* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/165/groups/1257)

[Урок QT Знакомство](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225)

**Что такое QT и PyQT. Знакомство**

1. [Графический интерфейс](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225/materials/2195#1)
2. [Установка и настройка](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225/materials/2195#2)
3. [Основные виджеты](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225/materials/2195#3)
4. [Кто отправил сигнал](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225/materials/2195#4)
5. [Открытие других форм](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225/materials/2195#5)
6. [Итоги](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1225/materials/2195#6)

**Аннотация**

*В уроке рассказывается о графическом интерфейсе и разных способах его реализации. Разбирается настройка окружения и примеры работы с основными элементами интерфейса.*

**Графический интерфейс**

Раньше большинство ваших программ запускались и выполнялись из консоли, то есть и ввод, и вывод осуществлялся с использованием интерфейса командной строки, без создания привычного пользовательского интерфейса (вы немного могли поиграть с графическим интерфейсом на дополнительном уроке по tkinter). Другими словами, до этого мы общались с пользователем, вводили и выводили данные с использованием интерфейса командной строки (**CLI** — Command Line Interface).

Сначала текстового интерфейса хватает, но потом наступает момент, когда программу необходимо передать незнакомому с консолью пользователю. Графический интерфейс (GUI — Graphical User Interface) более «дружелюбный» к пользователям, а если в программе необходимо отображать не только текст, но и графическую информацию, его использование становится необходимостью.

Рассмотрим основные понятия концепции GUI. Допустим, есть знакомый нам текстовый редактор. Когда мы нажимаем клавишу, возникает событие «Клавиша такая-то нажата». В то же время внутри программы запущен обработчик, который проверяет, какая клавиша нажата, какая раскладка выбрана и так далее. Затем он выполняет действие: выводит нужный символ на экран.

Для языка программирования Python есть много способов создания приложений с графическим интерфейсом, в частности, встроенная библиотека tkinter. Но возможности пакета PyQT гораздо шире.

Что же такое PyQT? Для начала разберемся, что такое QT. Это написанная на C++ библиотека с классами для создания графического интерфейса. Библиотека получилась настолько удачной и популярной, что в других языках стали появляться свои библиотеки — «прослойки». Для Python это **PyQT**.

**Установка и настройка**

PyQt устанавливается так же, как и любая другая библиотека в Python:

pip install PyQT5

**Основные виджеты**

Все графические приложения состоят из виджетов.

**Виджет**

Виджет — минимальный блок графического интерфейса пользователя.

В библиотеке PyQT5 существует множество модулей, но чаще других используется **QtWidgets**. Именно в нем находятся классы, соответствующие различным элементам интерфейса.

Напишем самую первую программу с графическим интерфейсом:

import sys

from PyQt5.QtWidgets import QApplication, QWidget

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Первая программа')

if \_\_name\_\_ == '\_\_main\_\_':

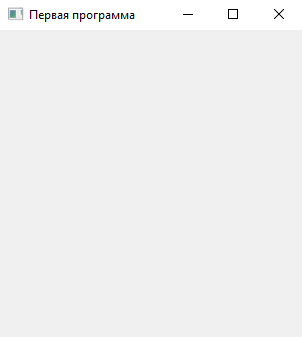
app = QApplication(sys.argv)

ex = Example()

ex.show()

sys.exit(app.exec())

Если мы запустим эту программу, увидим такое окно:



Давайте разберемся, что происходит в этой программе. Обратите внимание на наш класс — Example. Он наследуется от базового класса QWidget, который определяет простейшее окно. От него наследуется много встроенных виджетов.

Первое, что можно увидеть в классе, — перегруженный конструктор.

super().\_\_init\_\_() — эта строка вызывает конструктор родительского класса. Потом вызывается метод класса с названием initUI.

Разумеется, инициализацию интерфейса можно реализовать и в инициализаторе класса, но хорошей практикой считается вынос этой функциональности в отдельный метод класса с названием initUI.

**PEP 8**

Обратите внимание: метод initUI называется с нарушением правил PEP 8. Как мы говорили ранее, такое отступление возможно для сохранения совместимости с используемой библиотекой, а так как библиотека QT изначально написана на C++ с именованием методов в стиле camelCase, то initUI — допустимое название метода в классе, который наследован от QWidget.

