Руководство по PyQt5

PyQt5 является одним из наиболее часто используемых модулей для создания **GUI приложений** в Python, и это связанно с его простотой, о которой вы узнаете далее.

Еще одна замечательная особенность, которая вдохновляет разработчиков пользоваться PyQt5 – это **PyQt5 Designer**, благодаря которому можно создавать сложные GUI приложения достаточно быстро. Вам нужно только перетаскивать свои виджеты для создания собственной формы. У нас есть готовый сборник **54 уроков** по другому фреймворку wxPython.

Готовые уроки по PyQt5

- Создаем простой калькулятор в PyQt5
- Создаем игру Сапёр на PyQt5
- История курса рубля на PyQt5 + XML от ЦБ РФ

Другие фреймворки

- wxPython
- Tkinter
- PyCairo

В данном руководстве по PyQt5, я буду использовать Python 3.6 на Ubuntu и предположение, что вы уже имеете базовое представление о Python.

Звучит замечательно! Начнем с установки PyQt5, и затем перейдем к тому, как разрабатывать GUI приложение с примерами.

Краткое содержание

Установка PyQt5

Существует две версии PyQt5: коммерческая и **бесплатная версия** GPL, которой мы будем пользоваться в этом руководстве.

Есть два способа установки PyQt5:

- Установка PyQt5 через pip
- Установка PyQt5 из исходников на Linux

Установка PyQt5 через pip

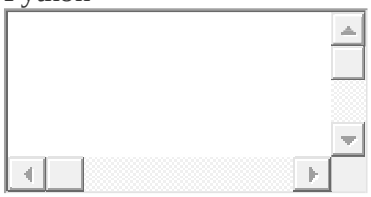
Чтобы установить **PyQt5 при помощи pip**, выполните следующую команду:

Python

```
1 sudo pip3 install PyQt5
```

Чтобы убедиться в том, что установка прошла успешно, запустите следующий код:

Python



```
1 import PyQt5
```

Если не возникло ни одной ошибки, это значит, что вы успешно **установили PyQt5**. В случае, если ошибки возникли, возможно это связанно с тем, что вы используете версию Python, которая не поддерживается.

Установка PyQt5 из исходников на Linux

Для установки **PyQt5 из исходника**, вам нужно сделать следующее:

1. Установить SIP;
2. Скачать исходник PyQt5;
3. Настроить и установить.

Как вы возможно знаете, PyQt5 связывает Python с популярной библиотекой **Qt**, которая написана на C++.

Инструмент, который создает эту связь, называется **SIP**. Так что для **установки PyQt5 из исходника**, вам для начала нужно установить SIP.

Для установки SIP, запустите следующую команду:

Python

```
1 sudo pip3 install pyqt5-sip
```

Теперь вы можете загрузить и установить исходник PyQt5.

Исходники

Скачать исходник PyQt5 можно
отсюда: https://www.riverbankcomputing.com/software/pyqt/download5

Внимание: На момент прочтения статьи возможно появилась новая версия которая отличается от той что в скрине. Версия на текущий момент **5.11.3**, вы должны самостоятельно скопировать ссылку с сайта и предоставить её в **wget**. Заметьте, что обновить версию придется во всех ниже предоставленных командах.

Python

```
1 wget https://sourceforge.net/projects/pyqt/files/PyQt5/PyQt-5.11.3/PyQt5_gpl-5.11.3.tar.gz
2 tar -xvzf PyQt5_gpl-5.11.3.tar.gz
3 cd PyQt5_gpl-5.11.3
```

Мы распаковали сжатый исходник, теперь запустите следующие команды внутри корня папки:

Python

```
1 sudo python3 configure.py
2 sudo make
3 sudo make install
```

Чтобы убедиться в том, что все прошло гладко, попробуйте импортировать модуль PyQt5 так же, как мы делали все раньше. Все должно пройти хорошо.

Установка PyQt5 из исходников на Windows

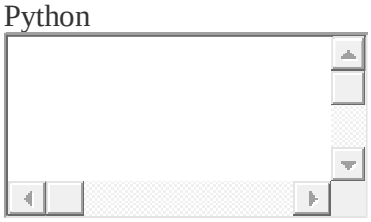
Скачивайте и распакуйте архив с сайта которого мы указали выше.

Так как SIP требует компилятор **GCC**, вам нужно установить **MinGW**, который является портом Windows для компилятора Linux, GCC.

Единственное, что нужно поменять — это момент конфигурации. Вам нужно сообщить Python о платформе.

Это можно выполнить следующим образом:

Python



```
1 python configure.py --platform win32-g++
2 make
3 make install
```

Поздравляем! Вы успешно **установили PyQt5** из исходника.

Установка PyQt5 Designer

Есть два способа создания GUI приложений при помощи PyQt5:

1. Дизайн виджетов при помощи кода;
2. Использование PyQt5 Designer.

В этом руководстве мы используем PyQt5 Designer, который упрощает процесс на столько, что вы можете выполнить большой объем работы за секунды.

Дизайнер PyQt5 поставляется вместе с набором инструментов. Для его установки, вам нужно установить эти инструменты.

Python

```
1 $ pip3 install pyqt5-tools
```

Где находится дизайнер PyQt5?

После удачной установки, вы можете найти дизайнер PyQt5 здесь:

Shell

```
1 C:\Program Files\Python36\Lib\site-packages\pyqt5-tools\
```

Кстати, если вы установили только для своего текущего пользовательского аккаунта, вы найдете дизайнер PyQt5 вот здесь:

Shell

```
1 C:\Users\PythonUser\AppData\Local\Programs\Python\Python36-32\Lib\site-packages\ pyqt5-tools\
```

Вы можете создать **короткий путь** для него, вместо того, чтобы постоянно переходить к этому расположению для запуска дизайнера PyQt5.

