Проект QT. Что такое QT и PyQT. Знакомство

План урока

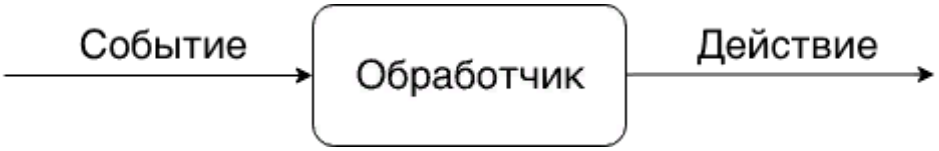
Аннотация

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

1. Графический интерфейс

Раньше большинство ваших программ запускались и выполнялись из консоли. Сначала текстового интерфейса хватает, но потом наступает момент, когда программу необходимо передать незнакомому с консолью пользователю. Графический интерфейс (**GUI** — Graphical User Interface) более *дружелюбный*, а если в программе необходимо отображать не только текст, но и графическую информацию — то консоли уже явно не хватит.

Рассмотрим основные понятия концепции GUI. Допустим, у нас 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 разберёмся, что такое QT. QT — 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.

2. Установка и настройка

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

```
pip install PyQT5
```

3. Основные виджеты

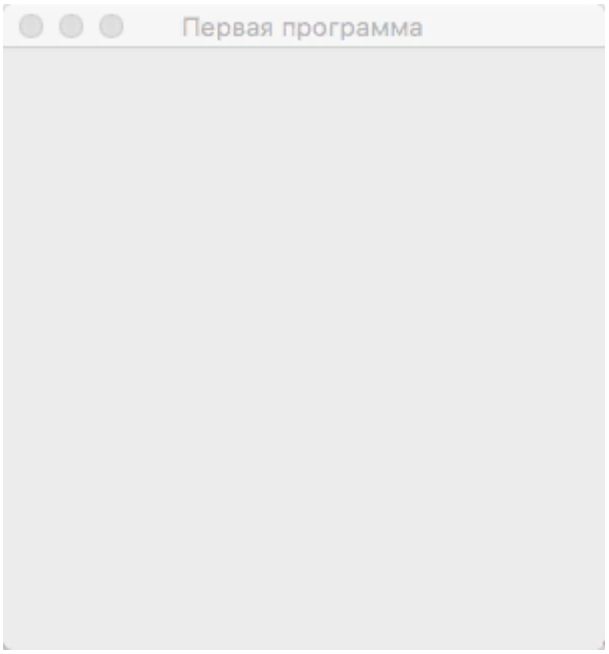
Все графические приложения состоят из **виджетов**. **Виджет** — это минимальный блок графического интерфейса (UI).

В библиотеке PyQT5 множество модулей, но чаще других используется **QtWidgets**. Именно в нём 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, то увидим такое окно:



Давайте разберёмся, 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**.

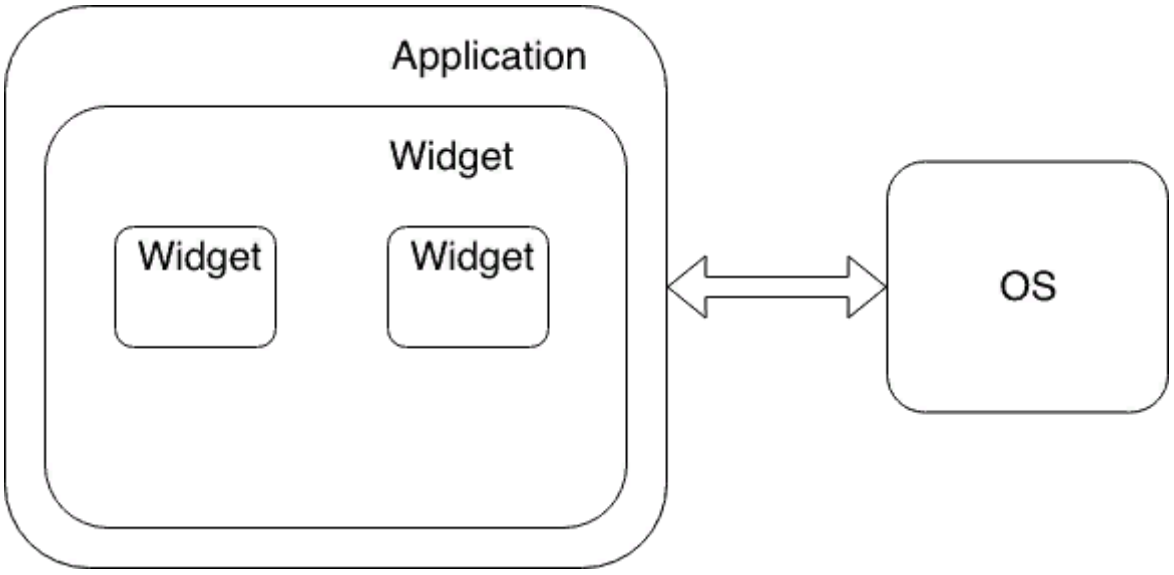
В нём мы определяем положение и размеры нашего окна методом **setGeometry**. Первые два параметра — X и У координаты левого верхнего угла. Оставшиеся — ширина и высота виджета. Методом **setWindowTitle** задаём заголовок нашего окна.

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. Всё 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 создаём экземпляр нашего класса: **ex = Example()**. Всё 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 «обёрнут» в **sys.exit**. Это сделано для корректного завершения программы.



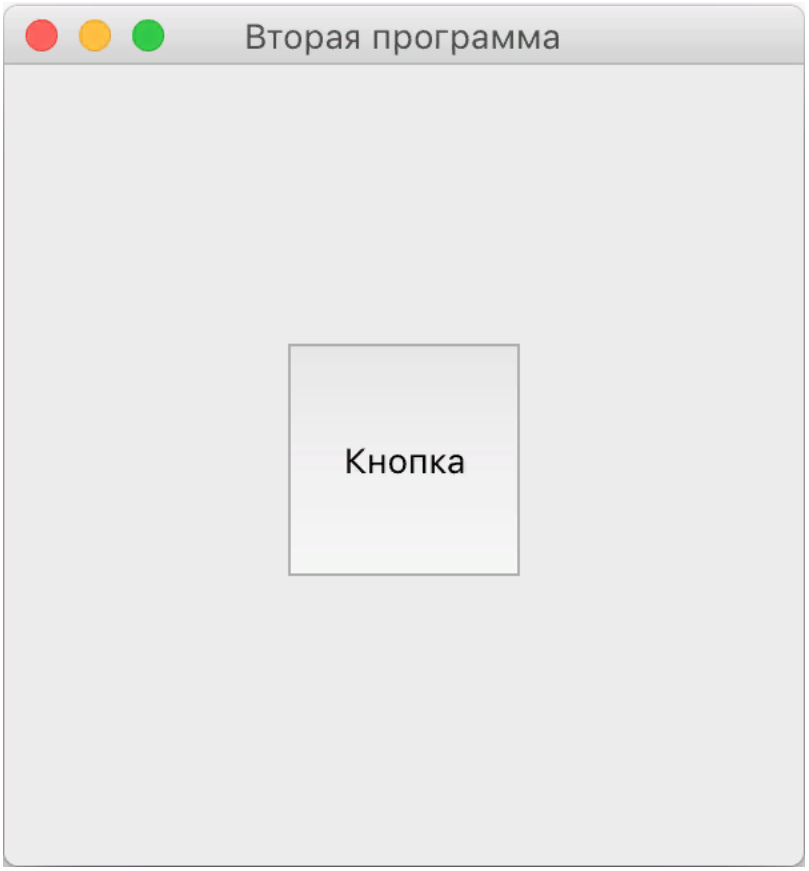
Но просто пустое окно — это скучно, начнём добавлять туда виджеты. Первый на очереди — знакомая нам кнопка. Класс, который необходим для работы с ней, называется **QPushButton**.

```
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 она бесполезная. Сделаем её полезной — добавим функциональность.

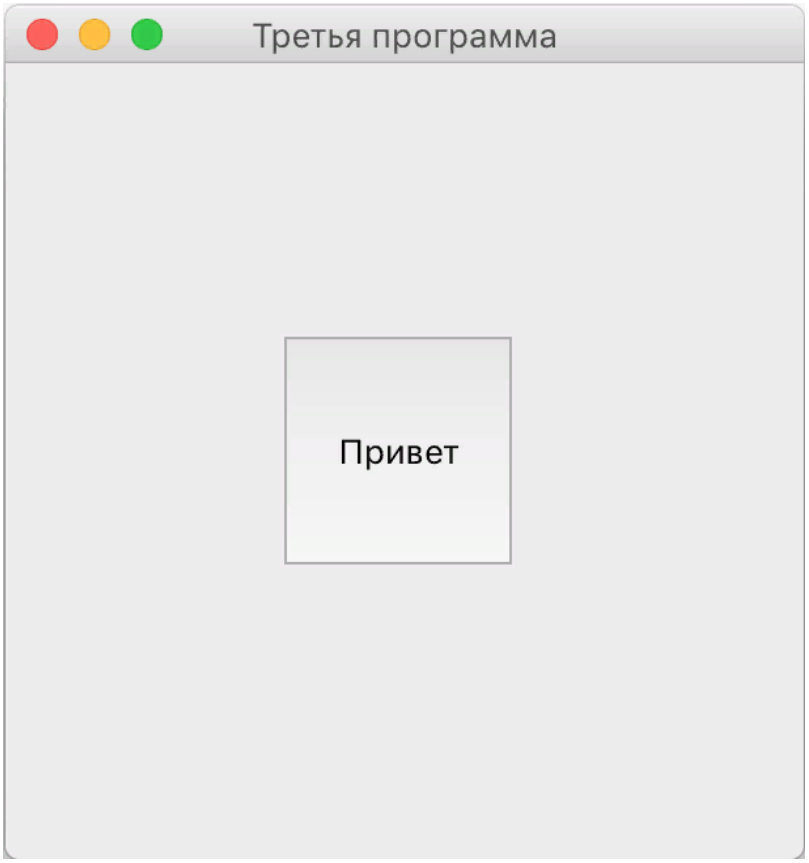


```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton


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. Ведь если бы мы во второй программе попросили отобразить окно заново, кнопки бы на нём уже не было. 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 нём с помощью метода **setText** мы меняем текст на кнопке.

Но эта кнопка фактически «одноразовая». Каждый раз при её нажатии произойдёт вызов функции изменения надписи, но мы, как пользователи, этого не увидим.

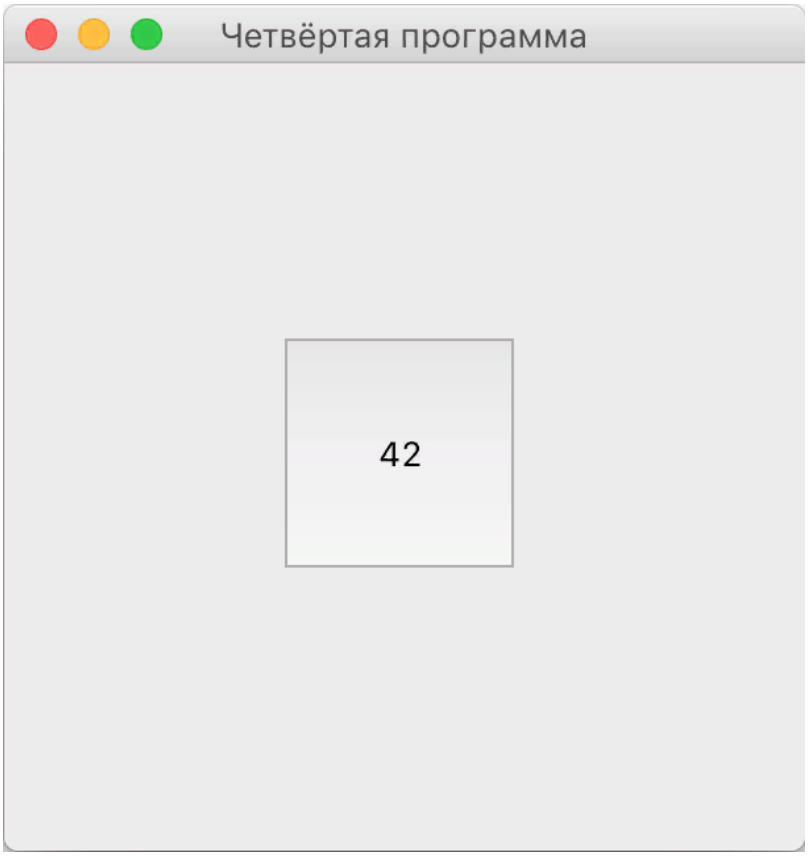


Давайте выводить на кнопке количество нажатий на неё. Никаких дополнительных полей нам не понадобится — хранить информацию будем прямо в «кнопке».

```
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.

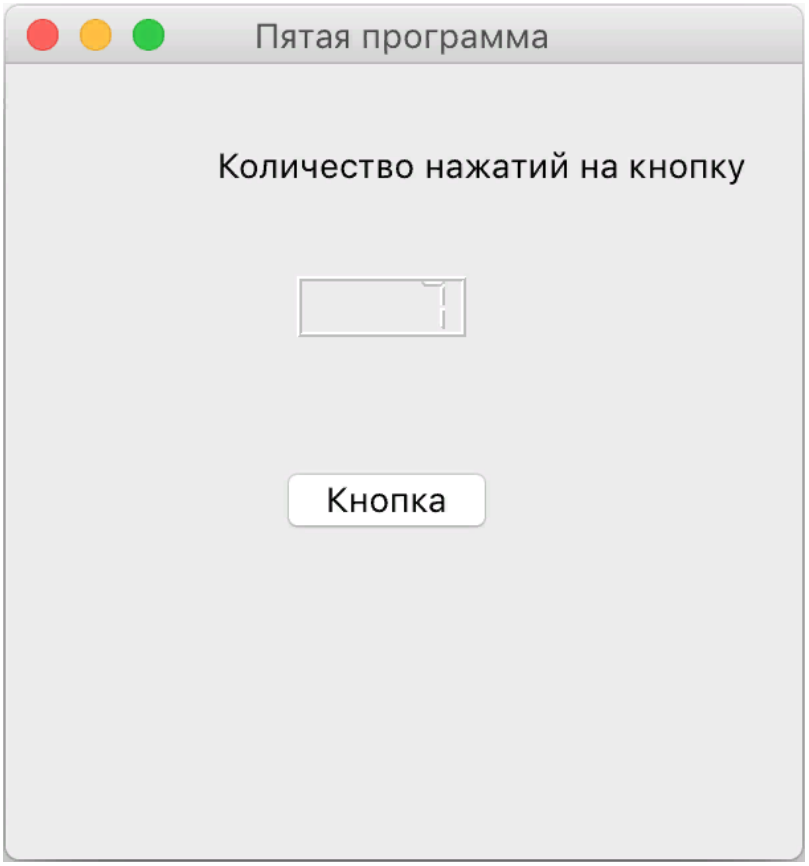


Но ведь надо и отображать данные. Для текстовых данных лучше использовать **QLabel**, а для цифр есть красивый виджет **QLCDNumber**.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
from PyQt5.QtWidgets import QLabel, 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())
```

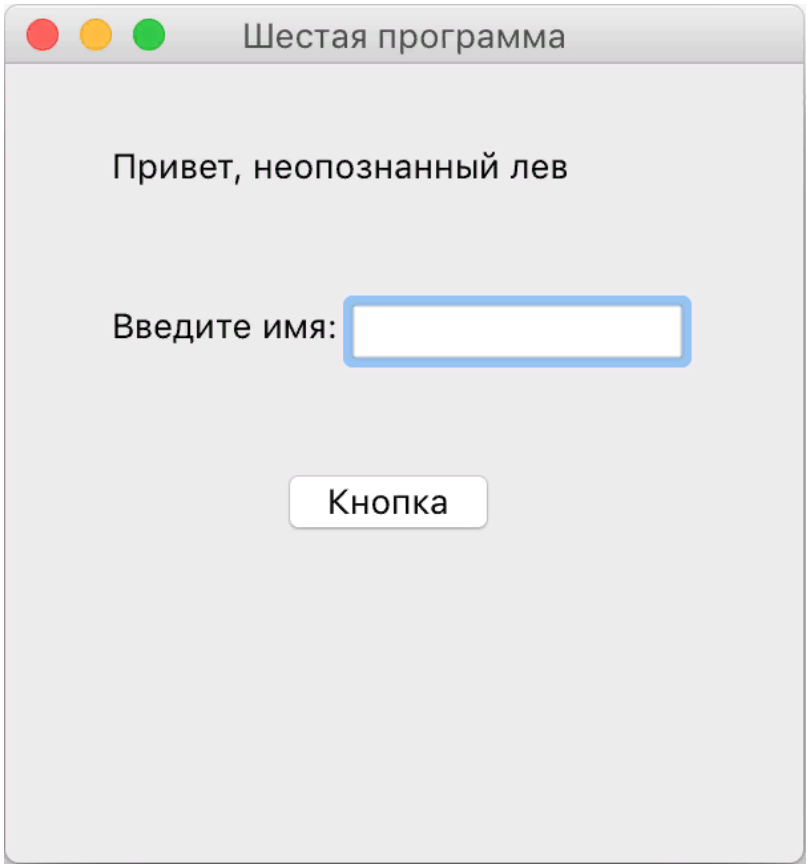
Теперь для хранения будем использовать отдельное поле — **"count"**. Для задания значения у **QLCDNumber** используется метод **display**, а для **QLabel** — **setText**.



Но ведь часто нужно вводить пользовательские данные. Для этого существует виджет **QLineEdit**. Он позволяет вводить одну строку пользовательских данных.

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
from PyQt5.QtWidgets import QLCDNumber, QLabel, 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 введённую пользователем строчку.



4. Кто отправил сигнал

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

Для того, чтобы определить, кто источник 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, 150, 150)
        self.setWindowTitle('Кто отправил сигнал')

        self.button_1 = QPushButton(self)
        self.button_1.move(20, 40)
        self.button_1.setText("Кнопка 1")
        self.button_1.clicked.connect(self.run)

        self.button_2 = QPushButton(self)
        self.button_2.move(20, 80)
        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())
```

5. 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. Обратите внимание, что документация написана с учётом C++, так что нужно обращать внимание лишь на название методов и параметры, но не на форму.