В нем мы определяем положение и размеры нашего окна методом setGeometry. Первые два параметра — X и У — координаты левого верхнего угла формы относительно левого верхнего угла нашего монитора. Оставшиеся — ширина и высота виджета. Методом setWindowTitle задаем заголовок нашего окна.

Мы создали класс, но пока его не используем. Чтобы начать с ним работать, необходимо куда-то разместить наш виджет.

Для этого нужно создать приложение, app = QApplication(sys.argv). Несмотря на то, что после инициализации переменная app используется только один раз, оно необходимо. Все скрытое от пользователя и разработчика взаимодействие программы с ОС возможно только благодаря этому классу. А в случае запуска программы из командной строки, мы сможем передать ей какие-либо аргументы.

Затем мы создаем экземпляр нашего класса: ex = Example(). Все готово, можно запускать. Метод show() отображает наш виджет в приложение. Но отобразить недостаточно, надо запустить. Для этого вызываем метод app.exec(). Он запускает основной цикл нашей программы, начинается обработка событий.

В последней строке программы этот вызов «обернут» в sys.exit. Это сделано для корректного завершения программы.

Но просто пустое окно — это скучно, начнем добавлять туда виджеты. Первый на очереди — знакомая нам кнопка. Класс, который необходим для работы с ней, называется QPushButton.

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Вторая программа')

btn = QPushButton('Кнопка', self)

btn.resize(100, 100)

btn.move(100, 100)

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

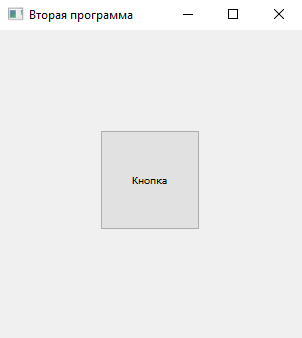
ex = Example()

ex.show()

sys.exit(app.exec())

Вспомним картинку выше: у любого виджета, кроме базового, должен быть «родитель». Когда мы добавляем кнопку, «родителем» выступает наш виджет окна. Поэтому при объявлении кнопки мы указываем не только текст, но и экземпляр класса QWidget (или, как в нашем случае, его наследника).

Метод resize позволяет изменить размеры кнопки. А с помощью метода move мы указываем расположение нашей кнопки в виджете-«родителе». Запустим программу и убедимся, что кнопка появилась.



Но пока она бесполезная. Сделаем ее полезной — добавим функциональность.

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Третья программа')

self.btn = QPushButton('Кнопка', self)

self.btn.resize(100, 100)

self.btn.move(100, 100)

self.btn.clicked.connect(self.hello)

def hello(self):

self.btn.setText('Привет')

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

ex = Example()

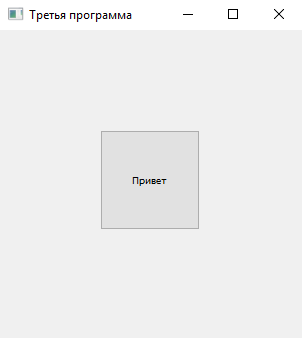
ex.show()

sys.exit(app.exec())

Что изменилось по сравнению с предыдущей программой? Во-первых, теперь btn — это поле класса, а не просто локальная переменная метода, как было в прошлый раз. Поэтому добраться до кнопки мы теперь можем не только из метода initUI, а из любого места внутри класса нашей формы (и даже из любого кода, который имеет доступ к объекту нашей формы). А во-вторых, добавилась функциональность.

self.btn.clicked.connect(self.hello) — что значит эта фраза в переводе на человеческий язык? «Если получишь **событие clicked** от **объекта self.btn**, вызови **обработчик self.hello()**». В нем с помощью метода setText мы меняем текст на кнопке.

Но эта кнопка фактически одноразовая. Каждый раз при ее нажатии произойдет вызов функции изменения надписи, но мы, как пользователи, этого не увидим.