Как использовать дизайнер PyQt5

Откройте **designer.exe** и увидите диалоговое окно, спрашивающее о том, какую форму шаблона вы предпочитаете.

Существует пять видов доступных шаблонов:

- **Диалоговое окно с кнопками внизу** (*Dialog with Buttons Bottom*): создает форму с кнопками OK и Cancel в правом нижнем углу формы.
- **Диалоговое окно с кнопками справа** (*Dialog with Buttons Right*): создает форму с кнопками OK и Cancel в верхнем правом углу формы.
- **Диалоговое окно без кнопок** (*Dialog without Buttons*): создает пустую форму;
- **Главное окно** (*Main Window*): создает окно с панелью меню и набором инструментов. Унаследовано из QmainWindow;
- **Виджет** (*Widget*): создает виджет, наследуемый из класса Qwidget. Отличается от диалоговых шаблонов тем, что они наследуются из класса QBialog

Итак, у нас есть три типа шаблонов. В чем между ними разница?

Разница между QDialog, QMainWindow, и Qwidget

- **QWidget** – это базовый класс для всех GUI Элементов в PyQt5;
- **QDialog** используется при запросе чего-либо у пользователя, например, запросы о принятии или отклонении чего-нибудь. Основан на Qwidget.
- **QMainWindow** – самый большой шаблон, где вы можете разместить вашу панель инструментов, меню, статус и другие виджеты. Не имеет встроенных кнопок разрешения, таких как в QDialog.

Загрузка .ui против конвертации .ui в .py

В данном руководстве мы используем **PyQt5 Designer**, но перед тем, как мы пойдем дальше, давайте рассмотрим, как еще мы можем использовать сгенерированный файл из **PyQt5 Designer**.

Нам нужно открыть PyQt5 Designer, выбрать шаблон **Main Window** и нажать кнопку **create**.

Далее в файловом меню (File), нажимаем сохранить. PyQt5 Designer экспортирует вашу форму в XML с расширением **.ui**.

Для использования этого дизайна, у вас есть два способа:

- Загрузить файл .ui в ваш код Python;
- **Конвертировать файл .ui** в файл .py при помощи **pyuic5**;

Загрузка .ui файла в ваш код Python

Чтобы **загрузить файл .ui** в ваш код Python, вы можете использовать функцию **loadUI**() из uic вот так:

Python

```
1 from PyQt5 import QtWidgets, uic
2 import sys
3
4 app = QtWidgets.QApplication([])
5 win = uic.loadUi("mydesign.ui") # расположение вашего файла .ui
6
7 win.show()
8 sys.exit(app.exec())
```

Если вы запустите код, вы увидите окно, в котором есть только ярлык.

Это значит, что **ui файл** успешно загрузился!

Мы используем **sys.exit(app.exec())** вместо использования **app.exec()** напрямую, чтобы выслать правильный код статуса, родительский процесс, или процесс вызова.

Если вы использовали **app.exec()** напрямую, приложение отправит ноль, что говорит об успехе, и это произойдет даже если приложение упадет.

Конвертация файла .ui в файл .py при помощи pyuic5

Давайте попробуем еще один способ и конвертируем файл **.ui** в код Python:

Shell

```
1 $ pyuic5 mydesign.ui -o mydesign.py
```

Да! Был создан новый файл под названием **mydesign.py**. Теперь, импортируем этот файл, чтобы показать его в окне.

Python

```
 1  from PyQt5 import QtWidgets
 2  from mydesign import Ui_MainWindow  # импорт нашего сгенерированного файла
 3  import sys
 4
 5
 6  class mywindow(QtWidgets.QMainWindow):
 7      def __init__(self):
 8          super(mywindow, self).__init__()
 9          self.ui = Ui_MainWindow()
10          self.ui.setupUi(self)
11
12
13 app = QtWidgets.QApplication([])
14 application = mywindow()
15 application.show()
16
17 sys.exit(app.exec())
```

Если запустите этот код, вы должны увидеть то же окно, что было в первом методе.

Преимущество использования второго метода — это автоматическое завершение, которое выполнит IDE, так как все ваши виджеты импортированы. В то же время, пользуясь первым методом, вы просто **загружаете файл .ui** и вам нужно обращать внимание на названия ваших виджетов.

Еще одно преимущество использования второго метода. **Скорость**: вам не нужен парсинг XML для загрузки UI.

Так что мы можем сказать, что конвертация файла .ui в файл .py – безопаснее для кодинга и быстрее для загрузки.

Настало время закатить рукава и поиграть с **виджетами PyQt5**.

Виджет QLabel

Для внесение виджета **QLabel** в вашу форму, выполните следующее:

- Откройте **PyQt5 Designer** и выберите шаблон **Main Window**;
- Перетяните виджет ярлыка из бокса слева;

Сохраните дизайн в файл как **qlabel.ui** и конвертируйте его в файл **qlabel.py**. Теперь поработаем с ярлыком виджета при помощи кода.

Shell

```
1 pyuic5 qlabel.ui -o qlabel.py
```

Результат:

Меняем шрифт QLabel

Чтобы **поменять шрифт QLabel**, используйте метод **setFont**() и передайте ему **QFont** следующим образом:

Python

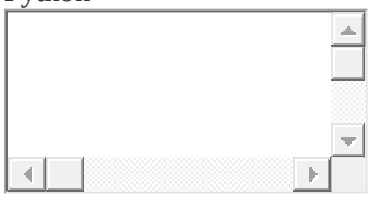
```
from PyQt5 import QtWidgets, QtGui
from mydesign import Ui_MainWindow
import sys


class mywindow(QtWidgets.QMainWindow):

    def __init__(self):
        super(mywindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.label.setFont(
            QtGui.QFont('SansSerif', 30)
        ) # Изменение шрифта и размера


app = QtWidgets.QApplication([])
application = mywindow()
application.show()

sys.exit(app.exec())
```

После того, как запустите этот код, обратите внимание на то, что ярлык возникает некорректно, так как размер — меньше, чем размер шрифта, который мы используем. Так что нам нужно установить размер ярлыка.

