometry(30,100,550,550)
        self.show()
    def getfile(self):
        f=QFileDialog.getOpenFileName(self,'Open
file','/home/Pictures','Image(*.)')
        self.label.setPixmap(QPixmap(f))
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QWidget

Kodlarımız daxilində düymə hazırladıq,daha sonra düyməni getfile funksiyası ilə əlaqələndirdik.label isə əldə edəcəyimiz şəkili ana pəncərədə yerləşdirmək üçün tərtib etdik.ölçüsünü də ana pəncərə ölçüsündən az olmaq şərtilə təyin etdik.
Və nəticədə home (windows üçün c:\\ yazılır) ana qovluğa daxil olaraq şəkli seçməyə nail olduq.

Ekran görünüşü

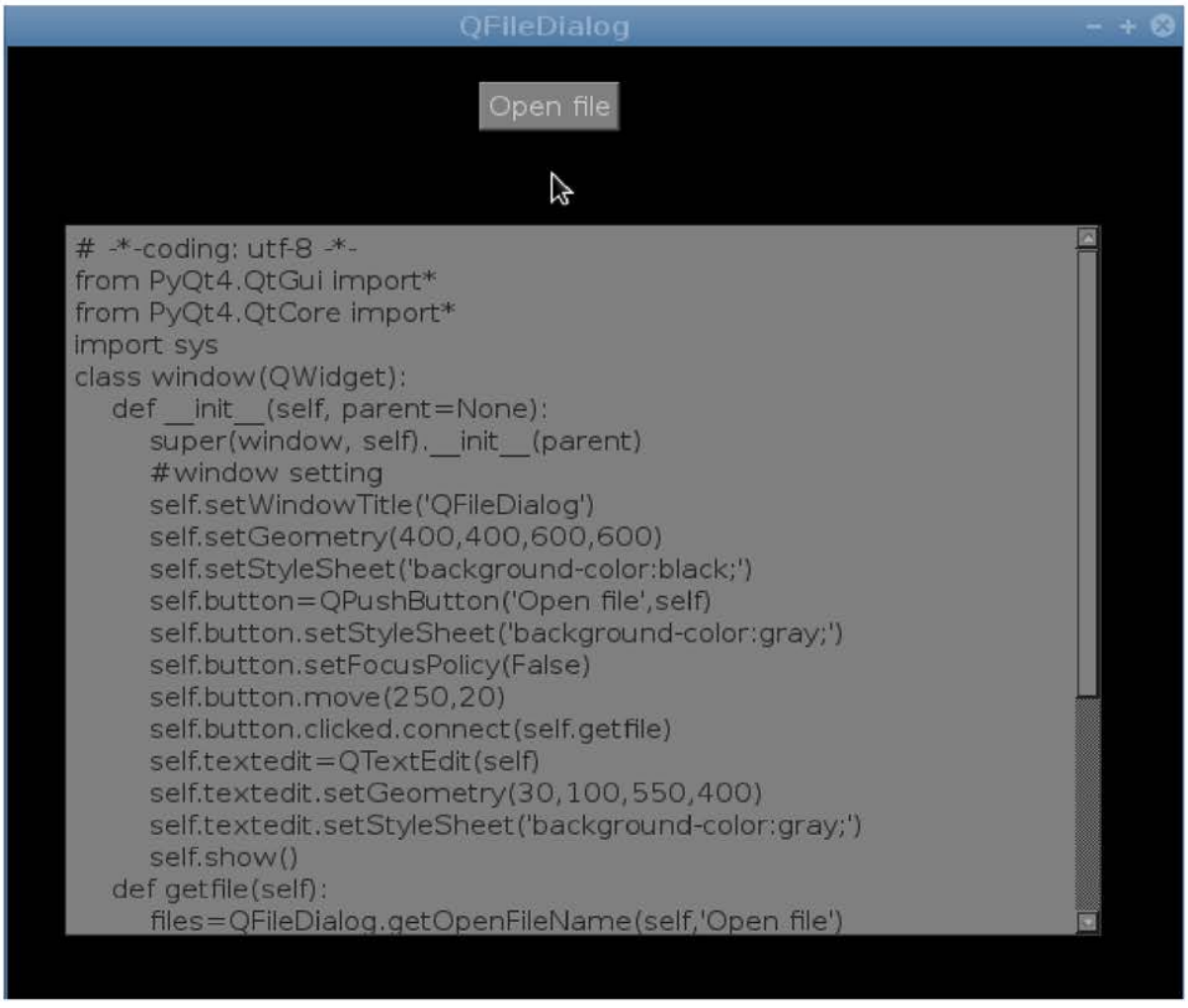


Bundan başqa mətn və ya digər modlarda olan faylları da açа bilərik,amma bunları yalnız labeldə tətbiq edə bilmərik.Yuxarıda şəkli etiketə(label) tətbiq etməyimizin səbəbi,labelin QPixmap metodunu ala bilməsidir.əgər bir txt (py,rb,cpp,java və s)formatında fayl açmaq istəsək o zaman QTextEdit istifadə edəcəyik.Bu vidjeti irəlidə daha aydın keçəcəyimiz üçün sadəcə filedialoga tətbiq edib sizə göstərmək istərdim.İrəlidə filedialogdan istifadə edib bir mətn editoru yaradacağıq.Haşiyədən kənara çıxmayaraq kodlarımıza davam edək.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #window setting
        self.setWindowTitle('QFileDialog')
        self.setGeometry(400,400,600,600)
        self.setStyleSheet('background-color:black;')
        self.button=QPushButton('Open file',self)
        self.button.setStyleSheet('background-color:gray;')
        self.button.setFocusPolicy(False)
        self.button.move(250,20)
        self.button.clicked.connect(self.getfile)
```

```
        self.textedit=QTextEdit(self)
        self.textedit.setGeometry(30,100,550,400)
        self.textedit.setStyleSheet('background-color:gray;')
        self.show()
    def getfile(self):
        files=QFileDialog.getOpenFileName(self,'Open file')
        file=open(files,'r')
        with file:
            text=file.read()
            self.textedit.setText(text)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



QFileDialog.getSaveFileName

yuxarıdakı kodlarımızı save düyməsi açıb aşağıdakı funksiyanı çağıracaq şəkildə yaza bilərik.

```
def save_file(self):
        self.name=QFileDialog.getSaveFileName(self,'Open file','/home','Save File (*.py)')
        file=open(self.name,'w')
        text=self.textedit.toPlainText()
        file.write(text)
        file.close
```

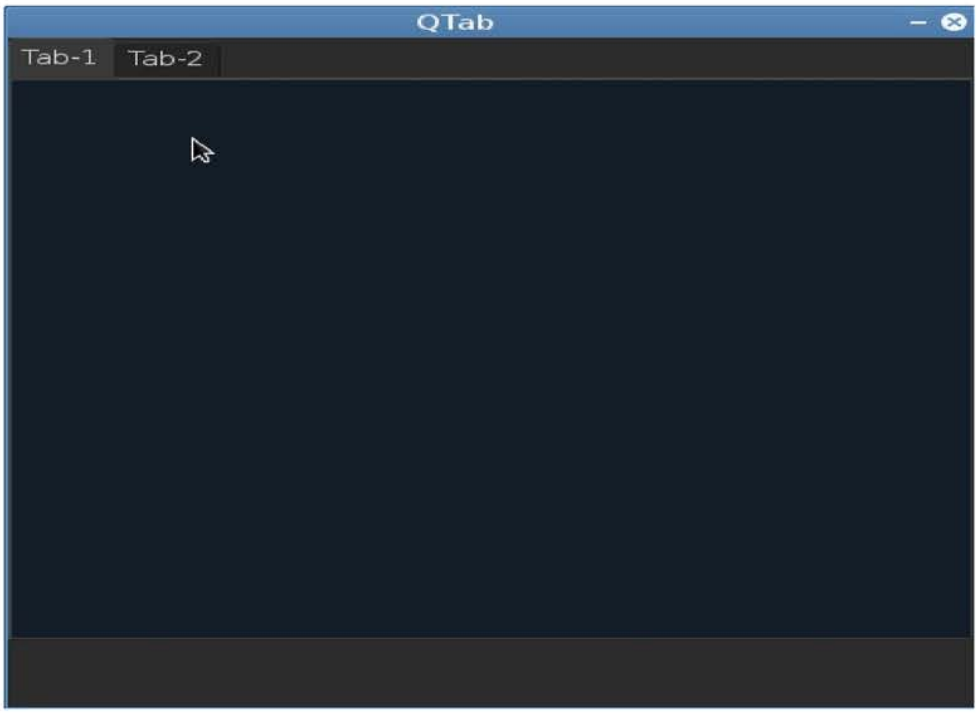
QTabWidget()

Brauzerlərdə rastlaşdığımız tab pəncərələri buna misal ola bilər.Tab vidjet daxilində istənilən verilənləri yerləşdirə bilir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #-----Main Window setting--------
        self.main()
    def main(self):
        self.main=QWidget()
        self.main.setWindowTitle('QTab')
        self.main.setGeometry(400,200,500,500)
        self.main.setFixedSize(500,500)
         self.main.show()
        #------Tab Widget---------
        self.tab=QTabWidget(self.main)
        self.tab.resize(500,450)
        self.tab1=QWidget()
        self.tab1.setStyleSheet("background-color:#121D28;")
        self.tab2=QWidget()
```

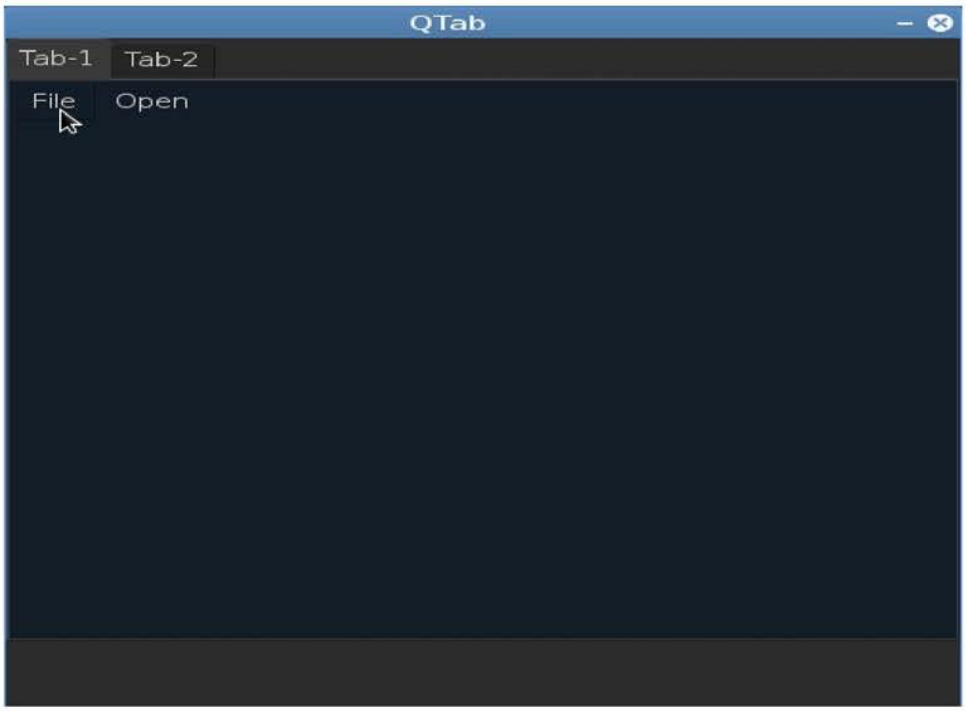
```
        self.tab.addTab(self.tab1,'Tab-1')
        self.tab.addTab(self.tab2,'Tab-2')
        self.tab.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



self.tab=QTabWidget(self.main) ifadəsi ilə vidjeti self.tab dəyişənə verdik.Ardından self.tab.resize(500,450) ifadəsi ilə tab ölçüsünü qeyd etdik.Biz əgər ölçü verməyib tab vidjetini açsaq görünüş bərbad olacaq.Ana pəncərə daxilində iki ədəd tab1 və tab2 vidjeti tərtib etdik.self.tab.resize(500,450) ifadəsini yazmaqla bundan sonra qeyd edəcəyimiz digər tablarada eyni ölçünü verdik.əgər ikinci bir tab-ı digər ölçüdə qeyd etmək istəsəniz,ikinci tabı atdığınız dəyişəni resize metodunda göstərməyiniz tələb olunacaq.Davam edərək tab1 in arxa plan rəngini Css atributları ilə dəyişdik self.tab1.setStyleSheet("background-color:#121D28;") .

Və sonda ümumi tabların ana pəncərədə görünməsini self.tab.show() ifadəsi ilə yerinə yetirdik.Və görünüş yuxarıdakı şəkildədir.Bundan başqa siz tab vidjeti daxilində menubar və ya toolbar da qeyd edə bilərsiniz.Misal üçün




```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #-----Main Window setting---------
        self.main()
    def main(self):
        self.main=QWidget()
        self.main.setWindowTitle('QTab')
        self.main.setGeometry(400,200,500,500)
        self.main.setFixedSize(500,500)
        self.main.show()
        #------Tab Widget---------
        self.tab=QTabWidget(self.main)
```

```
        self.tab.resize(500,450)
        self.tab1=QWidget()
        self.tab1.setStyleSheet("background-color:#121D28;")
        self.tab2=QWidget()
        self.tab.addTab(self.tab1,'Tab-1')
        self.tab.addTab(self.tab2,'Tab-2')
        self.tab.show()
        #-------toolBar--------------------
        self.toolbar=QToolBar(self.tab1)
        self.toolbar.addAction('File')
        self.toolbar.addAction('Open')
        self.toolbar.setMovable(False)
        self.toolbar.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Self.tab1 -də görünməsini təşkil etdik

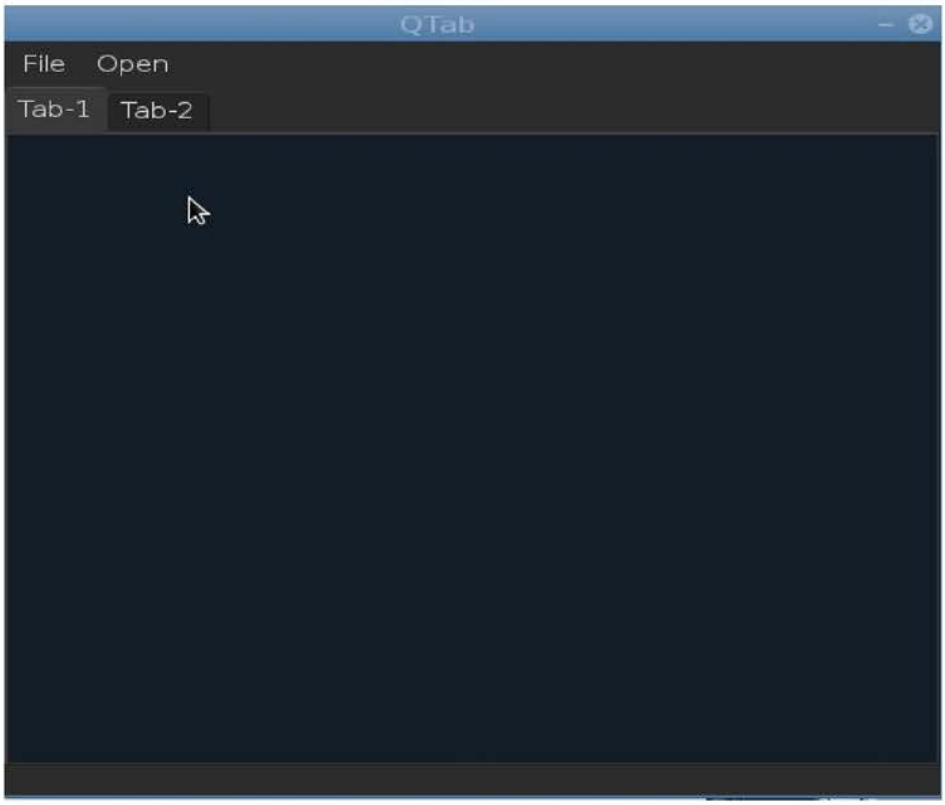
Kodlarımız daxilində self.toolbar=QToolBar(self.tab1) ifadəsi ilə toolbarı yaratdıq və görünməsini ana pəncərədə deyil self.tab1 -də olmasını tənzimlədik.əgər Tab üzərində bir toolbar açmaq istəsək bu zaman self.tab ifadəsinə setGeometry metodunu əlavə edərək tab-ı aşağı doğru yönəldəcəyik.
self.tab.setGeometry(0,30,500,450)

Kodlarımızın bütöv halı

```
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #-----Main Window setting---------
        self.main()
    def main(self):
        self.main=QWidget()
        self.main.setWindowTitle('QTab')
        self.main.setGeometry(400,200,500,500)
        self.main.setFixedSize(500,500)
        self.main.show()
        #------Tab Widget---------
        self.tab=QTabWidget(self.main)
        self.tab.setGeometry(0,30,500,450)
```

```
        self.tab1=QWidget()
        self.tab1.setStyleSheet("background-color:#121D28;")
        self.tab2=QWidget()
        self.tab.addTab(self.tab1,'Tab-1')
        self.tab.addTab(self.tab2,'Tab-2')
        self.tab.show()
        #-------toolBar------
        self.toolbar=QToolBar(self.main)
        self.toolbar.addAction('File')
        self.toolbar.addAction('Open')
        self.toolbar.setMovable(False)
        self.toolbar.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



Metodları

setTabPosition() metod dörd parametr alaraq vidjetin pəncərədə yerini göstərir.
1.QTabWidget.North yuxarı(şimal) hissədə
2.QTabWidget.South aşağı(cənub) hissədə
3.QTabWidget.West sol(qərb) hissədə
4.QTabWidget.East sağ(şərq) hissədə

İstifadə modeli - self.tab.setTabPosition(QTabWidget.West)

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QWidget):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #-----Main Window setting---------
        self.main()
    def main(self):
        #----------window setting-------------
        self.main=QWidget()
        self.main.setWindowTitle('QTab')
        self.main.setGeometry(400,200,500,500)
        self.main.setFixedSize(500,500)
        self.main.show()
        #------Tab Widget---------
        self.tab=QTabWidget(self.main)
        self.tab.setGeometry(0,30,500,450)
        self.tab1=QWidget()
        self.tab2=QWidget()
        self.tab.addTab(self.tab1,'Tab-1')
        self.tab.addTab(self.tab2,'Tab-2')
        self.tab.show()
        #----------button---------
        self.button=QPushButton('Start',self.tab1)
        self.button.move(20,20)
        self.button.show()
        #-------Label------------
        self.label=QLabel(u'Bakı Dövlət Universiteti',self.tab1)#Tab1-ə əlavə
        self.label.show()
        #--------QTextEdit---------
```

```
        self.textedit=QTextEdit(self.tab2)#Tab2 -ə əlavə
        self.textedit.show()
        #-------toolBar------
        self.toolbar=QToolBar(self.main)
        self.toolbar.addAction('File')
        self.toolbar.addAction('Open')
        self.toolbar.setMovable(False)
        self.toolbar.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Yuxarıdakı kodlara bəzi verilənləri tab vidjet daxilinə əlavə etdik.

Fikir verirsinizsə üzərində close düyməsi yoxdur.Misal üçün bir Brauzerdə,bir neçə açdığımız tab pəncərələri bağlamaq lazım gələcək.

QTab Brauzer

```
from PyQt4.QtGui import*
from PyQt4.QtCore import*
from PyQt4.QtWebKit import*
from PySide import QtCore, QtGui, QtWebKit, QtNetwork
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        #------------window setting--------------
        self.setWindowTitle('Text Editor')
        self.setGeometry(400,200,800,800)
```

Yuxarıdakı kodlarda ana pəncərəni tərtib etdik.

```
        #-------QToolBar--------------
        self.toolbar=QToolBar('',self)
        self.toolbar.setGeometry(0,770,500,30)
        go=QAction('Go',self)
        self.toolbar.addAction(go)
        go.triggered.connect(self.web)
        self.toolbar.setMovable(False)
        self.line=QLineEdit()
```

```
        self.toolbar.addWidget(self.line)
        self.toolbar.resize(500,30)
        add=QAction('addTab',self)
        self.toolbar.addAction(add)
        add.triggered.connect(self.addtab)
```

Və ardından toolbar ı pəncərənin aşağı hissəsində yerləşdirdik.Toolbar -a go və addtab düymələrini əlavə edirik

```
        #------QTab------------------
        self.tabs=QTabWidget(self)
        self.tabs.resize(800,700)
        self.tab=QWidget()
        self.tabs.addTab(self.tab,'')
        self.tabs.setMovable(True)
        self.tabs.setTabsClosable(True)
        self.tabs.tabCloseRequested[int].connect(self.tabclose)
        self.tab.show()
```

Tab videjetimizə self.tabs.setMovable(True) ifadəsini verməklə tab yerdəyişməsini True edirik,daha sonra tab üzərində close(x) işarəsinin görünməsi üçün self.tabs.setTabsClosable(True) ifadəsilə true parametri veririk.əgər bu ifadəni verməsək tablar üzərində close(x) işarəsi görünməyəcək. TabCloseRequested metodundan istifadə edərək istifadəçi tab üzərindəki close(x) işarəsinə vurduqda self.tabclose funksiyasını çağırmaqla tab -ı remove edirik. TabCloseRequested metodunun istifadə modelinə diqqət edin.PyQt ilə bağlı yazılan bir çox mənbələrdə xətalara yol veriblər.

Davam edərək aşağıdakı funksiyalara keçid edək

```
        self.show()
    def web(self,index):
        self.web=QWebView(self.tab)
        self.url='http://www.'+self.line.text()
        self.web.load(QUrl(self.url))
        self.web.resize(800,700)
        self.web.show()
```

Kodlarımızda istifadəçi Go düyməsinə basdıqda def veb(self,index) funksiyasını çağıracaq.Funksiya daxilində self.web=QWebView(self.tab) ifadəsini yazmaqla,səhifənin self.tab daxilində görünməsini təşkil edirik.url bölməsini sadəcə olaraq google.az yazmaqla QlineEdit klasından self.line.text() ələ alaraq self.web.load(QUrl(self.url)) ifadəsilə açırıq.və səhifə ölçüsünü en və uzunluğa

uyğun olaraq self.web.resize(800,700) ifadəsilə tərtib edirik.Səhifənin görünməsini isə self.web.show() ifadəsi təyin edir.

Toolbar -da olan addTab düyməsi isə aşağıdakı funksiyanı çağırır.

```
    def addtab(self,index):
        self.tab=QWidget(self)
        self.web=QWebView(self.tab)
        self.url='http://www.'+self.line.text()
        self.web.load(QUrl(self.url))
        self.web.resize(800,700)
        self.tabs.addTab(self.tab,str(self.tabs.setCurrentIndex(index)))
```

Funksiya bundan əvvəlki kodlardan çox fərqi yoxdur.Eyni ölçüləri yeni tab -a əlavə etdik.

Yuxarıdabəhs etdiyimiz tab üzərindəki close(x) işarəsini basdıqda self.tabs.tabCloseRequested[int].connect(self.tabclose) kodu self.tabclose funksiyasını çağırır.

```
    def tabclose(self,index):
        self.tabs.removeTab(index)
```

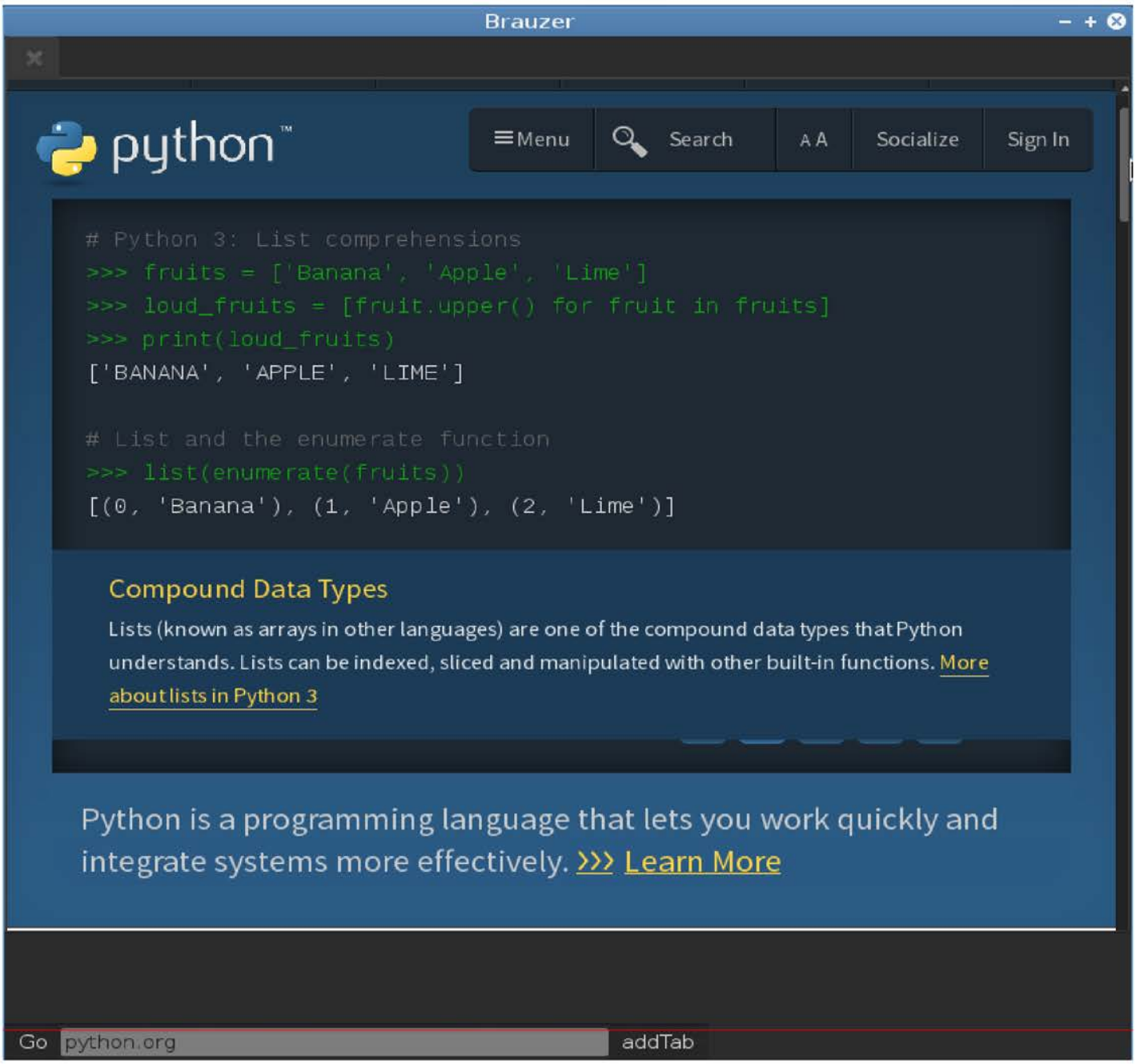
Funksiyamız vidjetin removeTab(int) funksiyası ilə istənilən əlavə edilmiş tab-ı silə bilir.

Və aşağıdakı kodları yazıb

```
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

proqramı çalışdırırıq

Ekran görünüşü



QStatusBar Widget
vidjet,pəncərənin ən son sətrində yer alan məlumat çubuğudur.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
```

```
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.statusBar().showMessage('Statusbar')
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Kodlarımız daxilində showMessage metodundan istifadə edərək pəncərədə ifadə yerləşdirdik.

addWidget()

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.label=QLabel("<a href
='http://www.techaz.wordpress.com'>techaz<a/>")
        self.label.setToolTip('http://www.techaz.wordpress.com')
        self.statusBar().addWidget(self.label)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

```
removeWidget()
statusBar().removeWidget(Widget)

# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('Del_wdg',self)
        self.button.clicked.connect(self.shw)
        self.button.show()
        self.label=QLabel("<a href
='http://www.techaz.wordpress.com'>techaz<a/>")
        self.label.setToolTip('http://www.techaz.wordpress.com')
        self.statusBar().addWidget(self.label)
        self.show()
    def shw(self):
        self.statusBar().removeWidget(self.label)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

clearMessage() metodu

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('Del_wdg',self)
        self.button.clicked.connect(self.shw)
        self.button.show()
        self.statusBar().showMessage('Statusclear')
        self.show()
    def shw(self):
        self.statusBar().clearMessage()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

addPermanentWidget(Widget,int)

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('Del_wdg',self)
```

```
        self.button.clicked.connect(self.shw)
        self.button.show()
      self.label=QLabel("<a href
='http://www.techaz.wordpress.com'>techaz<a/>")
        self.label.setToolTip('http://www.techaz.wordpress.com')
        self.show()
    def shw(self):
        self.statusBar().addPermanentWidget(self.label,1)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QListWidget

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('List')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('show',self)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.show()
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü

addItem('argument') metodu

Metod list qutusuna ifadə əlavə edir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('List')
        self.setGeometry(400,200,400,400)
```

```
        self.button=QPushButton('show',self)
        self.button.connect(self.button,SIGNAL('clicked()'),self.shw)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.show()
        self.show()
    def shw(self):
        self.list.addItem('Italy - Genoa')
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

addItems()

metod vasitəsilə, bir və ya birdən çox list şəklində ifadələr əlavə olunur.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('List')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('show',self)
        self.button.connect(self.button,SIGNAL('clicked()'),self.shw)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.show()
        self.show()
    def shw(self):
        self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Metoddan istifadə etdikdə çox sayda qeyd etdiyimiz ifadələr list qutusuna ardıcıl olaraq yerləşdi.

insertItem()

QListWidget.insertItem(int, QString)

metod list qutusuna sıra nömrəsini bildirməklə ifadəni əlavə edir.Yəni list 0-dan başlayaraq ifadələri saydığı üçün əgər biz 0,ifadə versək,qutu daxilində olan(və ya olayan)ifadələri sayıb 0-cı sıraya verdiyimiz ifadəni əlavə edəcək.
Misal olaraq

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('List')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('show',self)
        self.button.connect(self.button,SIGNAL('clicked()'),self.shw)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.show()
        self.show()
    def shw(self):
        self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
        self.list.insertItem(0,'Q.Q.Markez')
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

clear()

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QList')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('show',self)
        self.button.connect(self.button,SIGNAL('clicked()'),self.deleting)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
        self.list.show()
        self.show()
     def shw(self):
         self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
    def deleting(self):
        self.list.clear()
```

```
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

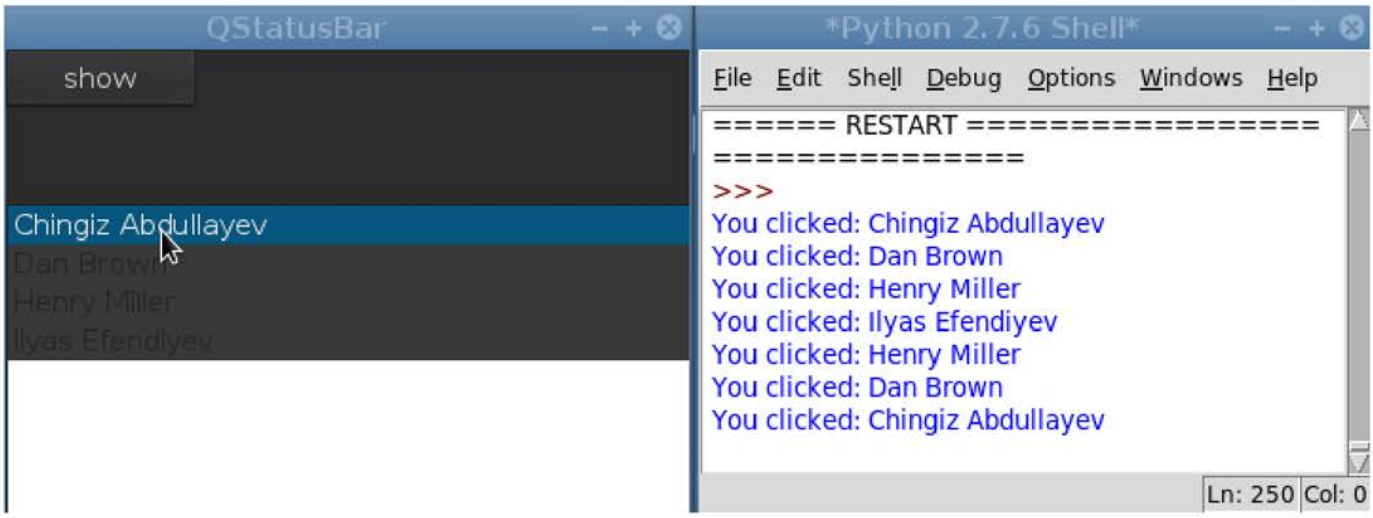
self.list.sortItems()
Metod list daxilindəki ifadələri əlifba sırası ilə düzür

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QList')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('show',self)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
        self.list.sortItems()
        self.list.show()
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

itemClicked() -Signal

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
```

```
    def main(self):
        self.setWindowTitle('List')
        self.setGeometry(400,200,400,400)
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
        self.list.sortItems()
        self.list.itemClicked.connect(self.click)
        self.list.show()
        self.show()
    def click(self,data):
        print 'You clicked: '+data.text()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```




```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
```

```
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.button=QPushButton('show',self)
        self.button.show()
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        self.list.addItems(['Henry Miller','Dan Brown','Chingiz Abdullayev','Ilyas
Efendiyev'])
        self.list.sortItems()
        self.list.currentItemChanged.connect(self.click)
        self.list.show()
        self.show()
    def click(self,data):
        print self.list.setCurrentItem(data)
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

.....

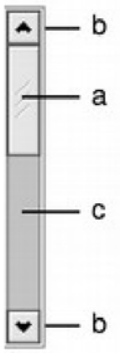
```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.list=QListWidget(self)
        self.list.setGeometry(0,80,400,300)
        for i in range(10):
            self.list.addItem('Items {} '.format(i))
        self.list.show()
        self.show()
if __name__ == '__main__':
```

```
    app = QApplication([])
    gui = window()
    app.exec_()
```

List qutusuna icon əlavə edilməsi

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.list=QListWidget(self)
        i=QListWidgetItem()
        i.setIcon(QIcon(r'icon.png'))
        self.list.addItem(i)
        self.list.setGeometry(0,80,400,300)
        self.list.show()
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QScrollBar Widget
Görünə bilən verilənləri qutu daxilində kənar çubuqla sürüşdürərək görünməyən hissəsinə götürür.Bir növ PgUp və PgDn düyməsini əvəzləyir.Görünüş etibarı ilə aşağıdakı kimidir.



PyQt gui proqramlama zamanı bir çox klaslar,həcmini çoxaltdığı zaman avtomatik olaraq scrol əlavə edir.

Vidjetin iki Signal -a malikdir
valueChanged() həcmin dəyişməsi
sliderMoved() hərəkət etməsi

Qutulara ScrollBar əlavə edilməsi

setVerticalScrollBarPolicy
setHorizontalScrollBarPolicy metodundan istifadə edəcəyik.

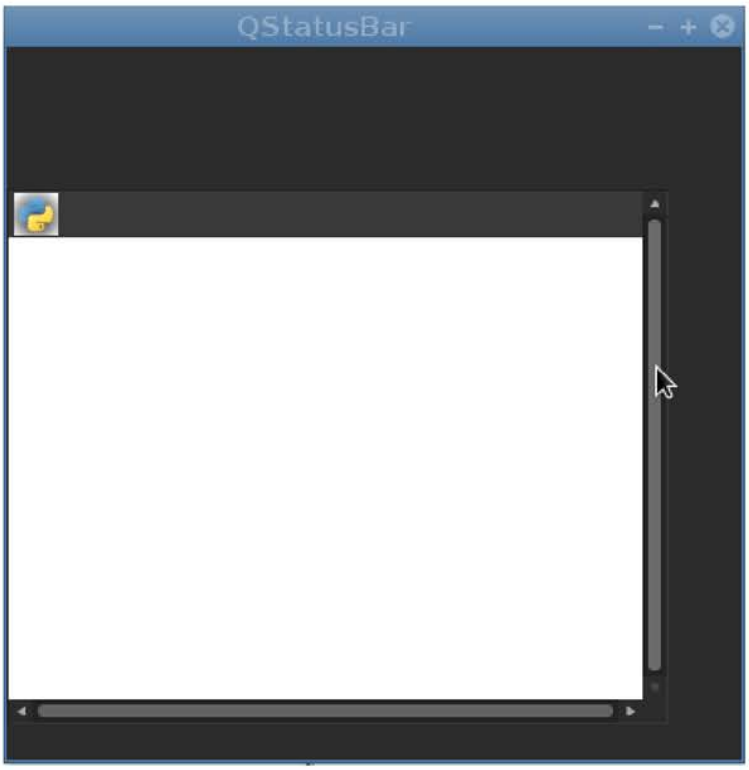
Metod Qt.ScrollBarAlwaysOn ; Qt.ScrollBarAlwaysOff parametr alır.

Bir List qutusuna həm vertikal həm də horizontal scroll əlavə edək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
```

```
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.list=QListWidget(self)
        i=QListWidgetItem()
        i.setIcon(QIcon(r'icon.png'))
        self.list.addItem(i)
        self.list.setGeometry(0,80,360,300)
        self.list.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
        self.list.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
        self.list.show()
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



Və ya ifadələr əlavə edərək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.list=QListWidget(self)
        i=QListWidgetItem()
        i.setIcon(QIcon(r'icon.png'))
        self.list.addItem(i)
        for v in range(100):
            self.list.addItem('{}'.format(v))
        self.list.setGeometry(0,80,360,300)
        self.list.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
        self.list.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOn)
        self.list.show()
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

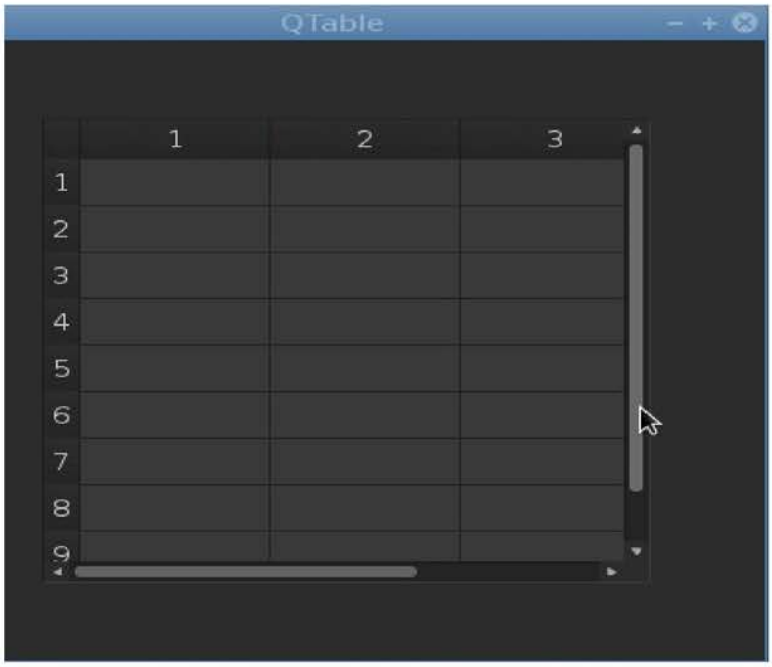
Ekran görünüşü


```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QStatusBar')
        self.setGeometry(400,200,400,400)
        self.scrol=QScrollBar(Qt.Vertical,self)
        self.scrol.setGeometry(50,80,10,200)
        self.scrol.show()
        self.scrol.sliderMoved.connect(self.f)
        self.show()
    def f(self):
        print 'value changed'
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

QTableWidget

Cədvəli təmsil edərk sətir və sütunlarla bölgü aparır və ifadələri sətir-sütunlara əlavə edir.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QTable')
        self.setGeometry(400,200,400,400)
        self.table=QTableWidget(self)
        self.table.setGeometry(20,50,320,300)
        self.table.setRowCount(9)
        self.table.setColumnCount(3)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```



self.table.setRowCount(int) sətir sayı
setColumnCount(int) sütun sayı

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QTable')
        self.setGeometry(400,200,400,400)
        self.table=QTableWidget(self)
        self.table1=QTableWidgetItem()
        self.table.setGeometry(20,50,320,300)
        self.table.setRowCount(9)
        self.table.setColumnCount(3)
        #--------------------------
        self.table.setItem(0,0,QTableWidgetItem('Item'))
        self.table.show()
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

self.table.setItem(0,0,QTableWidgetItem('Item')) ifadəsində 0,0 – tam ədədləri sətir 0,sütun 0 olmaqla İtem ifadəsini əlavə edirik.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QTable')
```

```
        self.setGeometry(400,200,400,400)
        self.table=QTableWidget(self)
        self.table1=QTableWidgetItem()
        self.table.setGeometry(20,50,320,300)
        self.table.setRowCount(9)
        self.table.setColumnCount(3)
        #-------------------------
        self.table.setColumnWidth(0,80)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

self.table.setColumnWidth(0,80) ifadəsilə sütun həcmini dəyişə bildik.

Çoxluq və listlərdən istifadə edərək ifadələri əlavə edək

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QTable')
        self.setGeometry(400,200,400,400)
        self.table=QTableWidget(self)
        self.table1=QTableWidgetItem()
        self.table.setGeometry(20,50,350,300)
        self.table.setRowCount(5)
        self.table.setColumnCount(3)
        #-------------------------
        self.table.setColumnWidth(0,140)
        v_item={'Ad':[u'Asim',u'Vüsal',u'Səbinə'],
                'Soyad':[u'Musabəyov',u'Rəhimov',u'Qazıyeva'],
                'Mobil':['552122213','502343434','704566779'],}
        head=[]
        for i,key in enumerate(sorted(v_item.keys())):
            head.append(key)
            for v,item in enumerate(v_item[key]):
                yeniitem=QTableWidgetItem(unicode(item))
```

```
                self.table.setItem(v,i,yeniitem)
                self.table.setHorizontalHeaderLabels(head)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Signal

```
table.clicked.connect(function)
table.cellChanged(function)
table.itemChanged(function)
```

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QTable')
        self.setGeometry(400,200,400,400)
        self.table=QTableWidget(self)
        self.table1=QTableWidgetItem()
        self.table.setGeometry(20,50,350,300)
        self.table.setRowCount(5)
        self.table.setColumnCount(3)
        #-------------------------
        self.table.setColumnWidth(0,140)
        v_item={'Ad':[u'Asim',u'Vüsal',u'Səbinə'],
                'Soyad':[u'Musabəyov',u'Rəhimov',u'Qazıyeva'],
                'Mobil':['552122213','502343434','704566779'],}
        head=[]
        for i,key in enumerate(sorted(v_item.keys())):
            head.append(key)
            for v,item in enumerate(v_item[key]):
                yeniitem=QTableWidgetItem(item)
                self.table.setItem(v,i,yeniitem)
                self.table.setHorizontalHeaderLabels(head)
        self.table.cellChanged.connect(self.bv)
```

```
        self.show()
    def bv(self,index):
        print 'You changed index:',index
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

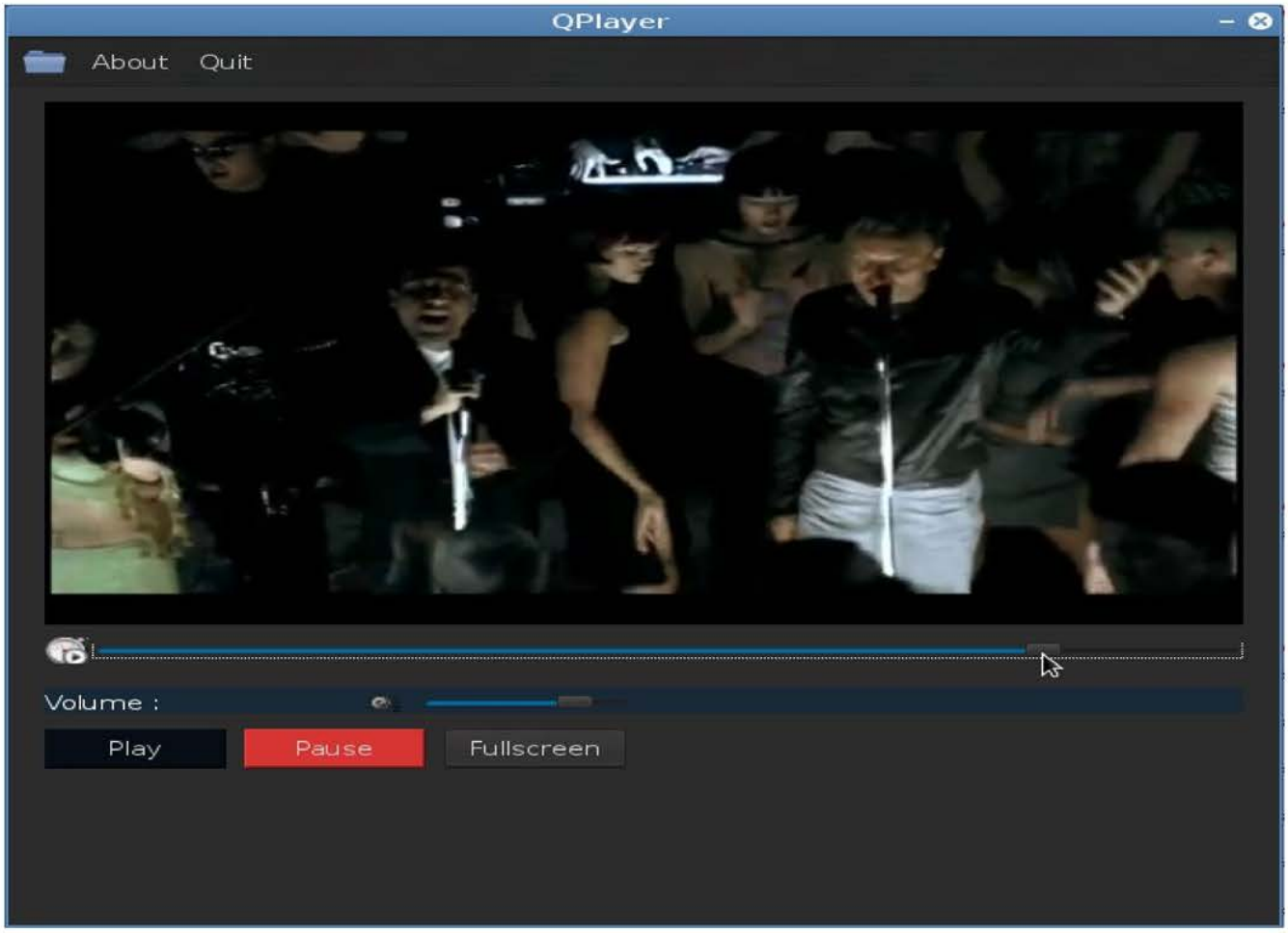
Cədvələ icon əlavə olunması

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.main()
    def main(self):
        self.setWindowTitle('QTable')
        self.setGeometry(400,200,400,400)
        self.table=QTableWidget(self)
        self.table1=QTableWidgetItem()
        self.table.setGeometry(20,50,350,300)
        self.table.setRowCount(5)
        self.table.setColumnCount(3)
        self.table.setStyleSheet('background-color:blue;')
        icon=QIcon(QPixmap('icon.png'))
        item=QTableWidgetItem(icon,'Python')
        self.table.setItem(0,0,item)
        self.show()
        #--------------------------
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Media player

https://github.com/RashadGarayev/QPlayer

Ekran görünüşü



ilkin kodlarımızı yazaq

```
from PyQt4.QtGui import*
from PyQt4.QtCore import*
from PyQt4.phonon import Phonon
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        _VERSION_=0.1
        #----------window setting-------------
        self.setWindowTitle('QPlayer')
        self.setWindowIcon(QIcon('icon.png'))
        self.setGeometry(50, 50, 700,680)
        self.setFixedSize(700,680)
```

```
#------------Execute all function(app)-----------------
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Yuxarıdakı kodlarımız arasında bizə yad olan ifadə yoxdur.Pəncərə başlığını Qplayer olaraq qeyd etdim.Daha Pəncərə üçün icon tətbiq etdik.Növbəti sətirdə isə pəncərənin koordinasıyasını eni və uzunluğunu yazdıq.self.setFixedSize(700,680) ifadəsi isə pəncərəyə müdaxiləni aradan qaldırmaq üçün eni və uzunluğunu stabil tərtib etdik.

Növbəti yazacağımız kodlarda isə ana pəncərəyə toolbar tətbiq edəcəyik

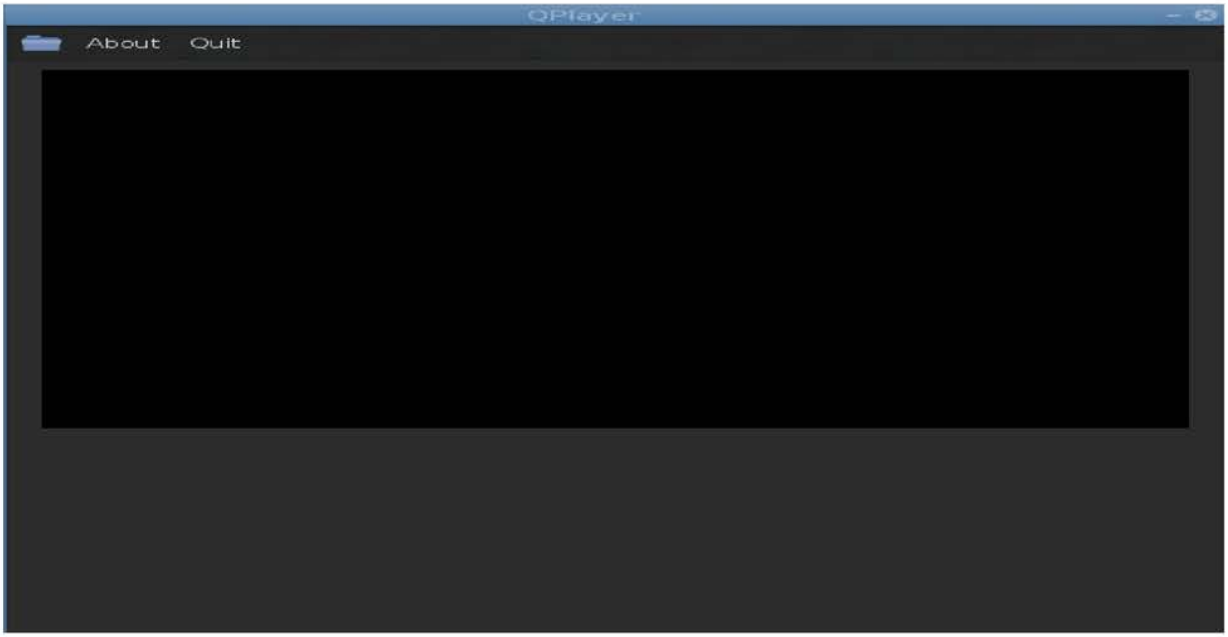
```
#--------ToolBar setting------------------------
        toolbar=self.addToolBar('')
        open=QAction(QIcon('open.png'),'open file',self)
        About=QAction('About',self)
        quit=QAction('Quit',self)
        toolbar.addAction(open)
        toolbar.addAction(About)
        toolbar.addAction(quit)
        #open.triggered.connect(self.open_file)
        #quit.triggered.connect(self.exiting)
        toolbar.setMovable(False)
        toolbar.setFocusPolicy(True)
```

toolbar=self.addToolBar('') ifadəsinin ardından open=QAction(QIcon('open.png'),'open file',self) ifadəsilə toolbar-a open menu(icon ilə) əlavə edirik və ardından About və quit əlavə edirik.əlavə üçün toolbar.addAction(quit) metodundan istifadə etdik.Qarşısına #- yazdığımız ifadə isə digər funksiyaya bağlı olduğundan hələlik istifadəsi bizə lazım deyil.Funksiyanı irəlidə yazıb izahat gətirəcəyik.Növbəti kodlarımıza keçək

```
#---------Phonon source--------------------
        self.media = Phonon.MediaObject(self)
        #self.media.stateChanged.connect(self.statech)
        self.video= Phonon.VideoWidget(self)
        self.audio = Phonon.AudioOutput(Phonon.VideoCategory, self)
        self.video.setGeometry(20,50,660,400)
        self.audio = Phonon.AudioOutput(Phonon.VideoCategory, self)
```

```
Phonon.createPath(self.media, self.audio)
Phonon.createPath(self.media, self.video)
```

Biz yuxarıdakı kodlarımızda Phonon modulunu istifadə etdik.Phonon.MediaObject() -i self.media dəyişəninə atdıq.İkinci sətir irəlidə lazım olacaq.Daha sonra eyni qayda ilə video və audio dəyişənlərinə modul klaslarını əlavə etdik.Və video pəncərəsinin görünüşünü setGeometry metodu ilə təyin edirik.
Və proqramı çalışdırıb görünüşünə nəzər yetirək



Və self.video eləcədə toolbara əlavələrimiz göründü.

Novbəti kodlarımzda media faylı üçün irəli geri hərəkətini tətbiqinə slider çubuğu əlavə edək

```
#---------Slider-------------------------
self.slider=Phonon.SeekSlider(self)
self.slider.setMediaObject(self.media)
self.slider.setGeometry(20,460,660,20)
```

Biz əvvəlki bəhslərdə keçdiyimiz slider çubuğuna bənzəsədə,bu Phonon modulunun içində metod və funksiyaları ilə bərabər gəlir.self.slider dəyişəninə atıb yerləşmə ölçülərini yazıb proqramı çalışdırırıq.

Daha sonra Phonon modulu ilə bərabər gələn volume çubuğunu əlavə edək.Çubuğu əlavə etməmişdən öncə əvvəlinə etiket əlavə edək

```
#--------Label -----------------
 self.label=QLabel(self)
 self.label.move(20,495)
 self.label.setText("<font color='White'>Volume :</font>")
```

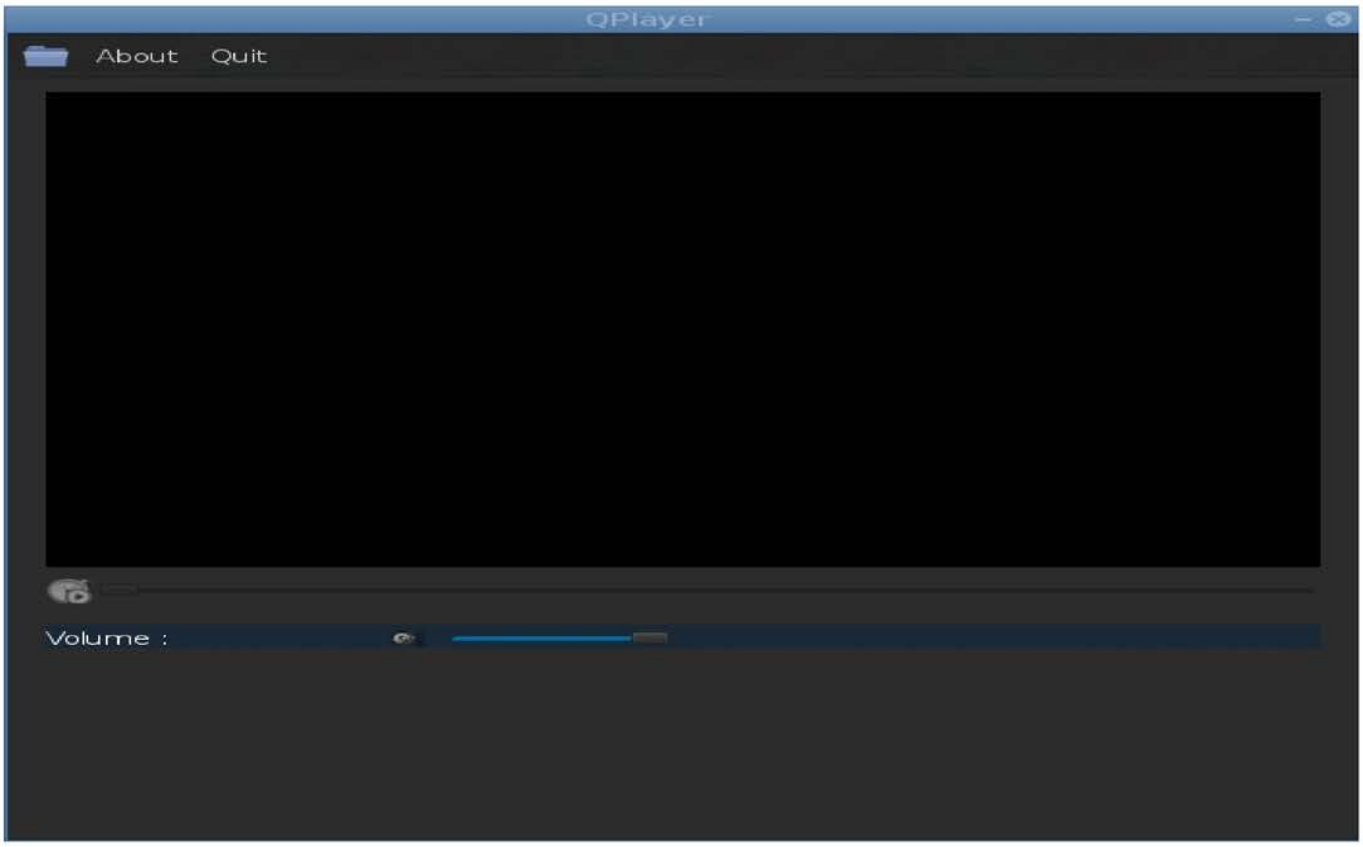
Etiketin və səs çubuğuna frame əlavə edək

```
 #--------Frame-----------------------
 self.frame=QFrame(self)
 self.frame.setGeometry(20,498,660,20)
 self.frame.setStyleSheet("background-color:#1A2938;")
```

və ardından səs çubuğunu tərtib etmək üçün kodları yazaq

```
self.volumeslider=Phonon.VolumeSlider(self)
self.volumeslider.setAudioOutput(self.audio)
self.volumeslider.setSizePolicy(QSizePolicy.Minimum,QSizePolicy.Minimum)
self.volumeslider.setGeometry(195,500,150,20)
```

Ekran görünüşü



Və növbəti, düymələri əlavə edək

```
 #-----------Button play--------------------
        self.button_play=QPushButton('Play',self)
        self.button_play.move(20,530)
        self.button_play.clicked.connect(self.playing)
        self.button_play.setStyleSheet('background-color:#070D14;')
#-------Button pause------------------

        self.button_paus=QPushButton('Pause',self)
        self.button_paus.move(130,530)
       #self.button_paus.clicked.connect(self.pausing)
        self.button_paus.setStyleSheet('background-color:#D73434;')
        #----------Button FullScreen watch---------------------
        self.button_full=QPushButton('Fullscreen',self)
        self.button_full.move(240,530)
      # self.button_full.clicked.connect(self.fullscreen)
        self.show()
```

və düymələri əlavə edib funksiyalar bölməsinə keçək.Toolbar a əlavə etdiyimiz open iconu çağırdığı open_file funksiyasını yazaq.

```
def open_file(self):
        if self.media.state() == Phonon.PlayingState:
            self.media.play()
            self.button_play.setEnabled(True)

        else:
            path = QFileDialog.getOpenFileName(self)
            if path:

                self.media.setCurrentSource(Phonon.MediaSource(path))
```

Funksiya daxilində QfileDialog klasından istifadə edərək faylı alıb self.media.setCurrentSource(Phonon.MediaSource(path)) ifadəsilə video vidjetə ötürürük.

Bu kodları toolbarda olan open -çağırdığı üçün ardıq open signalı qarşısından #-işarəsini götürə bilərik.
Daha sonra
düymələrin eləcədə slider çubuqların fəaliyyətini tənzimlək üçün

```
  def statech(self,state):
      try:
          if state == Phonon.PlayingState:
              self.button_play.setEnabled(False)
          elif (state != Phonon.LoadingState and state !=
Phonon.BufferingState):
              self.button_play.setEnabled(True)

          if state == Phonon.ErrorState:
              pass
      except AttributeError:
          pass
```

kodlarını try xəta blokuna alırıq.if operatoru ilə medianın oxunması zamanı (PlayingState) play düyməsini deaktiv etməyi əmr edirik.elif operatoru ilə isə faylın yüklənilməsi və ya xəta nisbətində olduğu zaman buttonu aktiv edirik.self.button_play.setEnabled(True)
Gələn errorlara isə pass deyib keçirik.Siz əlavə xətaları qabaqlamaq üçün except bölməsinə artıra bilərsiniz.

Daha sonra

```
def exiting(self):
      self.media.stop()
      self.close()
```

kodlarını toolbardakı quit -üçün hazırlayırıq.Yəni istifadəçi quit sıxarsa yuxarıdakı funksiya aktiv olacaq.medianı stop edərək ardından ümumi pəncərə bağlanacaq.
Növbəti kodlarımız

```
def playing(self):
      self.media.play()
      self.button_play.setFocusPolicy(False)
```

play düyməsidir.Yad kodlar olmadığı üçün medianı play vəziyyətinə salıb play düyməsini deaktiv edirik.

```
def pausing(self):
      self.media.pause()
```

funksiyası ilə pause düyməsini aktivə edirik.Yəni mediyanı olduğu yerdən pauz nöqtəsində tutmaq

```
def fullscreen(self):
      self.video.setGeometry(20,50,680,680)
```

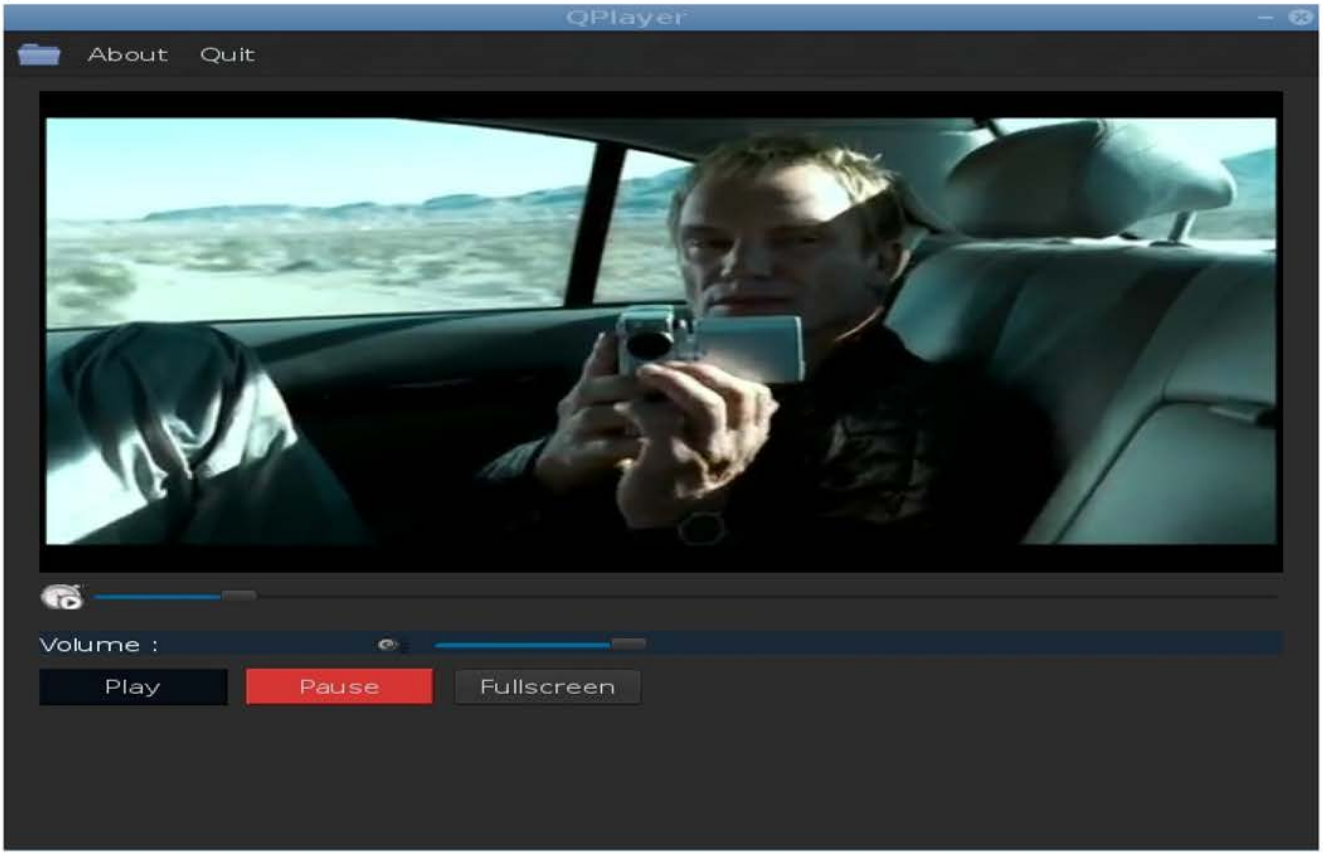
```
        self.slider.move(20,560)
        self.volumeslider.move(195,585)
        self.label.move(20,580)
        self.frame.move(20,585)
        self.button_play.move(20,630)
        self.button_paus.move(130,630)
        self.button_full.move(240,630)
```

Yuxarıdakı kodlarımız isə full screen düyməsi üçün bəsid kodlardır.

Və self.show

```
#------------Execute all function(app)-----------------
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

kodlarımızı yazırıq.Ardından yuxarıda olan #-işarələrini qaldırıb proqramı çalışdırırıq.



Proqrama bəzi əlavələr etmişəm. yuxarıda qeyd etdiyim github hesabıma daxil olub görə bilərsiniz.

QDockWidget

Vidjet bir frame təəssüratı bağışlayır.Daxilində digər vidjetləri yerləşdirir.Vidjet pəncərə daxilində həm də növbəti bir pəncərə açır,yəni pəncərənin bağlana biləcəyi paramaetrləri yuxarı hissəsində daşıyır.Vidjet dəyişilə biləndir,dartıb istənilən nöqtəyə yerləşdirmək olur.
Vidjetin görünüşünə baxaq

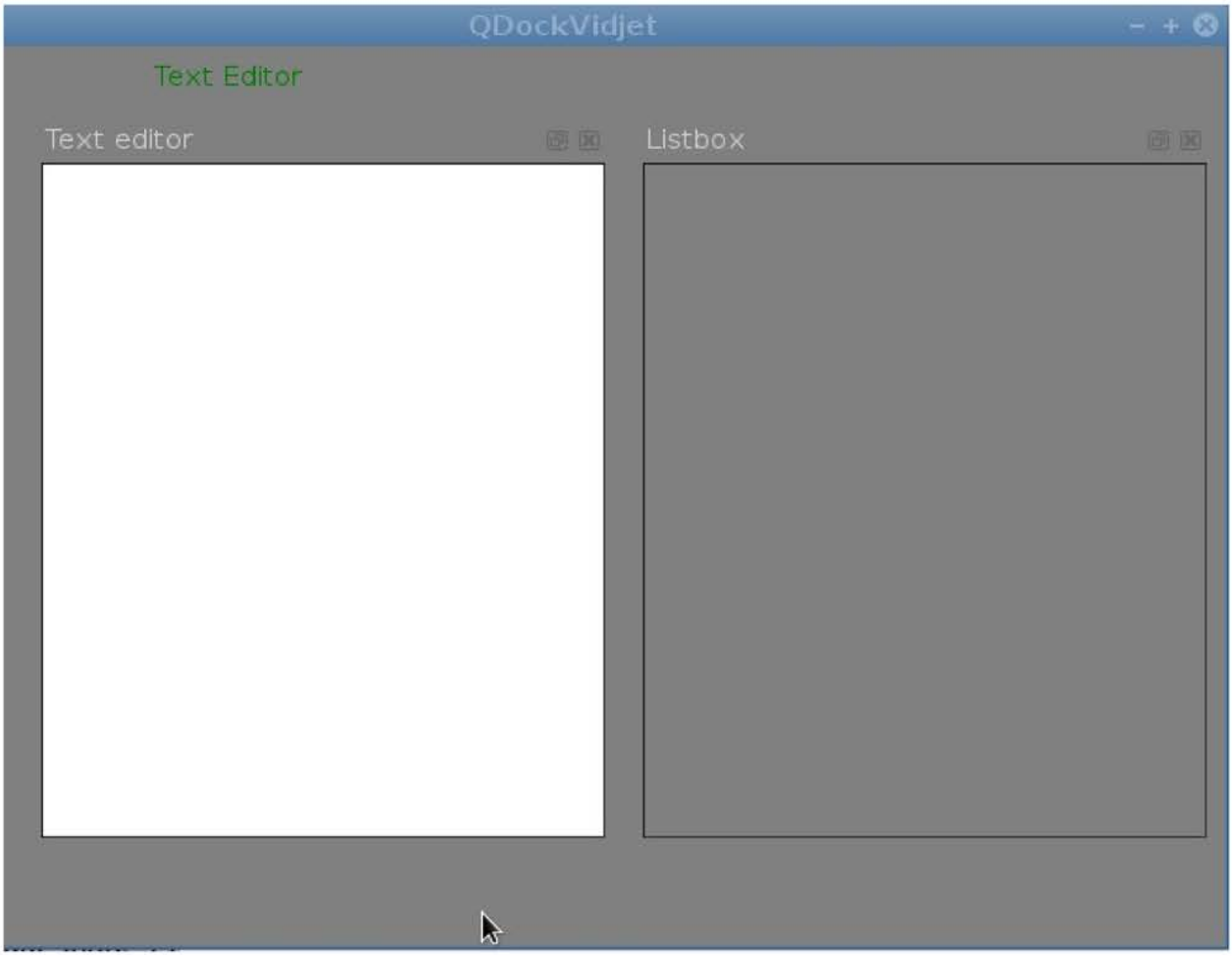
```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QDockVidjet')
        self.setGeometry(400,200,650,500)
        self.setStyleSheet('background-color:white;')
        self.qdock=QDockWidget("Listbox",self)
        self.qdock.setGeometry(300,40,350,400)
        self.listbox=QListWidget()
        self.listbox.setStyleSheet('background-color:black;')
        self.qdock.setWidget(self.listbox)
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

self.qdock=QDockWidget("Listbox",self) ifadəsilə vidjeti tərtib etdik və setGeometry metodu ilə pəncərədə yerləşdirdik.Daha sonra list qutusunu təyin edib vidjetin daxilinə vidjet metodu olan setWidget ilə yerləşdirdik.

```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
```

```
        super(window, self).__init__(parent)
        self.setWindowTitle('QDockVidjet')
        self.setGeometry(400,200,650,500)
        self.setStyleSheet('background-color:gray;')
        #-----------------------------
        self.label=QLabel(self)
        self.label.setText("<font color='Green'>Text Editor</font>")
        self.label.move(80,2)
        #---------------------------------
        self.tdock=QDockWidget("Text editor",self)
        self.tdock.setGeometry(20,40,300,400)
        self.textedit=QTextEdit()
        self.textedit.setGeometry(20,40,300,400)
        self.textedit.setStyleSheet('background-color:white;')
        self.tdock.setWidget(self.textedit)
        #-------------------------------
        self.listbox=QListWidget()
        self.listbox.setStyleSheet('background-color:gray;')
        #-------------------------------
        self.qdock=QDockWidget("Listbox",self)
        self.qdock.setGeometry(340,40,300,400)
        self.qdock.setWidget(self.listbox)
        #-------------------------------
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ekran görünüşü



Vidgeti pəncərə daxilində yerləşdirsəkdə,vidjetləri kənara çıxartmaq,eləcədə yerini dəyişmək mümkündür.

Kənara çıxarmağın qarşısını almaq üçün biz setFloating() metodundan istifadə edə cəyik.Metod true və False parametrlərini alır

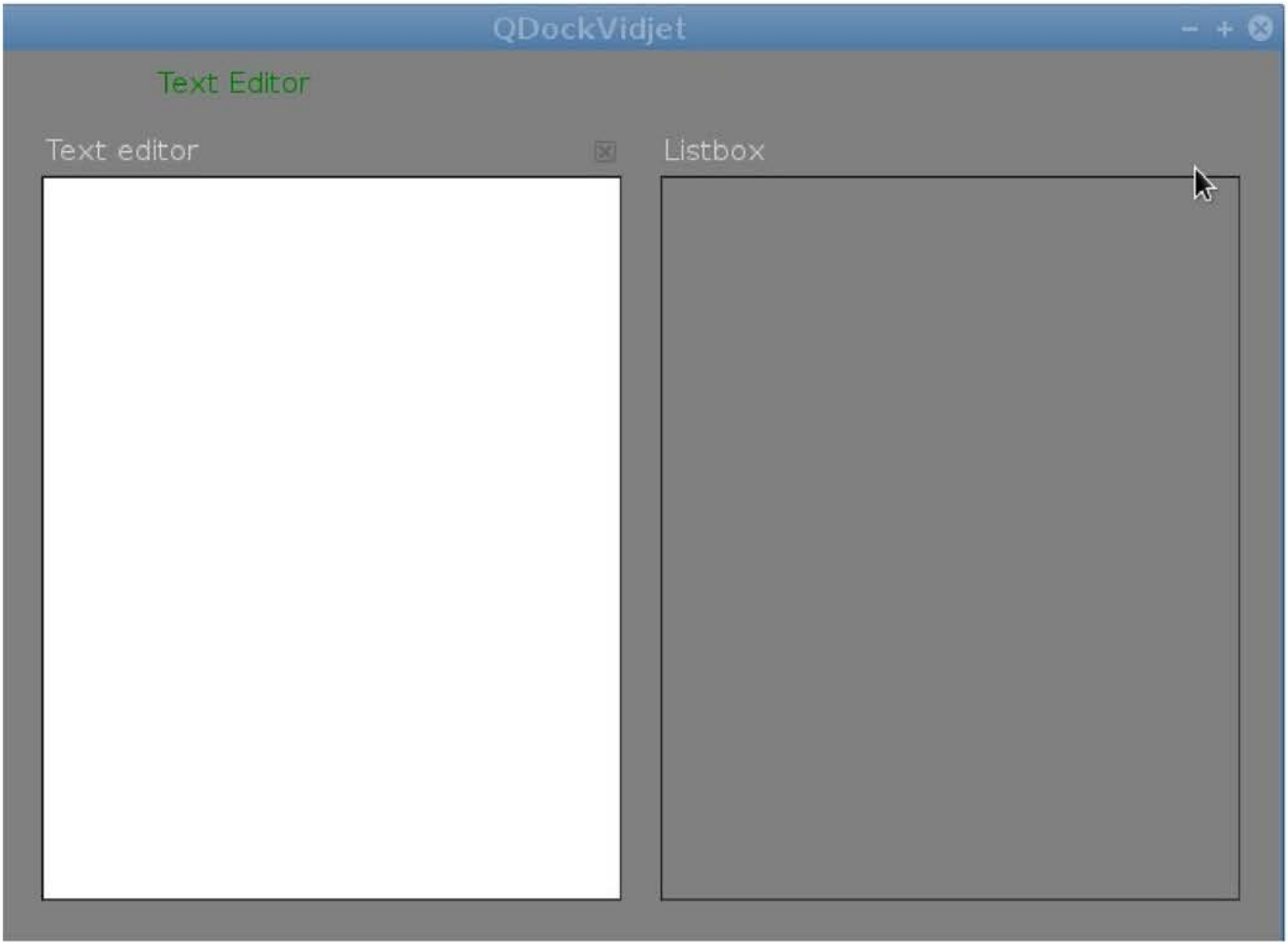
```
# -*-coding: utf-8 -*-
from PyQt4.QtGui import*
from PyQt4.QtCore import*
import sys
class window(QMainWindow):
    def __init__(self, parent=None):
        super(window, self).__init__(parent)
        self.setWindowTitle('QDockVidjet')
        self.setGeometry(400,200,650,500)
        self.setStyleSheet('background-color:gray;')
        #----------------------------
```

```
        self.label=QLabel(self)
        self.label.setText("<font color='Green'>Text Editor</font>")
        self.label.move(80,2)
        #---------------------------------
        self.tdock=QDockWidget("Text editor",self)
        self.tdock.setGeometry(20,40,300,400)
        self.textedit=QTextEdit()
        self.textedit.setGeometry(20,40,300,400)
        self.textedit.setStyleSheet('background-color:white;')
        self.tdock.setWidget(self.textedit)
        self.tdock.setFloating(False)
        #---------------------------------
        self.listbox=QListWidget()
        self.listbox.setStyleSheet('background-color:gray;')
        #-------------------------------
        self.qdock=QDockWidget("Listbox",self)
        self.qdock.setGeometry(340,40,300,400)
        self.qdock.setWidget(self.listbox)
        #-------------------------------
        self.show()
if __name__ == '__main__':
    app = QApplication([])
    gui = window()
    app.exec_()
```

Ümumiyyətlə vijetlərin hərəkətini tənzimləmək olur.Bunun üçün setFeatures() metodundan istifadə edə bilərik.İstifadə modeli
self.dock.setFeatures(QDockWidget.parametr)
Metod şağıdakı parametrləri alır

DockWidgetClosable -müdaxiləni bağlamaq
DockWidgetMovable -hərəkətli
DockWidgetFloatable -vidjet üzərindən x düyməsini disable etmək
DockWidgetVerticalTitleBar -vertikal istiqamətdə
NoDockWidgetFeatures -vidjetin görünməməsini aktiv etmək

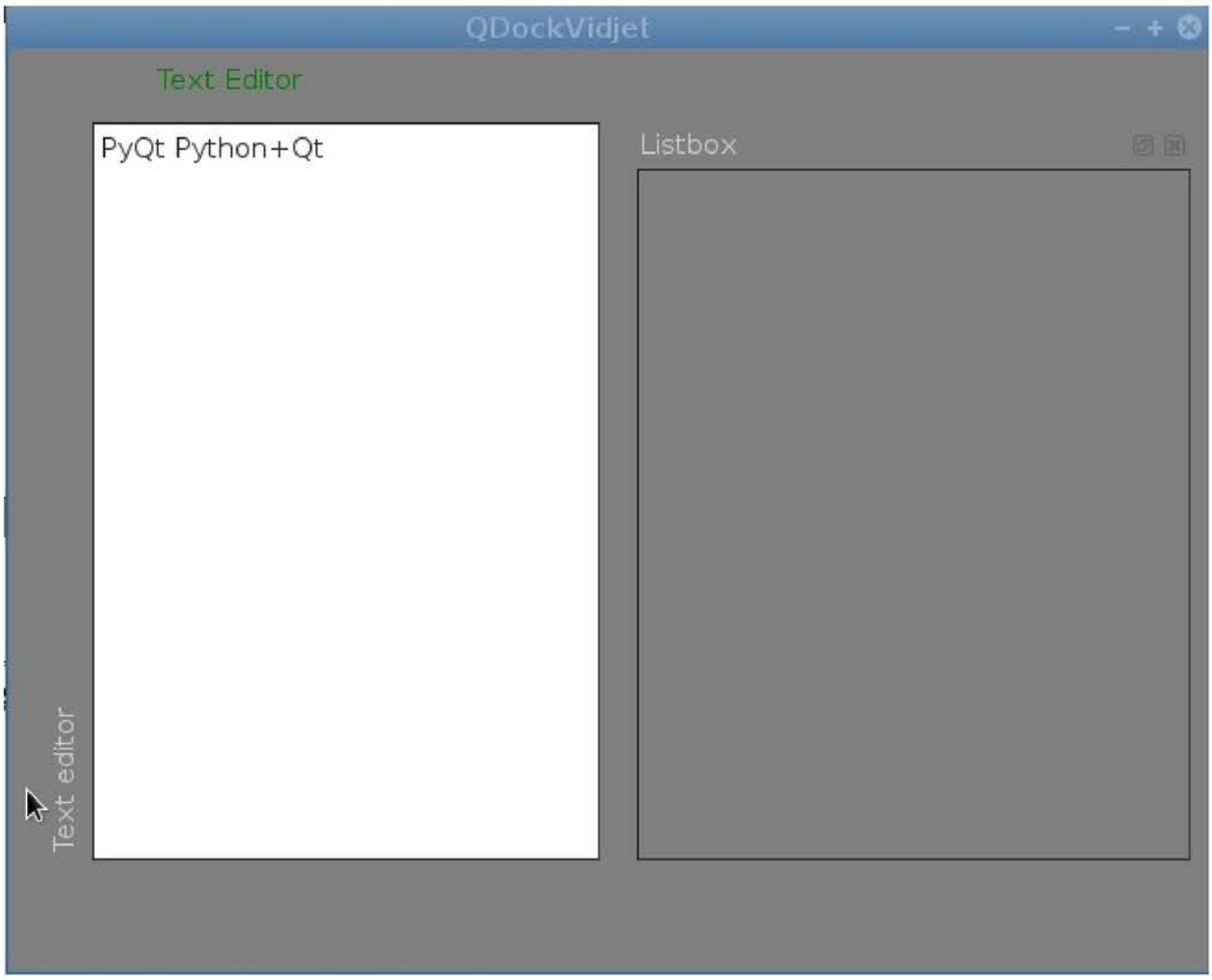
```
self.qdock=QDockWidget("Listbox",self)
self.qdock.setGeometry(340,40,300,400)
self.qdock.setWidget(self.listbox)
self.qdock.setFeatures(QDockWidget.NoDockWidgetFeatures)
```



Listbox üzərindən vidjeti aradan qaldırdıq
self.qdock.setFeatures(QDockWidget.NoDockWidgetFeatures)

```
self.tdock=QDockWidget("Text editor",self)
self.tdock.setGeometry(20,40,300,400)
self.textedit=QTextEdit()
self.textedit.setGeometry(20,40,300,400)
 self.textedit.setStyleSheet('background-color:white;')
 self.tdock.setWidget(self.textedit)
 self.tdock.setFeatures(QDockWidget.DockWidgetVerticalTitleBar)
```

Vertical istiqamətdə görünüş
self.tdock.setFeatures(QDockWidget.DockWidgetVerticalTitleBar)



Digər parametrləri verib nəticələrini test edin.

Vidgetin növbəti metoduna nəzər salaq

setAllowedAreas()
Metod aşağıdakı parametrləri alır

LeftDockWidgetArea
RightDockWidgetArea
TopDockWidgetArea
BottomDockWidgetArea
NoDockWidgetArea

Istifadə modeli self.qdock.setAllowedAreas(Qt.LeftDockWidgetArea)

Mətn editoru

Sadə mətn editorlarından notepad++ ,mousepad,leafpad,vim(vim gui) və s misal çəkmək olar.Təbiki qeyd etdiyim editorlar daha irəli səviyyədə yazılmış kodlardan ibarətdir.Biz aşağıdakı kodlarla bəsid mətn editoru hazırlayacağıq

Kodlar daha çox sətirə malik olduğundan aşağıdakı ünvandan yükləyib nəzər yetirə bilərsiniz.

https://github.com/RashadGarayev/XEditor