Давайте выводить на кнопке количество нажатий на нее. Никаких дополнительных полей нам не понадобится — хранить информацию будем прямо в «кнопке».

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Четвёртая программа')

self.btn = QPushButton('0', self)

self.btn.resize(100, 100)

self.btn.move(100, 100)

self.btn.clicked.connect(self.count)

def count(self):

self.btn.setText("{}".format(int(self.btn.text()) + 1))

if \_\_name\_\_ == '\_\_main\_\_':

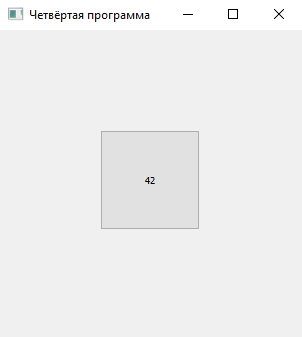
app = QApplication(sys.argv)

ex = Example()

ex.show()

sys.exit(app.exec())

Метод text() возвращает строку — текущую надпись на кнопке.



Наверняка нам понадобится отображать данные где-то еще, не только на кнопке. Для текстовых данных лучше использовать QLabel, а для цифр есть красивый виджет — QLCDNumber.

import sys

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

from PyQt5.QtWidgets import QLCDNumber

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Пятая программа')

self.btn = QPushButton('Кнопка', self)

self.btn.resize(self.btn.sizeHint())

self.btn.move(100, 150)

self.btn.clicked.connect(self.hello)

self.label = QLabel(self)

self.label.setText("Количество нажатий на кнопку")

self.label.move(80, 30)

self.LCD\_count = QLCDNumber(self)

self.LCD\_count.move(110, 80)

self.count = 0

def hello(self):

self.count += 1

self.LCD\_count.display(self.count)

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

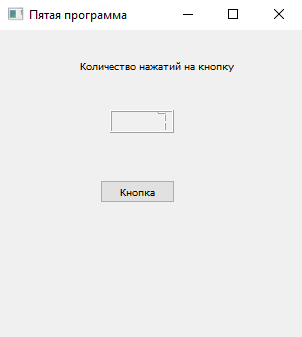
ex = Example()

ex.show()

sys.exit(app.exec())

Теперь для хранения будем использовать отдельное поле — count. Для задания значения у QLCDNumber используется метод display, а для QLabel — setText.

Код self.btn.resize(self.btn.sizeHint()) подстраивает размеры кнопки под размер надписи на ней, чтобы не возникла ситуация, что часть надписи заходила за границу этого виджета.



QLCDNumber, кроме чисел, умеет показывать следующие символы: O (нулем), S (с помощью 5), g (с помощью 9), минус, точку, A, B, C, D, E, F, h, H, L, o, P, r, u, U, Y, кавычку, пробел. То есть вполне возможно написать на QLCDNumber какое-либо сообщение, например Error. Те символы, которые QLCDNumber отобразить не может, он просто заменяет на пробелы.

Однако отображать данные часто недостаточно, не менее важно получать информацию от пользователя. Для этого существует виджет QLineEdit. Он позволяет вводить одну строку пользовательских данных.

import sys

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

from PyQt5.QtWidgets import QLCDNumber, QLineEdit

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Шестая программа')

self.btn = QPushButton('Кнопка', self)

self.btn.resize(self.btn.sizeHint())

self.btn.move(100, 150)

self.btn.clicked.connect(self.hello)

self.label = QLabel(self)

self.label.setText("Привет, неопознанный лев")

self.label.move(40, 30)

self.name\_label = QLabel(self)

self.name\_label.setText("Введите имя: ")

self.name\_label.move(40, 90)

self.name\_input = QLineEdit(self)

self.name\_input.move(130, 90)

def hello(self):

name = self.name\_input.text()

self.label.setText("Привет, {}".format(name))

if \_\_name\_\_ == '\_\_main\_\_':

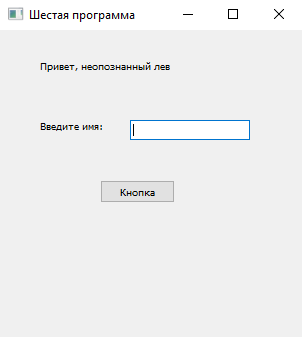
app = QApplication(sys.argv)

ex = Example()

ex.show()

sys.exit(app.exec())

Метод text() позволяет получить введенную пользователем строчку.