Меняем размер QLabel

Чтобы поменять размер **QLabel**, вам нужно указать его геометрию при помощи метода **setGeometry**(), вот так:

Python



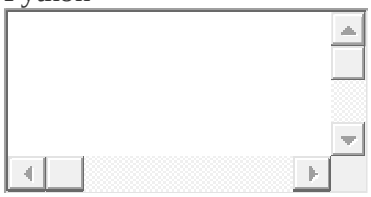
```
from PyQt5 import QtWidgets, QtGui,QtCore
from mydesign import Ui_MainWindow
import sys

class mywindow(QtWidgets.QMainWindow):

    def __init__(self):
        super(mywindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.label.setFont(
            QtGui.QFont('SansSerif', 30)
        )

        self.ui.label.setGeometry(
            QtCore.QRect(10, 10, 200, 200)
        ) # изменить геометрию ярлыка

```

```
20
21 app = QtWidgets.QApplication([])
22 application = mywindow()
23 application.show()
24
25 sys.exit(app.exec())
```

Меняем текст в QLabel

Чтобы изменить **текст в QLabel**, вы можете использовать метод **setText**(), вот так:

Python



```
 1  from PyQt5 import QtWidgets, QtGui,QtCore
 2  from qlabel import Ui_MainWindow
 3  import sys
 4
 5  class mywindow(QtWidgets.QMainWindow):
 6
 7      def __init__(self):
 8          super(mywindow, self).__init__()
 9          self.ui = Ui_MainWindow()
10          self.ui.setupUi(self)
11
12          self.ui.label.setFont(
13              QtGui.QFont('SansSerif', 30)
14          )
15
16          self.ui.label.setGeometry(
17              QtCore.QRect(10, 10, 200, 200)
18          )
19
20          self.ui.label.setText("PyScripts") # Меняем текст
21
22
23 app = QtWidgets.QApplication([])
24 application = mywindow()
25 application.show()
26
27 sys.exit(app.exec())
```

Именно на столько все просто! Давайте рассмотрим другие виджеты.

Виджет QLineEdit

Виджет QLineEdit – это редактируемое поле, где вы можете принимать данные от пользователя. **LineEdit** содержит множество методов, с которыми можно работать.

Я создам новый дизайн **qline.ui**, используя **дизайнер PyQt5** и внесу шесть виджетов **QLineEdit** и экспортирую его в файл **qline.py**.

Python

```
1 pyuic5 qline.ui -o qline.py
```

Содержимое файла qline.py после конвертации:

qline.py
Python

```
 1  # -*- coding: utf-8 -*-
 2
 3  # Form implementation generated from reading ui file 'qline.ui'
 4  #
 5  # Created by: PyQt5 UI code generator 5.11.3
 6  #
 7  # WARNING! All changes made in this file will be lost!
 8
 9  from PyQt5 import QtCore, QtGui, QtWidgets
10
11 class Ui_MainWindow(object):
12     def setupUi(self, MainWindow):
13         MainWindow.setObjectName("MainWindow")
14         MainWindow.resize(785, 600)
15         self.centralwidget = QtWidgets.QWidget(MainWindow)
16         self.centralwidget.setObjectName("centralwidget")
17         self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
18         self.lineEdit.setGeometry(QtCore.QRect(10, 10, 291, 31))
19         self.lineEdit.setObjectName("lineEdit")
20         self.lineEdit_2 = QtWidgets.QLineEdit(self.centralwidget)
21         self.lineEdit_2.setGeometry(QtCore.QRect(10, 50, 291, 31))
22         self.lineEdit_2.setObjectName("lineEdit_2")
23         self.lineEdit_3 = QtWidgets.QLineEdit(self.centralwidget)
24         self.lineEdit_3.setGeometry(QtCore.QRect(10, 90, 291, 31))
25         self.lineEdit_3.setObjectName("lineEdit_3")
26         self.lineEdit_4 = QtWidgets.QLineEdit(self.centralwidget)
27         self.lineEdit_4.setGeometry(QtCore.QRect(10, 130, 291, 31))
28         self.lineEdit_4.setObjectName("lineEdit_4")
29         self.lineEdit_5 = QtWidgets.QLineEdit(self.centralwidget)
30         self.lineEdit_5.setGeometry(QtCore.QRect(10, 170, 291, 31))
31         self.lineEdit_5.setObjectName("lineEdit_5")
32         self.lineEdit_6 = QtWidgets.QLineEdit(self.centralwidget)
33         self.lineEdit_6.setGeometry(QtCore.QRect(10, 210, 291, 31))
34         self.lineEdit_6.setObjectName("lineEdit_6")
35         MainWindow.setCentralWidget(self.centralwidget)
36         self.menubar = QtWidgets.QMenuBar(MainWindow)
37         self.menubar.setGeometry(QtCore.QRect(0, 0, 785, 25))
38         self.menubar.setObjectName("menubar")
39         MainWindow.setMenuBar(self.menubar)
40         self.statusbar = QtWidgets.QStatusBar(MainWindow)
41         self.statusbar.setObjectName("statusbar")
42         MainWindow.setStatusBar(self.statusbar)
43
44         self.retranslateUi(MainWindow)
45         QtCore.QMetaObject.connectSlotsByName(MainWindow)
46
47     def retranslateUi(self, MainWindow):
48         _translate = QtCore.QCoreApplication.translate
```

```
49        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
```

Давайте познакомимся с методами **QLineEdit**:

Python

```
 1  from PyQt5 import QtWidgets, QtCore
 2  # Импортируем наш файл
 3  from qline import Ui_MainWindow
 4  import sys
 5
 6
 7  class mywindow(QtWidgets.QMainWindow):
 8      def __init__(self):
 9          super(mywindow, self).__init__()
10          self.ui = Ui_MainWindow()
11          self.ui.setupUi(self)
12
13          # Меняем текст
14          self.ui.lineEdit.setText("Добро пожаловать на PythonScripts")
15
16          # указать максимальную длину
17          self.ui.lineEdit_2.setMaxLength(10)
18
19          # ввод пароля
20          self.ui.lineEdit_3.setEchoMode(QtWidgets.QLineEdit.Password)
21          # только чтение без изменения содержимого.
22          self.ui.lineEdit_4.setReadOnly(True)
23
24          # меняем цвет вводимого текста
25          self.ui.lineEdit_5.setStyleSheet("color: rgb(28, 43, 255);")
26
27          # изменение цвета фона QLineEdit
28          self.ui.lineEdit_6.setStyleSheet("background-color: rgb(28, 43, 255);")
29
30
31 app = QtWidgets.QApplication([])
32 application = mywindow()
33 application.show()
34
35 sys.exit(app.exec())
```

Результат:

- Для 1-го QLineEdit, мы изменили текст, при помощи метода **setText**().
- Для 2-го QLineEdit, мы установили максимум доступных 10-и символов, так что более 10-и приниматься не будут.
- Для 3-го QLineEdit, мы установили режим пароля, так ваши введенные данные отображаются как звездочки;
- Для 4-го QLineEdit, мы установили режим «**только для чтения**», так что редактировать контент не представляется возможным.
- Для 5-го QLineEdit мы изменили цвет шрифта при помощи метода **setStyleSheet**() и указали нужный цвет с помощью CSS (как и для обычной веб страницы).
- Для 6-го QLineEdit мы изменили цвет фона при помощи метода **setStyleSheet**() и CSS.

Метод setStyleSheet()

Метод **setStyleSheet**() может быть использован с виджетами PyQt5 для изменения стилей.

Вы можете изменить следующие параметры, пользуясь методом **setStyleSheet**():

- Размер и тип шрифта;
- Цвет текста;
- Цвет заднего фона;
- Цвет верхней границы;
- Цвет нижней границы;

- Цвет левой границы;
- Цвет правой границы;
- Цвет выделения;
- Цвет выделения заднего фона.

Это наиболее важные значения, которые можно передать методу setStyleSheet().

Виджет QPushButton

Большая часть ваших программ Python будут содержать виджет **QPushButton**. Вы нажимаете кнопку, и какой-нибудь код выполняется.

Если у вас имеется опыт в программировании, вы могли слышать об **обработке событий**, где вы взаимодействуете с виджетом и функция выполняется.

Суть идеи сохранилась такой же и в PyQt5, только определения немного отличаются.

Событие клика в PyQt5 называется сигналом, и метод, который будет выполняться, называется **слот**.

Так что при нажатии на **QPushButton**, сигнал издается. Названием сигнала в данном случае, является **clicked**().

Чтобы связать сигнал со слотом, вам нужно использовать метод **connect**(), что вы сейчас и увидите.

Этот процесс обработки события продолжает работать до тех пор, пока вы не закроете вашу форму, или главный виджет.

Давайте создадим форму **myform.ui** при помощи QLabel и **QPushButton** и экспортируем её в файл **myform.py**.

Экспортируем **myform.ui** в **myform.py**

Shell

```
1 pyuic5 myform.ui -o myform.py
```

Сейчас, мы подключим сигнал **clicked**() к слоту при помощи метода **connect**(), вот так:

Python

```
1 self.ui.pushButton.clicked.connect(self.btnClicked)
```

Здесь, **btnClicked** – это слот, или функция, которая будет выполнена после того, как вы кликните на **QPushButton**.

Итак, ваш код будет выглядеть следующим образом:

Python

```
 1  from PyQt5 import QtWidgets
 2
 3  # Импортируем наш шаблон.
 4  from myform import Ui_MainWindow
 5  import sys
 6
 7
 8  class mywindow(QtWidgets.QMainWindow):
 9      def __init__(self):
10          super(mywindow, self).__init__()
11          self.ui = Ui_MainWindow()
12          self.ui.setupUi(self)
13          # подключение клик-сигнал к слоту btnClicked
14          self.ui.pushButton.clicked.connect(self.btnClicked)
15
16      def btnClicked(self):
17          self.ui.label.setText("Вы нажали на кнопку!")
18          # Если не использовать, то часть текста исчезнет.
19          self.ui.label.adjustSize()
20
21
22 app = QtWidgets.QApplication([])
23 application = mywindow()
24 application.show()
25
26 sys.exit(app.exec())
```

Результат:

Замечательно!

Визуальный редактор слота/сигнала

Мы видели, как подключать сигнал виджета к слоту при помощи метода **connect**(), но это не единственный способ.

На самом деле, для каждого виджета есть предопределенные слоты. Вы можете связать сигнал с любым предопределенным слотом, без необходимости кодить в дизайнере PyQt5.

Перетяните **QPushButton** и **QLineEdit** в вашу форму.

Нажмите F4 и перетяните курсор из QPushButton и отпустите его в верхней части QLineEdit. Чтобы вернуться в обычный режим, нажмите на F3.

Благодаря этому появится редактор сигнала/слота.

Слева находятся предопределенные сигналы, справа — предопределенные слоты. Скажем, нам нужно подключить сигнал clicked() с слотом «*очистки содержимого*«.

Выберите сигнал **clicked** слева и выберите **clear слот** справа и нажмите OK.

После выполнения подключений ваших сигналов и слотов, вы можете выйти из этого режима, нажав **ESC**, или **F3**.

Сейчас, если вы запустите эту форму, и нажмете **QPushButton**, то любой текст в **QLineEdit** будет очищен. Вы можете редактировать или удалять эти связи в панели редактора сигналов и слотов.

Сохраните форму как **signaledit.ui** и конвертируем её в **signaledit.py**:

Shell

```
1 pyuic5 signaledit.ui -o signaledit.py
```

Полученный файл импортируем в наш код:

Python

```
 1  from PyQt5 import QtWidgets
 2
 3  # Импортируем наш шаблон.
 4  from signaledit import Ui_MainWindow
 5
 6  import sys
 7
 8
 9  class mywindow(QtWidgets.QMainWindow):
10      def __init__(self):
11          super(mywindow, self).__init__()
12          self.ui = Ui_MainWindow()
13          self.ui.setupUi(self)
14
15
16  app = QtWidgets.QApplication([])
```

```
17 application = mywindow()
18 application.show()
19
20 sys.exit(app.exec())
```

Результат:

Как выпускать сигналы в PyQt5

Мы увидели, как работают сигналы и слоты. Все сигналы, с которыми мы работали, являются предопределенными для нас.

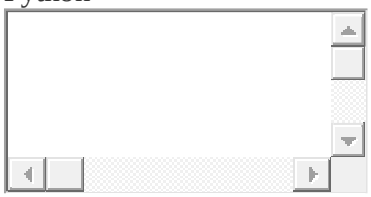
Но что на счет выпуска **собственного сигнала**?

Это очень просто! Вы можете сделать это, просто использовав класс **pyqtSignal**, как показано ниже:

- Определите ваше событие типом **pyqtSignal**;
- Вызовите метод **emit**() в том месте, где вы хотите, чтобы произошло событие.

Скажем, у нас есть класс **nut**, и мы хотим, чтобы сигнал **cracked** был выпущен.

Python



```
1  from PyQt5.QtCore import pyqtSignal, QObject
2
3
4  class nut(QObject):
5      cracked = pyqtSignal()
6
7      def __init__(self):
8          QObject.__init__(self)
9
10     def crack(self):
11         self.cracked.emit()
```

Как использовать сигнал

Сейчас мы сделаем наш пример более практичным, создаем экземпляр **класса nut** и выпуская сигнал **cracked**:

Python

```
from PyQt5.QtCore import pyqtSignal, QObject


class nut(QObject):
    cracked = pyqtSignal()

    def __init__(self):
        QObject.__init__(self)

    def crack(self):
        self.cracked.emit()


def crackit():
    print("hazelnut cracked!")


hazelnut = nut()
# подключение сигнала cracked к слоту crackit
hazelnut.cracked.connect(crackit)
hazelnut.crack()
```

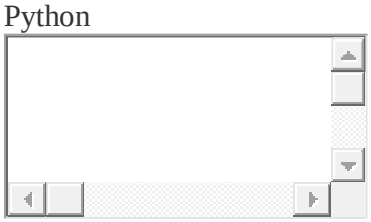
Сигнал cracked успешно выпущен.

Переопределение сигнала (события) в PyQt5

Иногда нам может понадобиться переопределить поведение по умолчанию для особенных событий или сигналов.

Давайте рассмотрим практичный пример для этого случая. Если вы хотите закрыть окно, когда пользователь нажимает на определенную клавишу, вы можете переопределить **keyPressEvent** внутри вашего главного окна следующим образом:

Python



```
def keyPressEvent(self, e):
    if e.key() == Qt.Key_F12:
        self.close()
```

Теперь, если пользователь нажмет клавишу **F12**, главное окно будет закрыто.

Здесь мы переопределили основной сигнал нажатия в главном окне и закрыли это окно.

Виджет QComboBox

Вместо того, чтобы разрешить пользователю вводить свои данные в QLineEdit, или любом другом редактируемом виджете, мы можем использовать **виджет QCombobBox**, чтобы дать список данных, из которого он сможет выбирать.

Давайте перетянем **QComboBox** в нашу форму и взглянем на её методы.

Сохраняем файл как **combobox.ui** и конвертируем его в **combobox.py**:

Shell

```
1 pyuic5 combobox.ui -o combobox.py
```

Если вы запустите приложение сейчас, обратите внимание на то, что **QComboBox** — пустой. Чтобы вносить объекты в **QComboBox**, используйте метод **addItem**():

Python

```
1  from PyQt5 import QtWidgets
2  from combobox import Ui_MainWindow
3  import sys
4
5
6  class mywindow(QtWidgets.QMainWindow):
7      def __init__(self):
```

```
        super(mywindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        # Добавляем новые значения
        self.ui.comboBox.addItem("Программист")
        self.ui.comboBox.addItem("Дизайнер")


app = QtWidgets.QApplication([])
application = mywindow()
application.show()

sys.exit(app.exec())
```

Получаем все элементы из QComboBox

Нет прямого способа получить все значения из **QComboBox**, но вы можете применить цикл Python для этой цели. Для этого подойдет функция range.

Python

```
for i in range(self.ui.comboBox.count()):
    print(self.ui.comboBox.itemText(i))
```

Выбор одного элемента из QCombobox

Чтобы выбрать элемент из **QComboBox**, у вас есть два метода:

Python

```
1 # по индексу, который начинается с нуля
2 self.ui.comboBox.setCurrentIndex(1)
3
4 #выбор по тексту
5 self.ui.comboBox.setCurrentText("Second item")
```

Обратите внимание на то, что при **выборе элемента по тексту**, следует убедиться в том, что вы вводите правильный текст. В противном случае, QComboBox останется на первом элементе.

QTableWidget

Если вы хотите просмотреть вашу базу данных в формате таблицы, в PyQt5 предоставляется **QTableWidget** как раз для этой цели.

QTableWidget состоит из клеток, каждая клетка — экземпляр класса **QTableWidgetItem**.

Давайте создадим форму, которая содержит **QTableWidget** и **QPushButton**.

Перетяните QTableWidget и QPushButton из PyQt5 Designer. После этого, сохраните форму как **qtable.ui** и конвертируйте дизайн в **qtable.py**.

Shell

```
1 pyuic5 qtable.ui -o qtable.py
```

Чтобы добавлять ряды в **QTableWidget**, вы можете использовать метод **setRowCount**().

Для внесения столбцов в **QTableWidget**, воспользуйтесь методом **setColumnCount**().

Python

```
 1  from PyQt5 import QtWidgets
 2
 3  # Импортируем нашу форму.
 4  from qtable import Ui_MainWindow
 5  import sys
 6
 7
 8  class mywindow(QtWidgets.QMainWindow):
 9      def __init__(self):
10          super(mywindow, self).__init__()
11          self.ui = Ui_MainWindow()
12          self.ui.setupUi(self)
13
14          self.ui.tableWidget.setColumnCount(2)
15          self.ui.tableWidget.setRowCount(4)
16
17
18 app = QtWidgets.QApplication([])
19 application = mywindow()
20 application.show()
21
22 sys.exit(app.exec())
```

Теперь вы можете писать текст вручную внутри клеток **QTableWidget**.

Очистка содержимого QTableWidget

Чтобы очистить содержимое **QTableWidget**, вы можете использовать метод clear, вот так:

Python

```
from PyQt5 import QtWidgets

# Импортируем нашу форму.
from qtable import Ui_MainWindow
import sys


class mywindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(mywindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.tableWidget.setColumnCount(2)
        self.ui.tableWidget.setRowCount(4)

        # очистка таблицы при клике на кнопку.
        self.ui.pushButton.clicked.connect(self.clear)

    def clear(self):
        self.ui.tableWidget.clear()
```

```
22
23
24 app = QtWidgets.QApplication([])
25 application = mywindow()
26 application.show()
27
28 sys.exit(app.exec())
```

Заполнение QTableWidget из кода

Чтобы заполнить QtableWidget программно, вам нужно использовать метод setItem() для каждого объекта QtableWidgetItem.

Python

```
 1  from PyQt5.QtWidgets import QTableWidgetItem
 2
 3  from qtable import *
 4  import sys
 5
 6  data = []
 7  data.append(('Заполнить', 'QTableWidget'))
 8  data.append(('с данными', 'в Python'))
 9  data.append(('очень', 'просто'))
10
11
12 class mywindow(QtWidgets.QMainWindow):
13
14     def __init__(self):
15         super().__init__()
16         self.ui = Ui_MainWindow()
17
18         self.ui.setupUi(self)
```

```
19        self.ui.tableWidget.setRowCount(3)
20        self.ui.tableWidget.setColumnCount(2)
21
22        # очистка таблицы при клике на кнопку.
23        self.ui.pushButton.clicked.connect(self.clear)
24
25        row = 0
26        for tup in data:
27            col = 0
28
29            for item in tup:
30                cellinfo = QTableWidgetItem(item)
31                self.ui.tableWidget.setItem(row, col, cellinfo)
32                col += 1
33
34            row += 1
35
36    def clear(self):
37        self.ui.tableWidget.clear()
38
39
40 app = QtWidgets.QApplication([])
41 win = mywindow()
42 win.show()
43
44 sys.exit(app.exec())
```

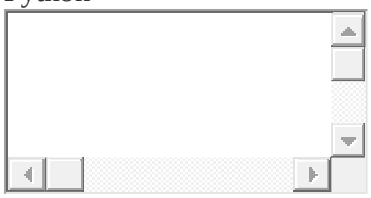
- Сначала мы создали список трех кортежей Python;
- Внутри конструктора главного окна, мы установили количество столбцов и рядов;
- Далее мы перебираем список и получаем каждый кортеж в списке, для заполнения клеток таблицы, при помощи метода **setItem**()
- Наконец, мы показываем главное окно.

Делаем QTableWidget нередактируемым (только для чтения)

Вам может не понравится то, что клетки в вашей таблице могут быть отредактированы пользователем в том или ином случае. Например, при отображении нередактируемых данных. В этом случае возможность редактирования не имеет никакого смысла.

Чтобы сделать **QTableWidget** нередактируемым, вы можете использовать метод **setFlags**(), чтобы сделать каждый **QTableWidgetItem** доступным только для чтения.

Python



```
1 # Только для чтения
2 cellinfo.setFlags(
3     QtCore.Qt.ItemIsSelectable | QtCore.Qt.ItemIsEnabled
4 )
```

Вам нужно установить флажки, перед тем как настраивать содержимое вашей клетки.

Таким образом, ваш код будет выглядеть вот так:

Python

```
 1  from PyQt5.QtWidgets import QTableWidgetItem
 2
 3  from qtable import *
 4  import sys
 5
 6  data = []
 7  data.append(('Заполнить', 'QTableWidget'))
 8  data.append(('с данными', 'в Python'))
 9  data.append(('очень', 'просто'))
10
11
12 class mywindow(QtWidgets.QMainWindow):
13
14     def __init__(self):
15         super().__init__()
16         self.ui = Ui_MainWindow()
17
18         self.ui.setupUi(self)
19         self.ui.tableWidget.setRowCount(3)
20         self.ui.tableWidget.setColumnCount(2)
21
22         # очистка таблицы при клике на кнопку.
23         self.ui.pushButton.clicked.connect(self.clear)
24
25         row = 0
26         for tup in data:
27             col = 0
28
```

```
29            for item in tup:
30                cellinfo = QTableWidgetItem(item)
31
32                # Только для чтения
33                cellinfo.setFlags(
34                    QtCore.Qt.ItemIsSelectable | QtCore.Qt.ItemIsEnabled
35                )
36
37                self.ui.tableWidget.setItem(row, col, cellinfo)
38                col += 1
39
40            row += 1
41
42    def clear(self):
43        self.ui.tableWidget.clear()
44
45
46 app = QtWidgets.QApplication([])
47 win = mywindow()
48 win.show()
49
50 sys.exit(app.exec())
```

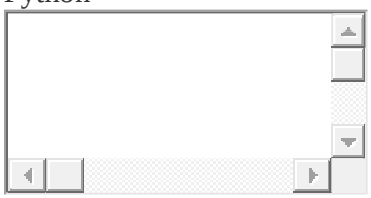
Теперь, если вы попробуете отредактировать какую-либо клетку — у вас не выйдет, так как QtableWidgetItem теперь нельзя редактировать

Заголовок для столбцов в QTableWidget

До этого момента, названия столбцов **QTableWidget** были числами. Как на счет того, чтобы поменять названия столбцов на что-нибудь другое?

Чтобы задать текст заголовкам **QTableWidget**, вы можете использовать метод **setHorizontalHeaderLabels**(), вот так:

Python

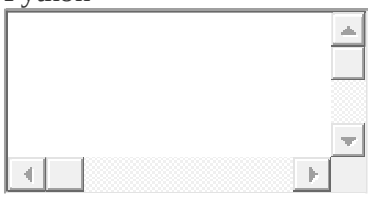


```
1  from PyQt5.QtWidgets import QTableWidgetItem
2  from qtable import *
3  import sys
4
5  data = []
6  data.append(('BMW', '1991'))
7  data.append(('Audi', '2003'))
8  data.append(('Volvo', '2010'))
9
10
11 class mywindow(QtWidgets.QMainWindow):
12
13     def __init__(self):
14         super().__init__()
15         self.ui = Ui_MainWindow()
16
17         self.ui.setupUi(self)
18         self.ui.tableWidget.setRowCount(3)
19         self.ui.tableWidget.setColumnCount(2)
20
21         # очистка таблицы при клике на кнопку.
22         self.ui.pushButton.clicked.connect(self.clear)
23
24         # заголовки для столбцов.
```

```
25        self.ui.tableWidget.setHorizontalHeaderLabels(
26            ('Марка', 'Год выпуска')
27        )
28
29        row = 0
30        for tup in data:
31            col = 0
32
33            for item in tup:
34                cellinfo = QTableWidgetItem(item)
35                self.ui.tableWidget.setItem(row, col, cellinfo)
36                col += 1
37
38            row += 1
39
40    def clear(self):
41        self.ui.tableWidget.clear()
42
43
44 app = QtWidgets.QApplication([])
45 win = mywindow()
46 win.show()
47
48 sys.exit(app.exec())
```

Таким же образом, вы можете **менять заголовок ряда**, при помощи метода **setVerticalHeaderLabels**():

Python



```
from PyQt5.QtWidgets import QTableWidgetItem
from qtable import *
import sys

data = []
data.append(('BMW', '1991'))
data.append(('Audi', '2003'))
data.append(('Volvo', '2010'))


class mywindow(QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()

        self.ui.setupUi(self)
        self.ui.tableWidget.setRowCount(3)
        self.ui.tableWidget.setColumnCount(2)

        # очистка таблицы при клике на кнопку.
        self.ui.pushButton.clicked.connect(self.clear)

        # заголовки для столбцов.
        self.ui.tableWidget.setHorizontalHeaderLabels(
            ('Марка', 'Год выпуска')
        )

        # названия рядов.
        self.ui.tableWidget.setVerticalHeaderLabels(
            ('900$', '5000$', '13000$')
        )

        row = 0
        for tup in data:
            col = 0

            for item in tup:
                cellinfo = QTableWidgetItem(item)
                self.ui.tableWidget.setItem(row, col, cellinfo)
                col += 1

            row += 1

    def clear(self):
        self.ui.tableWidget.clear()


app = QtWidgets.QApplication([])
win = mywindow()
win.show()

sys.exit(app.exec())
```

Как сортировать QTableWidget

Вы можете сделать ваш QTableWidget сортируемым, при помощи метода **setSortingEnabled**().

Python

```
1 self.ui.tableWidget.setSortingEnabled(True)
```

Теперь, если пользователь нажмет на любой заголовок столбца, он может сортировать данные в порядке убывания, или возрастания.

Вы можете использовать этот метод перед, или до наполнения **QTableWidget** данными.

Что на счет сортировки в QTableWidget, но только для определенного столбца?

Вы можете использовать метод **sortByColumn**(), и указать индекс столбца и порядок сортировки, вот так:

Python

```
from PyQt5.QtWidgets import QTableWidgetItem
from qtable import *
import sys

data = []
data.append(('BMW', '2005'))
data.append(('Audi', '2003'))
data.append(('Volvo', '1990'))
data.append(('Toyota', '2018'))


class mywindow(QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()

        self.ui.setupUi(self)

        # Кол-во рядов меняется в зависимости от значений в data.
        self.ui.tableWidget.setRowCount(
            len(data)
        )

        # Кол-во столбцов меняется в зависимости от data.
        self.ui.tableWidget.setColumnCount(
            len(data[0])
        )

        # очистка таблицы при клике на кнопку.
        self.ui.pushButton.clicked.connect(self.clear)

        # заголовки для столбцов.
        self.ui.tableWidget.setHorizontalHeaderLabels(
            ('Марка', 'Год выпуска')
        )

        row = 0
        for tup in data:
            col = 0

            for item in tup:
                cellinfo = QTableWidgetItem(item)
                self.ui.tableWidget.setItem(row, col, cellinfo)
                col += 1

            row += 1

        # Сортировка по году выпуска.
        # 0 - Марка
        # 1 - Год выпуска
        self.ui.tableWidget.sortByColumn(
            1, QtCore.Qt.AscendingOrder
        )

    def clear(self):
        self.ui.tableWidget.clear()


app = QtWidgets.QApplication([])
win = mywindow()
win.show()

```

```
64 sys.exit(app.exec())
```

Кстати, вы можете использовать метод **sortItems**() для сортировки **QTableWidget** в возрастающем порядке по умолчанию.

Python

```
1 self.ui.tableWidget.sortItems(0)
```

Или вы можете определить свой порядок сортировки:

Python

```
1 self.ui.tableWidget.sortItems(1, QtCore.Qt.DescendingOrder)
```

Помните, что если вы хотите сортировать ваши столбцы программно, вам нужно использовать методы сортировки **после заполнения QTableWidget** данными, иначе они не будут в нужном вам порядке.

Добавляем QComboBox в QTableWidget

У вас может появится задача, чтобы пользователь выбирал значение внутри **QTableWidget**, вместо ввода текста.
Как на счет того, чтобы добавить **QComboBox** в **QTableWidgetItem**?

Чтобы добавить **QComboBox** внутрь **QTableWidgetItem**, вы можете использовать метод **setCellWidget**():

Python

```
1  from PyQt5.QtWidgets import QTableWidgetItem
2  from qtable import *
3  import sys
4
5  data = ['Python', 'PHP', 'Java']
6
7
8  class mywindow(QtWidgets.QMainWindow):
9      def __init__(self):
10         super().__init__()
11         self.ui = Ui_MainWindow()
12         self.ui.setupUi(self)
13
14         self.ui.tableWidget.setRowCount(3)
15         self.ui.tableWidget.setColumnCount(2)
16
```

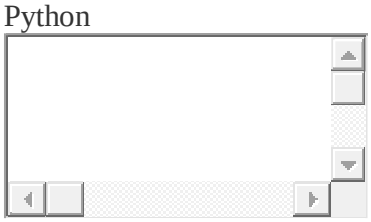
```
17        row = 0
18        for item in data:
19            cellinfo = QTableWidgetItem(item)
20
21            combo = QtWidgets.QComboBox()
22            combo.addItem("Изучить")
23            combo.addItem("Забыть")
24            combo.addItem("Удалить")
25
26            self.ui.tableWidget.setItem(row, 0, cellinfo)
27            self.ui.tableWidget.setCellWidget(row, 1, combo)
28            row += 1
29
30
31 app = QtWidgets.QApplication([])
32 win = mywindow()
33 win.show()
34
35 sys.exit(app.exec())
```

Отлично!

Не ограничивайте себя в воображении и попробуйте вставлять различные виджеты, такие как QCheckbox, или даже QProgressBar.

QProgressBar в QTableWidget

Python



```
1  from PyQt5.QtWidgets import QTableWidgetItem
2  from qtable import *
```

```
import sys

data = (
    ('Python', 95.5),
    ('PHP', 55.1),
    ('Java', 0.29)
)


class mywindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.tableWidget.setRowCount(3)
        self.ui.tableWidget.setColumnCount(2)

        self.ui.tableWidget.setHorizontalHeaderLabels(
            ('Язык', 'Знания')
        )

        line = 0
        for item in data:
            cellinfo = QTableWidgetItem(item[0])
            self.ui.tableWidget.setItem(line, 0, cellinfo)

            # Создаем QProgressBar
            progress = QtWidgets.QProgressBar()
            progress.setMinimum(0)
            progress.setMaximum(100)

            # Формат вывода: 10.50%
            progress.setValue(item[1])
            progress.setFormat('{0:.2f}%'.format(item[1]))

            # Добавляем виджет в ячейку.
            self.ui.tableWidget.setCellWidget(line, 1, progress)

            line += 1


app = QtWidgets.QApplication([])
win = mywindow()
win.show()

sys.exit(app.exec())
```

Указанный выше код будет таким же, за исключением строки, где вы создаете **QComboBox**, здесь вы внесете тот виджет, который вам нужен.

Единственное ограничение — это ваше собственное воображение!

Компиляция Python приложения

Вы можете конвертировать ваши программы Python в бинарные исполняемые, и для этого имеется множество инструментов.

Лично я предпочитаю **pyinstaller**, который подходит для упаковки кода Python в исполняемый файл под Windows, Mac OS X, Solaris, Linux и FreeBSD. Все это будет поддерживаться 32 и 64-битной архитектурой.

Лучшая в **pyinstaller** для нас — это наличие полной поддержки для PyQt5.

Отлично! Для начала, установим pyinstaller:

Shell

```
1 $ pip3 install pyinstaller
```

После проведения установки, вы можете конвертировать программы Python следующим образом:

Python

```
1 $ pyinstaller test.py
```

Ваш исполняемый файл будет создан в папке под названием **dist** в директории вашей программы Python.

Как вы могли догадаться, вместе с исполняемым файлом будет генерироваться множество зависимостей. Как сделать из этого один файл?

Вы можете создать один исполняемый файл. Вот так:

Shell

```
1 $ pyinstaller --onefile test.py
```

Каждый раз, когда вы запускаете ваш исполняемый файл, будет возникать окно, как его спрятать?

Вы можете использовать флажки **-w** или **–noconsole**, чтобы спрятать окно консоли:

Shell

```
1 $ pyinstaller -w test.py
```

Эта опция доступна только для **Windows** и **Mac OS X**.

Pyinstaller предоставляет множество вариантов для упаковки вашего приложения, чтобы увидеть полный список, используйте –help:

Shell

```
1 $ pyinstaller --help
```

Я старался сделать все на столько простым, на сколько это возможно. Надеюсь, это руководство оказалось для вас полезным.

Спасибо.