

# Создание настольных Python приложений

с графическим интерфейсом  
пользователя

Тимур Машнин

12+

**Тимур Машнин**  
**Создание настольных Python  
приложений с графическим  
интерфейсом пользователя**

*[http://www.litres.ru/pages/biblio\\_book/?art=65059196](http://www.litres.ru/pages/biblio_book/?art=65059196)*

*SelfPub; 2021*

*ISBN 978-5-532-96908-7*

### **Аннотация**

Python является наиболее популярным языком программирования, используемым для объектно-ориентированного программирования. И конечно, Python – это интерактивный язык программирования, который предоставляет широкий спектр возможностей для создания GUI (Graphical User Interface) и разработки настольных приложений. В этой книге вы познакомитесь с различными фреймворками Python для создания настольных приложений с графическим интерфейсом пользователя, такими как PyQt, PySide, Tkinter, Kivy, WxPython и Dear PyGUI.

# Содержание

Исходный код	4
Введение	5
Библиотека PyQt	13
Виджеты и компоновки PyQt	29
Конец ознакомительного фрагмента.	50

# **Тимур Машнин**

# **Создание настольных**

# **Python приложений**

# **с графическим**

# **интерфейсом пользователя**

## **Исходный код**

Исходный код к примерам можно скачать по адресу <https://github.com/novts/python-gui>.

# Введение



## Создание настольных Python приложений с GUI

### Введение

Де факто Python является наиболее популярным объектно-ориентированным языком программирования, который используется для веб-разработки и анализа больших данных.

И конечно, Python – это интерактивный язык программирования, который предоставляет широкий спектр возможностей для создания графического интерфейса пользователя.



## What is PyQt?

PyQt is a set of Python v2 and v3 bindings for [The Qt Company's](#) Qt application framework and runs on all platforms supported by Qt including Windows, macOS, Linux, iOS and Android. PyQt5 supports Qt v5. PyQt4 supports Qt v4 and will build against Qt v5. The bindings are implemented as a set of Python modules and contain over 1,000 classes.

PyQt4 and Qt v4 are no longer supported and no new releases will be made. PyQt5 and Qt v5 are strongly recommended for all new development.

### License

PyQt is dual licensed on all supported platforms under the GNU GPL v3 and the Riverbank Commercial License. Unlike Qt, PyQt is not available under the LGPL. You can purchase the commercial version of PyQt [here](#). More information about licensing can be found in the [License FAQ](#).

PyQt does not include a copy of Qt. You must obtain a correctly licensed copy of Qt yourself. However, binary wheels of the GPL version of PyQt5 are provided and these include a copy of the LGPL version of Qt.

### PyQt Components

A description of the components of PyQt5 can be found in the [PyQt5 Reference Guide](#).

A description of the components of PyQt4 can be found in the [PyQt4 Reference Guide](#).

PyQt – это библиотека графического фреймворка Qt для языка программирования Python.

А Qt кью-ти – это кроссплатформенный инструментарий для разработки программного обеспечения на языке программирования C++, такого как графические интерфейсы, работа с сетью, базами данных и XML.

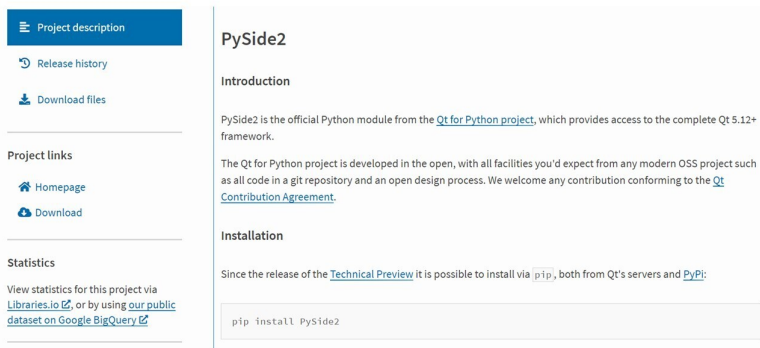
PyQt работает на всех платформах, поддерживаемых Qt – Linux и другие UNIX-подобные ОС, Mac OS и Windows.

И существуют 2 версии: PyQt5, поддерживающий Qt 5, и PyQt4, поддерживающий Qt 4.

PyQt практически полностью реализует возможности Qt, включая набор виджетов графического интерфейса, доступ к базам данных с помощью SQL, парсер XML и так далее.

PyQt также включает в себя Qt Designer— дизайнер гра-

фического интерфейса пользователя с генерацией Python кода из файлов, созданных в Qt Designer.



The screenshot shows the PySide2 project page on PyPI. On the left, there is a navigation menu with 'Project description' selected, and links for 'Release history', 'Download files', 'Project links' (including 'Homepage' and 'Download'), and 'Statistics'. The main content area has the title 'PySide2' and an 'Introduction' section. The introduction states that PySide2 is the official Python module from the Qt for Python project. Below this is an 'Installation' section with instructions on how to install via pip. A code block at the bottom shows the command: `pip install PySide2`.

Project description

Release history

Download files

Project links

Homepage

Download

Statistics

View statistics for this project via [Libraries.io](#), or by using [our public dataset on Google BigQuery](#)

## PySide2

### Introduction

PySide2 is the official Python module from the [Qt for Python project](#), which provides access to the complete Qt 5.12+ framework.

The Qt for Python project is developed in the open, with all facilities you'd expect from any modern OSS project such as all code in a git repository and an open design process. We welcome any contribution conforming to the [Qt Contribution Agreement](#).

### Installation

Since the release of the [Technical Preview](#) it is possible to install via `pip`, both from Qt's servers and [PyPI](#):

```
pip install PySide2
```

PySide – это также библиотека графического фреймворка Qt для языка программирования Python.

Основное отличие PySide от PyQt – это лицензии под которыми распространяются эти две обёртки Qt.

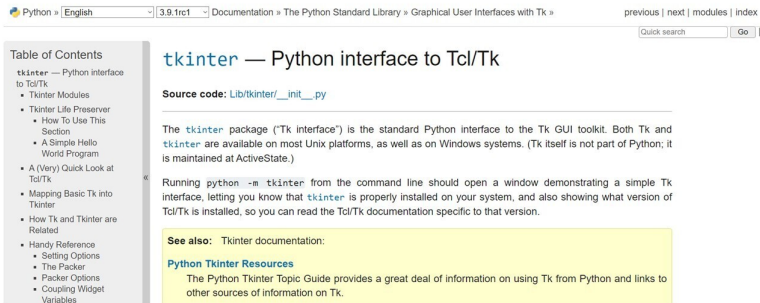
PyQt5 распространяется под GPL и коммерческой лицензией.