**Кто отправил сигнал**

Вспомним немного теории, рассмотренной в самом начале урока. Когда нажимают кнопку, возникает сигнал, который обрабатывается функцией. Но если у нас одна функция выполняется в нескольких ситуациях при нажатиях разных кнопок?

Чтобы определить, кто является источником сигнала, в PyQT есть встроенный метод .sender().

Рассмотрим пример такой программы.

import sys

from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import QWidget, QApplication, QPushButton

import PyQt5.QtGui as QtGui

class Example(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Кто отправил сигнал')

self.button\_1 = QPushButton(self)

self.button\_1.resize(100, 50)

self.button\_1.move(100, 80)

self.button\_1.setText("Кнопка 1")

self.button\_1.clicked.connect(self.run)

self.button\_2 = QPushButton(self)

self.button\_2.resize(100, 50)

self.button\_2.move(100, 150)

self.button\_2.setText("Кнопка 2")

self.button\_2.clicked.connect(self.run)

self.show()

def run(self):

print(self.sender().text())

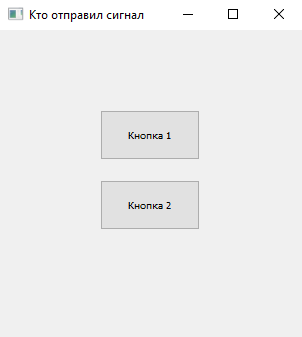
if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

ex = Example()

ex.show()

sys.exit(app.exec())



**Открытие других форм**

Разумеется, далеко не всегда приложение ограничивается одной формой. Обычно, как минимум, есть еще одна — «О программе», а также формы с настройками, более детальным просмотром информации и т. д. Конечно, PyQT дает возможность в нашем приложении создавать формы из других форм. Обычно (но далеко не всегда) для главной формы приложения выбирают класс QMainWindow, а для дочерних форм — класс QWidget. Чтобы создать форму из другой формы, достаточно сделать объект нужного нам класса и вызвать у него метод show.

import sys

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

from PyQt5.QtWidgets import QMainWindow, QLabel

class FirstForm(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initUI()

def initUI(self):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Главная форма')

self.btn = QPushButton('Другая форма', self)

self.btn.resize(100, 100)

self.btn.move(100, 100)

self.btn.clicked.connect(self.open\_second\_form)

def open\_second\_form(self):

self.second\_form = SecondForm(self, "Данные для второй формы")

self.second\_form.show()

class SecondForm(QWidget):

def \_\_init\_\_(self, \*args):

super().\_\_init\_\_()

self.initUI(args)

def initUI(self, args):

self.setGeometry(300, 300, 300, 300)

self.setWindowTitle('Вторая форма')

self.lbl = QLabel(args[-1], self)

self.lbl.adjustSize()

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

ex = FirstForm()

ex.show()

sys.exit(app.exec())

Обратите внимание на два момента:

1. Мы сохраняем созданную форму в атрибут нашей родительской формы для того, чтобы иметь возможность управлять ею из других методов. И для того, чтобы сборщик мусора Python не удалил ее случайно как объект, на который нет ссылок.
2. В инициализаторе дочерней формы, помимо self, есть еще \*args, куда мы можем помещать информацию, которую хотим передать из родительской формы в дочернюю. Тут мы передаем ссылку на объект-родитель (его обычно передают первым) и сообщение, которое мы будет отображать на второй форме в QLabel.

**Итоги**

На этом уроке мы изучили работу с основными виджетами:

* QWidget
* QPushButton
* QLCDNumber
* QLabel
* QLineEdit

Но мы рассмотрели далеко не все доступные методы. Информацию о других методах можно посмотреть на [официальном сайте QT](https://doc.qt.io/qt-5/qtwidgets-module.html). Обратите внимание: документация написана для языка C++, так что нужно обращать внимание лишь на названия методов и параметры, а не на форму.

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты