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

[Урок QT standalone](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1232)

**Обработка событий. Сборка независимого приложения**

1. [Обработка нажатий клавиатуры](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1232/materials/2579#1)
2. [Работа с мышью](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1232/materials/2579#2)
3. [Создание standalone-приложений](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1232/materials/2579#3)
4. [Создание БД](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1232/materials/2579#4)
5. [requirements.txt](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1232/materials/2579#5)

**Аннотация**

*В этом уроке мы научимся взаимодействовать с пользователем на новом уровне: добавим в программу работу с клавиатурой и мышкой. И рассмотрим, как построить собрать нашу программу в exe-файл.*

**Обработка нажатий клавиатуры**

У каждой клавиши на клавиатуре есть свой уникальный код. Однако работать с числами не всегда удобно, поэтому в PyQt в модуле Qt с кодами сопоставлены и понятные имена. Согласитесь, что гораздо легче понимать смысл вот такой строки: print(Qt.Key\_F), нежели такой: print(70).

Подключить модуль Qt можно, например, таким способом:

from PyQt5.QtCore import Qt

Все имена кнопок и их коды можно найти [здесь](https://doc.qt.io/qt-5/qt.html).

Для обработки нажатий на клавиши в вашем классе необходимо переопределить (создать) метод keyPressEvent(self, event):

def keyPressEvent(self, event):

if event.key() == Qt.Key\_F:

# code

Если во время работы приложения будет нажата клавиша **F**, условие выполнится.

Чтобы обработать комбинацию клавиш (Hotkey), код должен быть таким:

def keyPressEvent(self, event):

if int(event.modifiers()) == (Qt.AltModifier + Qt.ShiftModifier):

if event.key() == Qt.Key\_Q:

# code

Теперь код выполнится, если нажмут клавишу **Q** при зажатых кнопках **Shift** и **Alt**.

Важно, что кнопки обработаются только в том случае, если окно активно. Активным называется окно, с которым пользователь работает в данный момент.

**Работа с мышью**

Для работы с мышью точно так же, как и в случае с клавиатурой, необходимо переопределять встроенные методы. Рассмотрим пример, в котором мы будем выводить координаты курсора на экран.

import sys

from PyQt5.QtWidgets import QWidget, QApplication, QLabel

class Example(QWidget):

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

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

self.initUI()

def initUI(self):

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

self.setWindowTitle('Координаты')

self.coords = QLabel(self)

self.coords.setText("Координаты:None, None")

self.coords.move(30, 30)

self.show()

def mouseMoveEvent(self, event):

self.coords.setText("Координаты:{}, {}".format(

event.x(), event.y()))

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

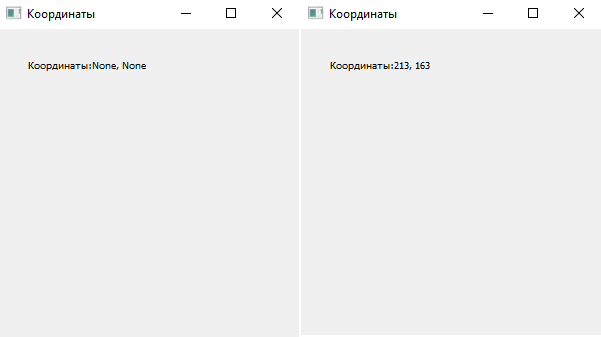
app = QApplication(sys.argv)

ex = Example()

sys.exit(app.exec\_())

Метод mouseMoveEvent срабатывает при изменении положения мышки. Функции .x() и .y() возвращают координаты текущего положения курсора в экранных координатах. Но, если вы запустите эту программу и попробуете просто поводить мышкой, ничего не произойдет. По умолчанию это событие активируется только при зажатой левой клавише мыши. Зажмите ее — и все заработает. А чтобы отлавливать положение мыши, даже когда клавиша не зажата, в метод init(self) надо добавить следующую строку:

self.setMouseTracking(True)



Кроме метода mouseMoveEvent, есть и другие. Например, метод mousePressEvent, который отвечает за обработку нажатия мыши. Кроме координат, полезно узнать, какая из кнопок была нажата, левая или правая. Для этого существует метод .button(). Сравнить можно со встроенными значениями из модуля PyQt5.QtCore.Qt: LeftButton, RightButton и даже MiddleButton.

import sys

from PyQt5.QtWidgets import QWidget, QApplication, QLabel

from PyQt5.QtCore import Qt

class Example(QWidget):

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

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

self.initUI()

def initUI(self):

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

self.setWindowTitle('Координаты')

self.coords = QLabel(self)

self.coords.setText("Координаты:None, None")

self.coords.move(30, 30)

self.btn = QLabel(self)

self.btn.setText("Никакая")

self.btn.move(30, 50)

self.show()

def mousePressEvent(self, event):

self.coords.setText("Координаты:{}, {}".format(

event.x(), event.y()))

if (event.button() == Qt.LeftButton):

self.btn.setText("Левая")

elif (event.button() == Qt.RightButton):

self.btn.setText("Правая")

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

app = QApplication(sys.argv)

ex = Example()

sys.exit(app.exec\_())

**Создание standalone-приложений**

В отличие от компилируемых языков программирования вроде C++, Python — язык интерпретируемый. И создание standalone-приложения — по своей сути «упаковка» виртуальной машины Python, заточенной под выполнение одной программы.

Мы будем рассматривать работу в операционной системе Windows, отличия в других OS (Linux и MacOS) не являются существенными.

Для создания .exe приложений из программ на Python есть несколько библиотек. Мы используем одну из самых популярных. Она называется pyinstaller.

Сначала необходимо ее установить:

pip install pyinstaller

Затем открываем командную строку, переходим в папку с нашим проектом и выполняем следующую команду:

pyinstaller --onefile --noconsole main.py

Что делают модификаторы:

* **onefile** собирает программу в один exe-файл. Это удобно, если программа небольшая, у нее мало аудиофайлов, графических файлов и пр., не относящихся к программному коду
* **noconsole**. Если не писать этот модификатор, то, кроме окна программы, будет открываться python-консоль. Иногда, например, для отладки, это удобно

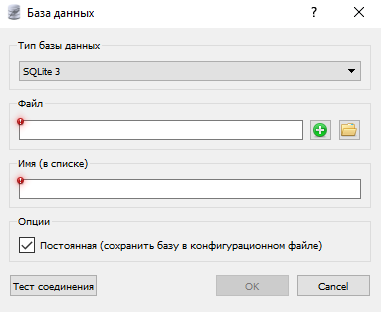
Есть и другие модификаторы. Например, **icon**. В нем можно указать путь до картинки в формате .ico, и она будет отображаться при просмотре программы в проводнике.

После выполнения этой команды в директории проекта появится две папки — build и dist. В папке dist будет лежать наша программа, или, если мы не указали модификатор **onefile**, кроме .exe, там будет еще множество файлов и папок. В нее следует добавить папки с графическими и другими файлами, которые используются в программе.

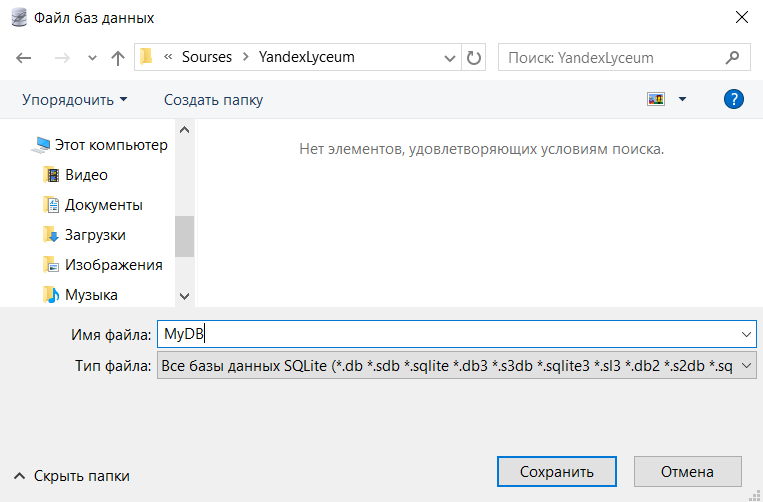
**Создание БД**

Чтобы создать простую БД, можно воспользоваться SQLiteStudio. Создадим базу данных для хранения любимых книг.

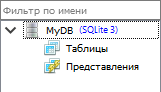
Открываем в верхнем меню вкладку **Database** и выбираем пункт **Add a database**.



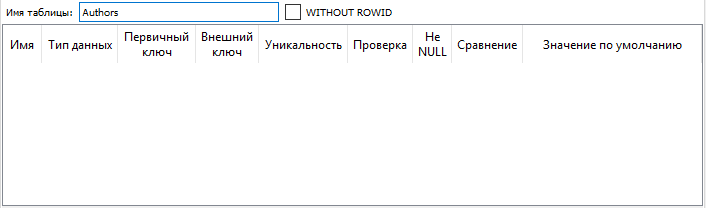
Нажимаем на **плюс** и вводим желаемое имя для нашей базы.



Сохраняем. В боковом меню должна появиться новая база данных.

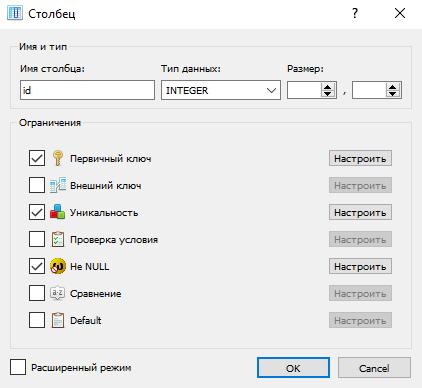


Теперь в нашей БД необходимо создать таблицы, в которых будет храниться информация. Для этого необходимо навести курсор на базу, нажать правую кнопку и выбрать пункт **Create a table**. Откроется окно для создания таблицы. Введем ее имя.

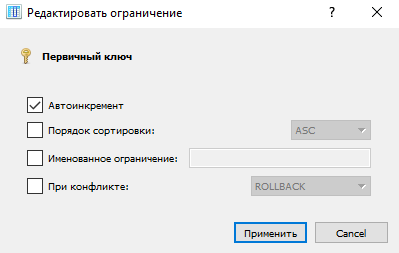


Двойным нажатием на белое поле откроем редактор столбца. Для того чтобы идентифицировать книги, создадим специальный столбец, назовем его id. Это будет число (то есть тип INTEGER), которое будет являться т. н. **первичным ключом**, то есть оно должно быть обязательно заполнено (не может быть NULL), уникально и для удобства он будет автоматически увеличиваться при добавлении новых значений (это называется автоинкремент).

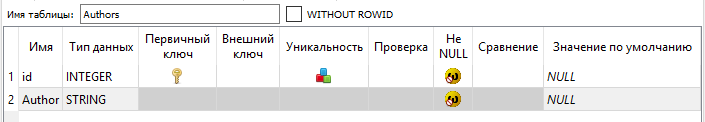
Настроим все эти значения:



Чтобы включить функцию автоинкремента, необходимо нажать на кнопку **Настроить** напротив **Первичного ключа**:



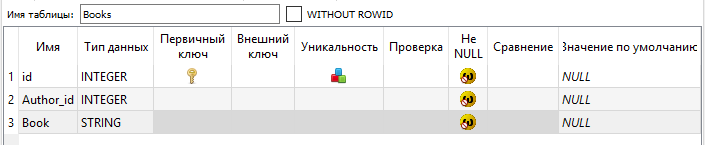
Теперь в нашей таблице есть первое поле — id. Добавим еще одно поле — Author, в котором будем хранить фамилию автора и его инициалы. Для него подойдет тип данных STRING.



После добавления всех полей необходимо применить все изменения. Затем выполнить commit. При нажатии на зеленую кнопку с галочкой будет показан запрос для создания таблицы, которую мы создали с помощью графического интерфейса.

Смело нажимаем ОК.

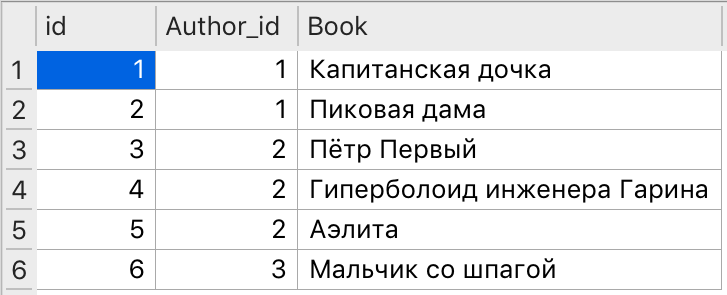
Теперь аналогичным образом создаем таблицу для хранения книг. Но будем указывать не ФИО автора, а его идентификатор из первой таблицы.



Теперь заполним наши таблицы. Таблицу авторов:



И таблицу книг:



Зачем мы использовали идентификаторы вместо полных ФИО? На это есть несколько причин:

1. Чтобы избежать опечаток и последующих проблем с выборками. Ведь ФИО можно записать очень по-разному.
2. Для экономии памяти. Одна ячейка типа INTEGER занимает во много раз меньше, нежели строковая ячейка.

Поздравляем! Вы создали вашу первую базу данных и первые две таблицы в ней. Подробнее про создание БД и таблиц можно прочитать [здесь](http://www.sqlitetutorial.net/).

**requirements.txt**

Очень часто большие проекты содержат массу зависимостей. Под зависимостями подразумеваются библиотеки, необходимые для корректной работы программы. Если программа распространяется не как standalone-приложение, а в исходном коде, хорошим тоном считается создавать специальный файл, в котором указываются зависимости. Принято называть этот файл **requirements.txt**. Как он выглядит? На каждой новой строке написано название пакета и необходимая версия в таком формате:

название\_пакета==его.версия

Например:

PyQt5==5.11

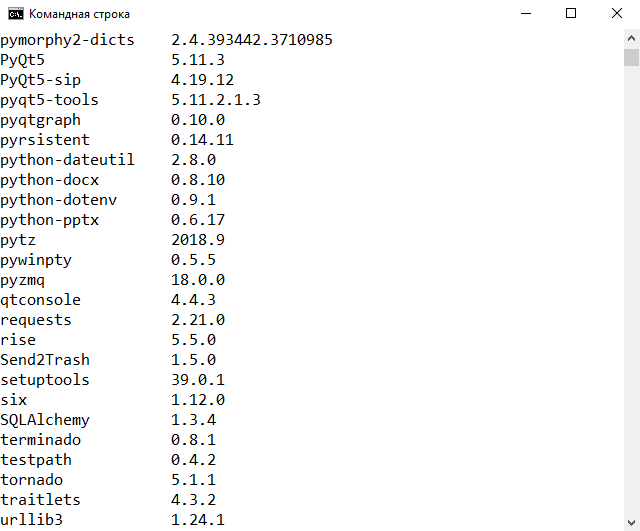
matplotlib==2.2.2

И чтобы установить все зависимости, достаточно в консоли (терминале) вызвать команду

pip install -r requirements.txt

Благодаря ключу -r pip автоматически установит внутренние зависимости. То есть, если в файле есть указание пакета X, которому для работы требуется пакет Y, который не указан в этом файле и не является стандартным, он все равно будет установлен.

Чтобы узнать версию уже установленных библиотек, достаточно воспользоваться командой pip list. Эта команда отображает список установленных пакетов и их версии.



[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