А PySide2 распространяется как Qt под GPL, LGPL и коммерческой лицензией.

То есть если вы пишете открытое ПО – можно использовать как PyQt5, так и PySide2.

Но если вы пишете закрытое/коммерческое ПО – бесплат-

но можно использовать только PySide2, а для использования PyQt5 потребуется покупать коммерческую лицензию.



The screenshot shows the Python documentation page for Tkinter. At the top, there is a navigation bar with "Python" and "English" dropdowns, a version selector set to "3.9.tre1", and a breadcrumb trail: "Documentation » The Python Standard Library » Graphical User Interfaces with Tk ». On the right, there are links for "previous", "next", "modules", and "index", along with a "Quick search" input field and a "Go" button. The main content area is titled "tkinter — Python interface to Tcl/Tk". Below the title, it lists the "Source code: Lib/tkinter/\_\_init\_\_.py". The text explains that the tkinter package is the standard Python interface to the Tk GUI toolkit, available on most Unix and Windows systems. It provides instructions on running `python -m tkinter` to see a demo window. A yellow callout box contains a "See also:" section with a link to "Tkinter documentation:" and a "Python Tkinter Resources" section with a link to "The Python Tkinter Topic Guide". On the left, a "Table of Contents" sidebar lists various topics like "Tkinter Modules", "Tkinter Life Preserver", and "Handy Reference".

Tkinter – это самая популярная библиотека для создания графического интерфейса пользователя или настольных приложений.

Tkinter – это комбинация стандартного графического интерфейса пользователя Tk и Python.

А Tk – это кроссплатформенная библиотека графического интерфейса с открытым исходным кодом.

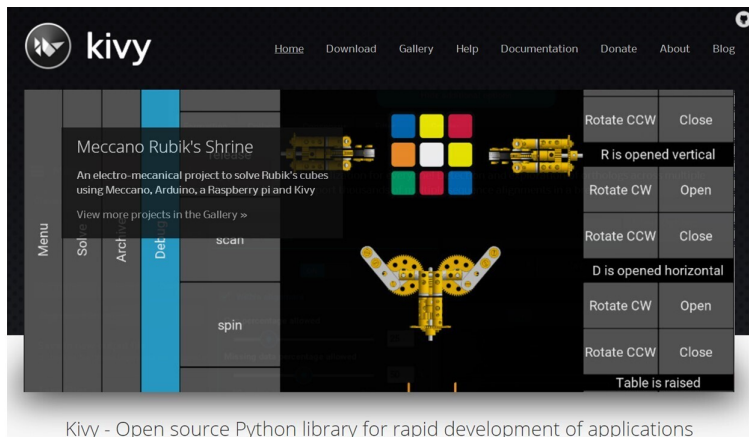
Tkinter входит в стандартную библиотеку Python.

И Tkinter – это свободное программное обеспечение, распространяемое под Python-лицензией.

Tkinter поставляется с хорошей документацией, что яв-

ляется основным ее достоинством.

И получить ответы на свои вопросы здесь легко, так как у Tkinter тысячи пользователей, потому что эта библиотека используется в течение очень долгого времени.



Kivy – это бесплатная среда Python с открытым исходным кодом для разработки кросс-платформенных приложений с поддержкой мультитач с пользовательским интерфейсом.

Kivy создана поверх OpenGL и для создания пользовательских интерфейсов дает возможность один раз написать код и запустить его на разных платформах Windows, MacOSX, Linux, Android, iOS и Raspberry.



**wxPython**

The GUI Toolkit for Python

[About](#) ▾ [News](#) [Downloads](#) [Documentation](#) ▾ [Support](#) ▾ [Developers](#) ▾ [Blog](#) ▾

## Welcome to wxPython!

This website is all about wxPython, the cross-platform GUI toolkit for the Python language. With wxPython software developers can create truly native user interfaces for their Python applications, that run with little or no modifications on Windows, Macs and Linux or other unix-like systems.

[Learn more](#)

WxPython – это обёртка библиотеки кроссплатформенного графического интерфейса пользователя wxWidgets, написанной на языке программирования C++.

Это еще одна из альтернатив Tkinter, которая поставляется вместе с Python.

И WxPython реализована в виде модуля расширения Python.



(This library is available under a free and permissive license however, if you Enjoy *Dear PyGui* please consider becoming a Sponsor)

python 3.6 | 3.7 | 3.8 | 3.9 | pypi v0.6.403 | downloads 364k

Embedded Build passing | static-analysis passing | build pending

Dear PyGui is a simple to use (but powerful) Python GUI framework. *Dear PyGui* is NOT a wrapping of [Dear ImGui](#) in the normal sense. It is a library built with *Dear ImGui* which simulates a traditional retained mode GUI (as opposed to *Dear ImGui*'s immediate mode paradigm).

## Dear

PyGui – это простая и легкая библиотека графического интерфейса пользователя, так как она полностью связана с языком программирования Python.

Dear PyGui предоставляет оболочку библиотеки C++ Dear ImGui, которая имитирует традиционный графический интерфейс.

Это кроссплатформенная среда приложений с графическим интерфейсом пользователя, которая отображает естественный графический интерфейс платформы.

Здесь мы перечислили наиболее широко используемые и лучшие доступные фреймворки графического пользовательского интерфейса Python.

И вы можете выбрать наиболее подходящую вам среду для разработки графического интерфейса Python.

Далее мы более подробно разберем каждую библиотеку.

# Библиотека PyQt



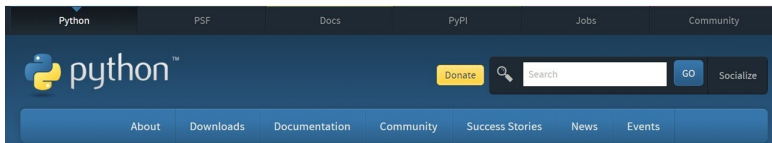
## Создание настольных Python приложений с GUI

### Библиотека PyQt

PyQt – это библиотека, которая позволяет использовать библиотеку графического интерфейса Qt в Python.

Сама библиотека Qt написана на C ++.

Самая последняя версия библиотеки – это PyQt5, и она поддерживает последнюю версию Qt5.

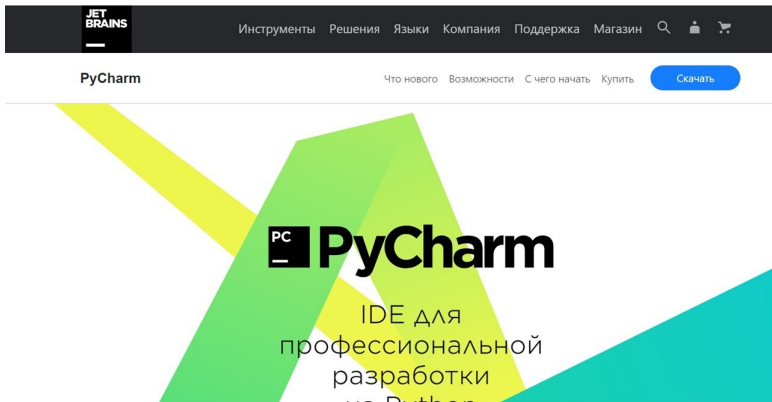


## Python 3.6.0

Release Date: Dec. 23, 2016

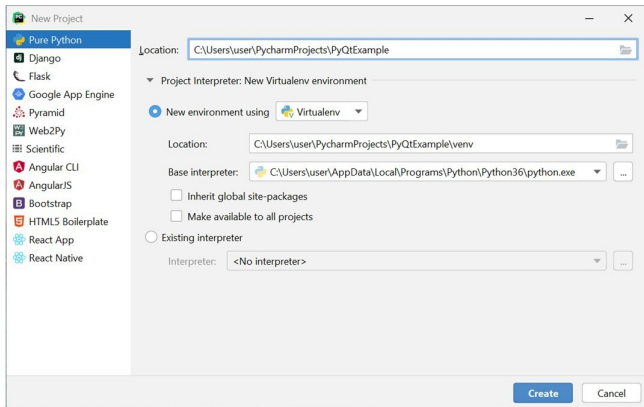
Python 3.6.0 was the initial feature release of Python 3.6.

Для работы с библиотекой PyQt5, установим питон 3.6.



Для разработки приложений питон с графическим интерфейсом пользователя мы будем пользоваться средой разработки

## PyCharm



Создадим питон проект. При этом будет автоматически создана виртуальная среда.

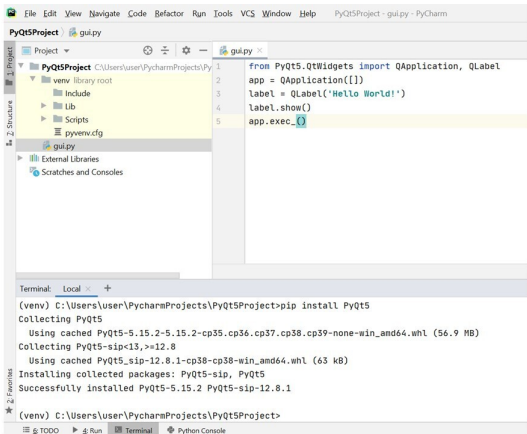
Виртуальная среда – это просто локальный каталог, содержащий библиотеки для конкретного проекта.

# pip install PyQt5

```
Terminal: Local x +
(venv) C:\Users\user\PycharmProjects\PyQt5Project>pip install PyQt5
Collecting PyQt5
  Using cached PyQt5-5.15.2-5.15.2-cp35-cp36-cp37-cp38-cp39-none-win_amd64.whl (56.9 MB)
Collecting PyQt5-sip<13,>=12.8
  Using cached PyQt5_sip-12.8.1-cp38-cp38-win_amd64.whl (63 kB)
Installing collected packages: PyQt5-sip, PyQt5
Successfully installed PyQt5-5.15.2 PyQt5-sip-12.8.1

(venv) C:\Users\user\PycharmProjects\PyQt5Project>
```

И для установки библиотеки PyQt просто наберите в окне терминала, в командной строке `pip install PyQt5`.



The screenshot shows the PyCharm IDE interface. The main editor window displays a Python script named `gui.py` with the following code:

```
1 from PyQt5.QtWidgets import QApplication, QLabel
2 app = QApplication([])
3 label = QLabel('Hello World!')
4 label.show()
5 app.exec_()
```

Below the editor, the Terminal window is open, showing the execution of the `pip install PyQt5` command. The output is identical to the first screenshot, indicating a successful installation of PyQt5 and its dependencies.

Далее в проекте создадим питон файл и наберем в нем код.

```
from PyQt5.QtWidgets import QApplication, QLabel

app = QApplication([])

label = QLabel('Hello World!')

label.show()

app.exec_()
```

Сначала мы загружаем PyQt с помощью оператора импорта.

И из PyQt5 виджетов импортируем QApplication, QLabel. Класс QApplication управляет потоком управления и основными настройками приложения с графическим интерфейсом.

Виджет QLabel обеспечивает отображение текста или изображения.

Затем мы создаем QApplication с помощью команды:  
`app = QApplication`

Это требование Qt – каждое приложение с графическим интерфейсом должно иметь ровно один экземпляр QApplication.

Здесь квадратные скобки представляют аргументы командной строки, переданные приложению.

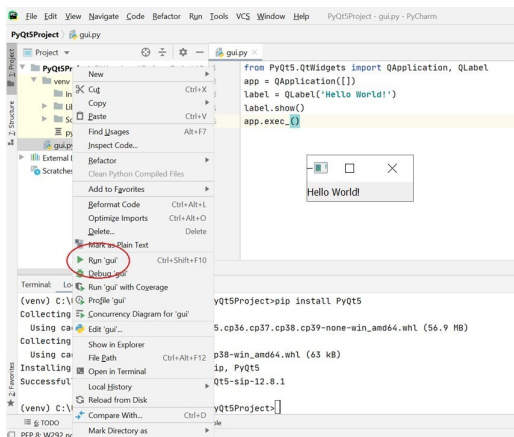
Так как наше приложение не использует никаких параметров, мы оставляем скобки пустыми.

Далее мы создаем простую метку 'Привет, мир!'.

И затем мы говорим Qt показать метку на экране с помощью команды show.

И последний шаг – это передать управление среде Qt и попросить ее «запустить приложение, пока пользователь не закроет его».

Это делается с помощью команды exec.

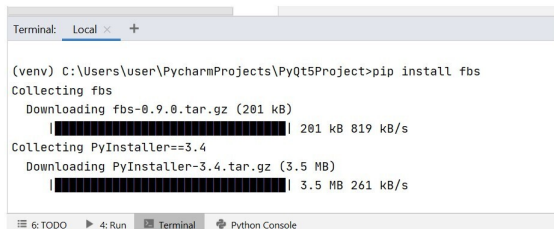


Далее нажмем правой кнопкой мыши на созданном питон файле и выберем команду

run

В результате будет запущено приложение и откроется окно с меткой.

pip install fbs



```
Terminal: Local x +  
  
(venv) C:\Users\user\PycharmProjects\PyQt5Project>pip install fbs  
Collecting fbs  
  Downloading fbs-0.9.0.tar.gz (201 kB)  
    |████████████████████████████████████████| 201 kB 819 kB/s  
Collecting PyInstaller==3.4  
  Downloading PyInstaller-3.4.tar.gz (3.5 MB)  
    |████████████████████████████████████████| 3.5 MB 261 kB/s
```

Теперь у нас есть приложение с графическим интерфейсом пользователя.

И оно работает на вашем компьютере.

Вопрос – как его передать другим людям, чтобы они тоже могли его запустить?

Вы можете попросить пользователей вашего приложения

установить Python и PyQt, а затем предоставить им свой исходный код.

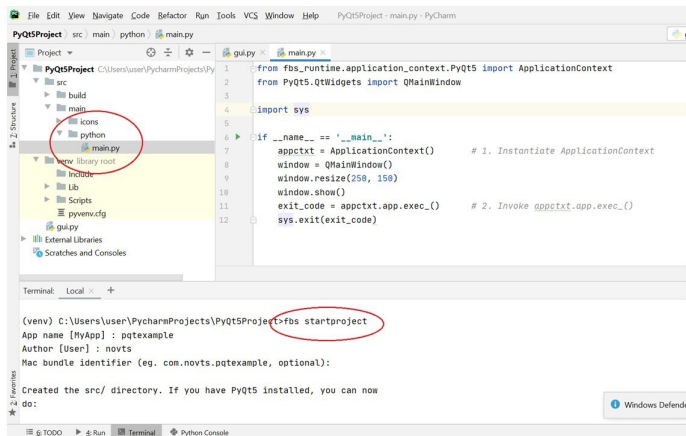
Но это очень неудобно.

Вместо этого нам нужен исполняемый файл, который другие люди могут запускать в своих системах, ничего не устанавливая.

В Python процесс превращения исходного кода в автономный исполняемый файл называется замораживанием.

Хотя существует множество библиотек, которые решают эту проблему, например PyInstaller, py2exe и так далее, здесь мы будем использовать библиотеку под названием fbs, которая позволяет создавать автономные исполняемые файлы для приложений PyQt.

Поэтому для начала установим библиотеку fbs.



Далее мы в терминале запускаем команду `fbс startproject`.

В результате выполнения которой будет создана папка `src/main/python/c` файлом `main.py`.

Команда `startproject` создает необходимую структуру папок для приложения `fbс`.

Если мы наберем в терминале команду `fbс run`, откроется просто пустое окно.

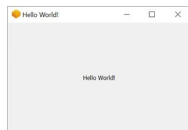
Теперь, как нам вставить в это окно нашу метку.

```
from fbс_runtime.application_context.PyQt5 import ApplicationContext
from PyQt5.QtWidgets import QMainWindow, QLabel
from PyQt5.QtCore import Qt

import sys

if __name__ == '__main__':
    appctx = ApplicationContext() # 1. Instantiate ApplicationContext
    window = QMainWindow()
    window.setWindowTitle("Hello World!")
    window.resize(500, 300)
    label = QLabel("Hello World!")
    label.setAlignment(Qt.AlignCenter)
    window.setCentralWidget(label)
    window.show()
    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    sys.exit(exit_code)
```

`fbс run`



Здесь вы можете заметить, что создание приложения с помощью `fbс` представляет новую концепцию – `ApplicationContext`.

При создании приложений `PyQt5` обычно используется

ряд компонентов или ресурсов, которые используются во всем приложении.

И `ApplicationContext` предоставляет центральное место для инициализации и хранения этих компонентов, а также предоставляет доступ к некоторым основным функциям `fbs`.

Объект `ApplicationContext` также создает и содержит ссылку на глобальный объект `QApplication`, доступный в `ApplicationContext.app`, так как каждое приложение Qt должно иметь один и только один объект `QApplication` для хранения цикла событий и основных настроек.

Теперь, чтобы вставить нашу метку, помимо `QMainWindow` импортируем метку.

Создадим метку и методом `setAlignment` установим ее по середине.

Методом `setCentralWidget` добавим метку в окно `QMainWindow`.

В результате после вызова команды `fbs run` мы увидим окно с меткой.

Wiki Documentation Forum Bug Reports Code Review Resource Center Qt Extensions

Qt Documentation

Qt 5.15 > Qt Widgets > C++ Classes > QMainWindow

Contents

- Public Types
- Properties
- Public Functions
- Public Slots
- Signals
- Reimplemented Protected Functions
- Detailed Description
  - Qt Main Window Framework

## QMainWindow Class

The QMainWindow class provides a main application window. [More...](#)

Header:	#include <QMainWindow>
qmake:	QT += widgets
Inherits:	QWidget

> [List of all members, including inherited members](#)

Более подробно про окно QMainWindow можно посмотреть в QT документации.

```
appctxt = ApplicationContext()
```

```
im=QImage(appctxt.get_resource('images/im.jpg'))
```

```
src/main/resources/base/images/
```

Теперь, приложениям обычно требуются дополнительные файлы данных помимо исходного кода – например, изображения.

И вот здесь может быть полезен `ApplicationContext`.

Вы можете поместить ресурсы приложения в папку `resources`, и чтобы упростить загрузку ресурсов из папки `resources`, `fbс` предоставляет метод `ApplicationContext.get_resource`.

Этот метод принимает имя файла, который можно найти в папке `resources`, и возвращает абсолютный путь к этому файлу.

И вы можете использовать этот возвращенный абсолютный путь, чтобы открыть файл.

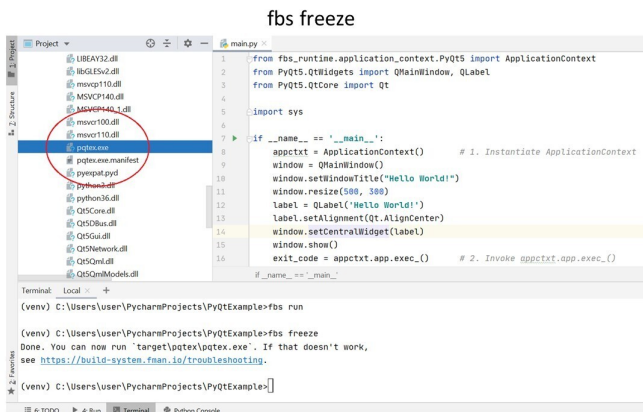
Папка `resources` должна содержать папку `base` плюс любую

комбинацию других папок.

Базовая папка содержит файлы, общие для всех операционных систем, в то время как папки для конкретных платформ могут использоваться для файлов, специфичных для данной ОС.

Теперь, далее мы можем использовать `fbz`, чтобы превратить файл питона в отдельный исполняемый файл.

fbz freeze



```
1 from fbs_runtime.application_context.PyQt5 import ApplicationContext
2 from PyQt5.QtWidgets import QMainWindow, QLabel
3 from PyQt5.QtCore import Qt
4
5 import sys
6
7 if __name__ == '__main__':
8     appctx = ApplicationContext() # 1. Instantiate ApplicationContext
9     window = QMainWindow()
10    window.setWindowTitle("Hello World!")
11    window.resize(500, 300)
12    label = QLabel("Hello World!")
13    label.setAlignment(Qt.AlignCenter)
14    window.setCentralWidget(label)
15    window.show()
16    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
17
18 if __name__ == '__main__':
```

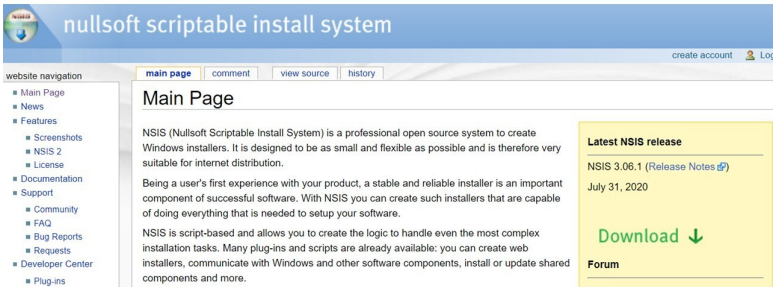
```
Terminal Local +
(venv) C:\Users\user\PycharmProjects\PyQtExample>fbz run

(venv) C:\Users\user\PycharmProjects\PyQtExample>fbz freeze
Done. You can now run 'target\ptex\ptex.exe'. If that doesn't work,
see https://build-system.fman.io/troubleshooting.
(venv) C:\Users\user\PycharmProjects\PyQtExample>
```

Для этого в терминале наберем команду `fbz freeze`.

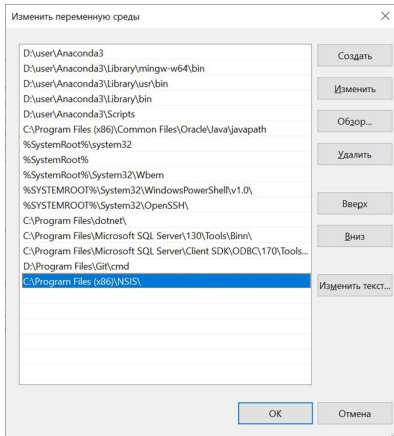
Эта команда помещает исполняемый двоичный файл в целевую папку текущего каталога.

Далее мы можем создать установщик приложения.



The screenshot shows the main page of the NSIS website. At the top, there is a blue header with the NSIS logo on the left and the text "nullsoft scriptable install system" in the center. On the right side of the header, there are links for "create account" and "Log". Below the header is a navigation bar with tabs for "main page", "comment", "view source", and "history". On the left side, there is a "website navigation" menu with a tree structure of links including Main Page, News, Features, Screenshots, NSIS 2, License, Documentation, Support, Community, FAQ, Bug Reports, Requests, Developer Center, and Plug-ins. The main content area has a heading "Main Page" and three paragraphs of text describing NSIS as a professional open source system for creating Windows installers, its design for internet distribution, its importance for user experience, and its script-based nature for handling complex tasks. On the right side, there is a yellow box titled "Latest NSIS release" containing the text "NSIS 3.06.1 (Release Notes)", the date "July 31, 2020", a green "Download" button with a downward arrow, and a "Forum" link.

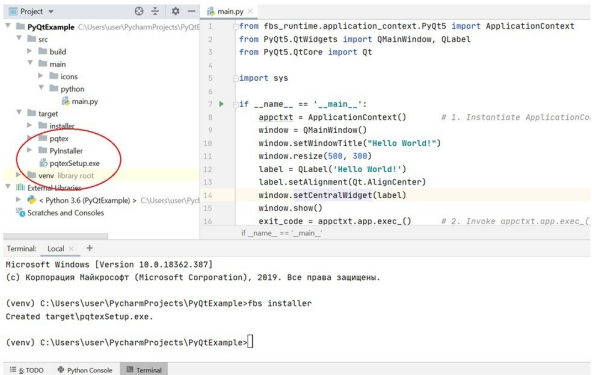
Но для начала мы должны установить NSIS – систему с открытым исходным кодом для создания установщиков Windows.



Также нужно добавить каталог NSIS в переменную среды Windows PATH.

После этого нужно перезапустить среду разработки PyCharm, чтобы она увидела эти изменения.

## fbс installer



```
Project ▾
└─ PyQtExample C:\Users\user\PycharmProjects\PyQtExample
   └─ src
      └─ build
         └─ icons
            └─ python
               └─ main.py
                  └─ target
                     └─ installer
                        └─ pqtex
                           └─ PyInstaller
                              └─ pqtexSetup.exe
                                 └─ library root
└─ External Libraries
   └─ Python 3.6 (PyQtExample) > C:\Users\user\PycharmProjects\PyQtExample\Python\Python36\Python.exe
      └─ Scratch and Consoles
```

```
1 from fbs_runtime.application_context.PyQt5 import ApplicationContext
2 from PyQt5.QtWidgets import QMainWindow, QLabel
3 from PyQt5.QtCore import Qt
4
5 import sys
6
7 if __name__ == '__main__':
8     appctx = ApplicationContext() # 1. Instantiate ApplicationCo
9     window = QMainWindow()
10    window.setWindowTitle("Hello World!")
11    window.resize(500, 300)
12    label = QLabel("Hello World!")
13    label.setAlignment(Qt.AlignCenter)
14    window.setCentralWidget(label)
15    window.show()
16    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    if __name__ == '__main__':
```

```
Terminal: Local > +
Microsoft Windows [Version 10.0.18362.387]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

(venv) C:\Users\user\PycharmProjects\PyQtExample>fbс installer
Created target\pqtexSetup.exe.

(venv) C:\Users\user\PycharmProjects\PyQtExample>
```

Далее создадим установщик с помощью команды `fbс installer`.

Эта команда помещает исполняемый двоичный файл в целевую папку текущего каталога.

Теперь вы можете отправлять его для установки приложения.

# Виджеты и компоновки PyQt



## Создание настольных Python приложений с GUI

### Виджеты и компоновки PyQt

Система компоновки Qt предоставляет простой и мощный способ организации дочерних виджетов внутри родительского виджета.

QBoxLayout - выстраивает дочерние виджеты по горизонтали или вертикали

QButtonGroup - контейнер для организации групп виджетов кнопок

QFormLayout - управляет формами виджетов ввода и связанными с ними метками.

QGraphicsAnchorLayout - связывает виджеты между собой

QGridLayout - размещает виджеты в сетке

QGroupBox - групповой фрейм окна с заголовком

QHBoxLayout - выстраивает виджеты по горизонтали

QLayout - базовый класс компоновки

QLayoutItem - абстрактный элемент, которым управляет QLayout

QSizePolicy - описывает политику изменения размера по горизонтали и вертикали

QSpacerItem - пустое место в макете

QStackedLayout - стек виджетов, в котором одновременно виден только один виджет

QVBoxLayout - выстраивает виджеты вертикально

QWidgetItem - элемент, представляющий виджет

Qt предоставляет набор классов управления компоновкой.

Эти компоновки автоматически позиционируют и изменяют размер виджетов.

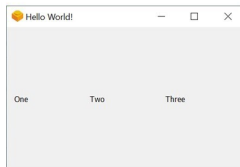
И все виджеты Qt могут использовать компоновки для управления своими дочерними элементами с помощью функции `setLayout`.

```
from fbs_runtime.application_context.PyQt5 import ApplicationContext
from PyQt5.QtWidgets import QWidget, QLabel, QHBoxLayout
from PyQt5.QtCore import Qt
import sys
```

```
if __name__ == '__main__':
    appctx = ApplicationContext() # 1. Instantiate ApplicationContext
    window = QWidget()
    window.setWindowTitle("Hello World!")
    window.resize(500, 300)

    layout = QHBoxLayout()
    layout.addWidget(QLabel('One'))
    layout.addWidget(QLabel('Two'))
    layout.addWidget(QLabel('Three'))
    window.setLayout(layout)

    window.show()
    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    sys.exit(exit_code)
```



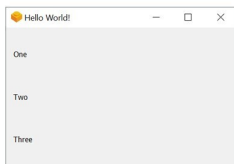
Здесь мы с помощью компоновки QHBoxLayout располагаем метки горизонтально в окне QWidget.

```
from fbs_runtime.application_context.PyQt5 import ApplicationContext
from PyQt5.QtWidgets import QWidget, QLabel, QHBoxLayout, QVBoxLayout
from PyQt5.QtCore import Qt
import sys
```

```
if __name__ == '__main__':
    appctx = ApplicationContext() # 1. Instantiate ApplicationContext
    window = QWidget()
    window.setWindowTitle("Hello World!")
    window.resize(500, 300)

    layout = QVBoxLayout()
    layout.addWidget(QLabel('One'))
    layout.addWidget(QLabel('Two'))
    layout.addWidget(QLabel('Three'))
    window.setLayout(layout)

    window.show()
    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    sys.exit(exit_code)
```



## Компоновка QVBoxLayout размещает метки вертикально.

```
from fbs_runtime.application_context.PyQt5 import ApplicationContext
from PyQt5.QtWidgets import QWidget, QLabel, QGridLayout
from PyQt5.QtCore import Qt
import sys
```

```
if __name__ == '__main__':
    appctx = ApplicationContext() # 1. Instantiate ApplicationContext
    window = QWidget()
    window.setWindowTitle("Hello World!")
    window.resize(500, 300)
```

```
    layout = QGridLayout()
    layout.addWidget(QLabel('One'),0,0,1,2, Qt.AlignHCenter)
    layout.addWidget(QLabel('Two'),1,0, Qt.AlignHCenter)
    layout.addWidget(QLabel('Three'),1,1, Qt.AlignHCenter)
    window.setLayout(layout)
```

```
    window.show()
    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    sys.exit(exit_code)
```



Компоновка QGridLayout располагает элементы в сетке.

При этом для каждого элемента можно указать строку, столбец, в которых должен находиться элемент.

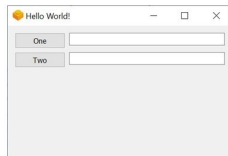
Также можно указать затем сколько строк и столбцов должен заполнять элемент.

```
from fbs_runtime.application_context.PyQt5 import ApplicationContext
from PyQt5.QtWidgets import QWidget, QFormLayout, QPushButton, QLineEdit
from PyQt5.QtCore import Qt
import sys
```

```
if __name__ == '__main__':
    appctx = ApplicationContext() # 1. Instantiate ApplicationContext
    window = QWidget()
    window.setWindowTitle("Hello World!")
    window.resize(500, 300)

    layout = QFormLayout()
    layout.addRow(QPushButton("One"), QLineEdit());
    layout.addRow(QPushButton("Two"), QLineEdit());
    window.setLayout(layout)

    window.show()
    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    sys.exit(exit_code)
```



Компоновка `QFormLayout` добавляет два виджета в строку, обычно `QLabel` и `QLineEdit` для создания форм.

В качестве резюме – виджеты могут иметь в качестве родительских только другие виджеты, но не компоновки.

Но вы можете вкладывать компоновки в родительскую компоновку с помощью метода `addLayout`, тогда внутренний макет становится дочерним по отношению к макету, в который он вставлен.

Теперь, модуль `Qt Widgets` предоставляет набор элементов пользовательского интерфейса для создания пользовательских интерфейсов приложения.

И все, что вы видите в приложении `PyQt`, представляет собой виджеты: кнопки, метки, окна, диалоговые окна, индикаторы выполнения и т. д.

```
from fbs_runtime.application_context.PyQt5 import ApplicationContext
from PyQt5.QtWidgets import QWidget, QLabel
from PyQt5.QtCore import Qt

import sys

if __name__ == '__main__':
    appctx = ApplicationContext() # 1. Instantiate ApplicationContext
    window = QWidget()
    window.setWindowTitle("Hello World!")
    window.resize(500, 300)

    label = QLabel(window)
    label.setText("Hello PyQt5")
    label.adjustSize()
    label.move(200, 100)

    window.show()
    exit_code = appctx.app.exec_() # 2. Invoke appctx.app.exec_()
    sys.exit(exit_code)
```

С меткой мы уже познакомились.

При создании метки, передавая в конструктор в качестве параметра объект окна, мы сообщаем, что метка является частью окна.

Метки имеют размер по умолчанию, и для длинных строк текста размер по умолчанию может быть слишком мал.

К счастью, у нас есть метод `adjustSize`, который автоматически настраивает размер метки.

В противном случае длинный текст будет отображаться на экране только частично.

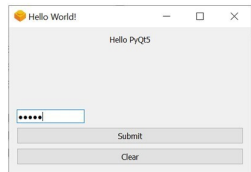
И метод `move` определяет начальную позицию метки от левого верхнего угла экрана.

```
def show():
    print(line.text())

line = QLineEdit()
line.setEchoMode(QLineEdit.Password)
line.setFixedWidth(140)

buttonS = QPushButton()
buttonS.setText("Submit")
buttonS.clicked.connect(show)

buttonC = QPushButton()
buttonC.setText("Clear")
buttonC.setIcon(QIcon('im.jpg'))
buttonC.clicked.connect(line.clear)
```



Каждому графическому интерфейсу нужен какой-нибудь способ получения ввода от пользователя.

В PyQt такой способ ввода данных – это использование виджета `QLineEdit`.

Или если вы хотите получить ввод многострочного текста, вам нужно использовать виджет `QTextEdit`.

И конечно ввод как правило используется вместе с кнопкой `QPushButton`.

Чтобы получить введенный текст из виджета `QLineEdit`, вы должны использовать метод `text`.

Здесь мы создаем кнопку `Submit`, которая вызывает функцию `show`, использующую метод `text` виджета `QLineEdit`.

Мы также создаем кнопку `Clear`, которая вызывает метод `clear` виджета `QLineEdit`, который удаляет все его содержи-

мое.

Метод `SetEchoMode` принимает несколько различных «режимов», одним из которых является режим пароля, который скрывает ввод.

Используя метод `setFixedWidth`, мы можем установить размер виджета `QLineEdit` в пикселях.

Значение по умолчанию для каждого виджета составляет около 100 пикселей.

Теперь о кнопке `QPushButton`.

Как следует из названия, это кнопка, которая запускает функцию при нажатии.

Кнопка, которая не связана с функцией, бесполезна.

Кнопки предназначены для подключения функции, которая будет выполняться после нажатия кнопки.

И такая функция подключается с помощью метода `clicked.connect`.

Также вы можете установить изображение на кнопку, с помощью виджета `QIcon`.

Просто передайте в него путь к файлу, и все готово.

```
def show():
    print(combo.currentText())

buttonS.clicked.connect(show)

combo = QComboBox()
combo.addItem("Python")
combo.addItem("Java")
combo.addItem("C++")
combo.setFixedWidth(140)

combo.addItems(["Python", "Java", "C++"])
```

Виджет `QComboBox` представляет собой раскрывающийся список элементов, из которых пользователь может выбрать свой вариант.

Преимущество этого виджета в том, что он занимает очень мало места на экране, при наличии большого списка элементов.

Здесь мы добавляем элементы в список по одному методом `addItem`.

Хотя мы можем добавить сразу все элементы кортежем с помощью метода `addItems`.

Зафиксировать ширину списка мы можем методом `setFixedWidth`.

Теперь, обработать выбор пользователем элемента в списке мы можем с помощью метода `currentText()`, который воз-

вращает элемент списка в виде строки.

```
def show():  
    print(check.text())  
    print(check.checkState())  
    print(check.isChecked())
```

```
check = QCheckBox()  
check.setText("Option1")  
check.stateChanged.connect(show)
```

Теперь перейдем к флажкам и радиокнопкам.

И здесь мы можем использовать сам флажок как кнопку, чтобы связать его с функцией обработки выбора флажка.

Мы делаем это с помощью метода `stateChanged.connect`.

Вы можете получить значение флажка с помощью метода `text`, который вернет текстовое значение флажка.

И вы можете использовать метод `checkState`, который возвращает целое число 0, если флажок не выбран и 2 – если он выбран.

Метод `isChecked` возвращает `true`, если флажок выбран.

```
def show():  
    print(radio1.isChecked(), radio1.text())  
    print(radio2.isChecked(), radio2.text())
```

```
radio1 = QRadioButton()  
radio1.setText("Option1")  
radio1.toggled.connect(show)  
layout.addWidget(radio1)
```

```
radio2 = QRadioButton()  
radio2.setText("Option2")  
radio2.toggled.connect(show)  
layout.addWidget(radio2)
```

В отличие от флажка, вы можете выбрать только одну радиокнопку из многих.

И здесь мы также можем использовать саму радиокнопку как кнопку, чтобы связать ее с функцией обработки выбора.

Мы делаем это с помощью метода `toggled.connect`.

Вы можете получить значение радиокнопки с помощью метода `text`, который вернет текстовое значение переключателя.

Есть еще один метод, который вы можете использовать — это метод `isChecked`, который возвращает `true` или `false`, показывая состояние выбора переключателя.

```
def show():
    print(text.toPlainText())

buttonS.clicked.connect(show)

text = QTextEdit()
text.setPlaceholderText("Enter some text here")
text.setUndoRedoEnabled(True)
layout.addWidget(text)
```

Как и виджет `QLineEdit`, виджет `QTextEdit` используется для ввода данных пользователем в виде текста.

Однако, в отличие от `QLineEdit`, который вводит только одну строку, `QTextEdit` позволяет ввести несколько строк текста.

Чтобы получить от пользователя введенный текст, мы можем использовать метод `toPlainText` и метод кнопки `clicked.connect`.

Метод `setPlaceholderText` используется для установки в виджет выделенного серым цветом текста, который исчезает при взаимодействии с виджетом.

Метод `setUndoRedoEnabled` отключает / включает возможность для пользователя использовать функции `Undo` и `Redo` с помощью клавиш `Ctrl + Z` и `Ctrl + Y`.

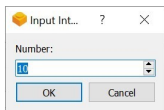
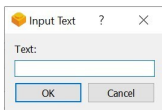
```
buttonD.clicked.connect(display)
```

```
def display():
```

```
    text, pressed = QDialog.getText(window, "Input Text", "Text: ", QLineEdit.Normal, "")
```

```
    if pressed:
```

```
        print(text)
```



```
intV, pressed = QDialog.getInt(window, "Input Integer", "Number:", 10, 0, 100, 1)
```

Помимо строки ввода и текстовой области, диалоговые окна обычно используются для ввода данных пользователя.

И в PyQt5 есть виджет `QInputDialog`, который позволяет создавать множество различных диалоговых окон для ввода данных различными способами.

И самое распространенное диалоговое окно – это ввод текста.

Оно представляет собой простое поле ввода и две кнопки, ОК и Отмена.

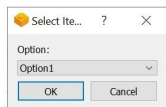
Функция `getText` возвращает два значения, поэтому мы устанавливаем две переменные для них.

Первая переменная `input` получает ввод пользователя, вторая переменная получает значение `True` или `False`, указы-

вающее, была ли нажата кнопка ОК или нет.

Далее есть диалоговое окно ввода целого числа.

Здесь вы можете использовать стрелки для изменения целочисленного значения или непосредственно ввести значение самостоятельно.



```
options = ("Option1", "Option2", "Option3")
```

```
option, pressed = QInputDialog.getItem(window, "Select Item", "Option:", options, 0, False)
```



```
decimal, pressed = QInputDialog.getDouble(window, "Input Decimal", "Value:", 10.00, 0, 100, 2)
```

Следующее окно – это окно выбора элемента.

И здесь мы сначала определяем список / кортеж строк, отображаемых в качестве элементов.

И последнее диалоговое окно – это окно ввода чисел с плавающей запятой.

Здесь вы также можете использовать стрелки для изменения значения или непосредственно ввести значение самостоятельно.

```

def show_popup():
    msg = QMessageBox()
    msg.setWindowTitle("Message Box")
    msg.setText("This is some random text")
    msg.setIcon(QMessageBox.Question)

    msg.setStandardButtons(QMessageBox.Cancel | QMessageBox.Ok)
    msg.setDefaultButton(QMessageBox.Ok)

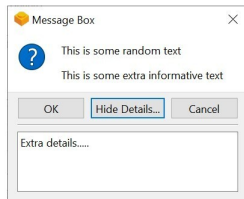
    msg.setDetailedText("Extra details.....")
    msg.setInformativeText("This is some extra informative text")

    msg.buttonClicked.connect(popup)

x = msg.exec_()

def popup(i):
    print(i.text())

```



`QMessageBox` – это виджет, который используется для создания диалоговых окон для отображения различных сообщений и кнопок.

Как правило, в приложении определяется метод, который открывает окно сообщения.

И этот метод связывается с какой-нибудь кнопкой или событием приложения.

Здесь метод `setWindowTitle` устанавливает заголовок окна сообщения.

Метод `setText` устанавливает текст под заголовком.

Метод `exec` открывает окно сообщения.

Окно сообщения имеет 4 различных типа значков, которые можно изменить с помощью метода `setIcon`.

В окне сообщения по умолчанию используется кнопка «ОК».

Однако на самом деле существует более десятка различных кнопок, которые `QMessageBox` предлагает для использования.

Используя функцию `setStandardButtons`, мы можем установить другой тип кнопки.

Когда окно открывается, вокруг кнопки «ОК» есть синий контур.

Это обозначение кнопки по умолчанию.

Используя функцию `setDefaultButton`, мы можем изменить кнопку по умолчанию.

По умолчанию у нас есть только одна область в `QMessageBox`, где мы показываем текст.

Однако есть еще две дополнительные области, которые мы можем разблокировать, чтобы добавить больше текста.

Первая область – это дополнительный текстовый раздел в самом окне сообщения, который мы можем определить с помощью функции `setInformativeText`.

Вторая область текста отображается в расширении окна.

При настройке этого раздела автоматически создается кнопка, которая используется для отображения этой области.

И для создания этой области потребуется функция `setDetailedText`.

Теперь, если у нас есть 3 разные кнопки в окне сообще-

ний, как мы узнаем, какая из них была нажата?

Используя метод `buttonClicked.connect`, мы можем вызвать запуск функции всякий раз, когда нажимается кнопка в окне сообщения.

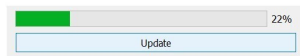
Здесь когда нажимается кнопка окна сообщений, она автоматически запускает функцию `popup`, объявленную в `buttonClicked.connect`.

И вызов функции `text` кнопки вернет ее текстовое значение.

```
def update():  
    value = prog_bar.value()  
    prog_bar.setValue(value + 1)
```

```
prog_bar = QProgressBar()  
prog_bar.setGeometry(50, 50, 250, 30)  
prog_bar.setValue(0)  
layout.addWidget(prog_bar)
```

```
buttonUp = QPushButton()  
buttonUp.setText("Update")  
buttonUp.clicked.connect(update)  
layout.addWidget(buttonUp)
```



`reset()` - сбрасывает значение на индикаторе выполнения.

`value()` - возвращает текущее целочисленное значение.

`setRange (n1, n2)` - определяет начальную и конечную точки.  
Значения по умолчанию - 0 и 100 соответственно.

Виджет `ProgressBar`, индикатор выполнения – это отличный способ визуализировать процесс выполнения длительных операций, таких как передача файлов, загрузка, выгрузка, копирование и т. д.

Здесь с помощью метода `setGeometry`, мы определяем расположение и размеры индикатора выполнения.

Первые два параметра представляют положение X и Y индикатора выполнения в окне.

Третий и четвертый параметры – это ширина и высота индикатора выполнения.

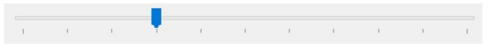
Метод `setValue` устанавливает значения индикатора.

Однако имейте в виду, что диапазон для индикатора выполнения составляет от 0 до 100.

Если вы хотите изменить этот диапазон, используйте метод `setRange`.

Метод `reset` сбрасывает значение на индикаторе выполнения.

Метод `value` возвращает текущее целочисленное значение индикатора.



```
slider = QSlider(Qt.Horizontal)
slider.setGeometry(50, 50, 200, 50)
slider.setMinimum(0)
slider.setMaximum(20)
slider.setTickPosition(QSlider.TicksBelow)
slider.setTickInterval(2)
slider.valueChanged.connect(show)
layout.addWidget(slider)

def show():
    print(str(slider.value()))
```

Виджет PyQt QSlider предоставляет графический интерфейс для выбора значения из диапазона различных значений.

У виджета Slider есть ползунок, который можно перемещать.

При перемещении ползунка выбранное значение соответственно изменяется.

По умолчанию ориентация слайдера вертикальная.

Для горизонтального ползунка вам нужно использовать значение Qt.Horizontal.

Затем нам нужно установить минимальный и максимальный диапазон с помощью методов setMinimum и setMaximum.

Используя метод setGeometry, мы определяем расположе-

ние и размеры слайдера.

Первые два параметра определяют положение ползунка по осям X и Y в окне.

Третий и четвертый параметры – это ширина и высота слайдера.

Затем нам нужно установить Ticks, маркеры по ползунку.

Вы можете выбрать место размещения тиков и установить интервал для расстояния между тиками.

Используя метод valueChanged, мы можем связать значение ползунка с функцией show.

Каждый раз, когда значение ползунка изменяется, вызывается функция show и показывается значение слайдера.



```
date = QDateEdit()
date.setMinimumDate(QDate(1900, 1, 1))
date.setMaximumDate(QDate(2100, 12, 31))
layout.addWidget(date)
```

```
def show():
    print(date.date().toPyDate())
```

Виджет PyQt5 QDateEdit – это интерактивный и визуаль-

ный способ ввести дату в приложении.

Здесь вы можете редактировать дату вручную или с помощью встроенных клавиш со стрелками.

И здесь вам не нужно назначать минимальное или максимальное значение, PyQt сам назначит значения по умолчанию.

Но вы можете сделать это с помощью методов `setMinimumDate` и `setMaximumDate`.

Чтобы получить введенную дату, вы можете использовать метод `date` и метод `toDate`, чтобы распечатать дату в более читаемом формате.

# Конец ознакомительного фрагмента.

Текст предоставлен ООО «ЛитРес».

Прочитайте эту книгу целиком, [купив полную легальную версию](#) на ЛитРес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.