**Лабораторная работа 2. QtDesigner**

Qt Designer — кроссплатформенная свободная среда для разработки графических интерфейсов (GUI) программ, использующих библиотеку PyQt, позволяет создавать графические интерфейсы пользователя при помощи ряда инструментов.

QtDesigner включена в сборку PyQT5. Но для ее использования необходимо установить библиотеку pyqt5-tools.

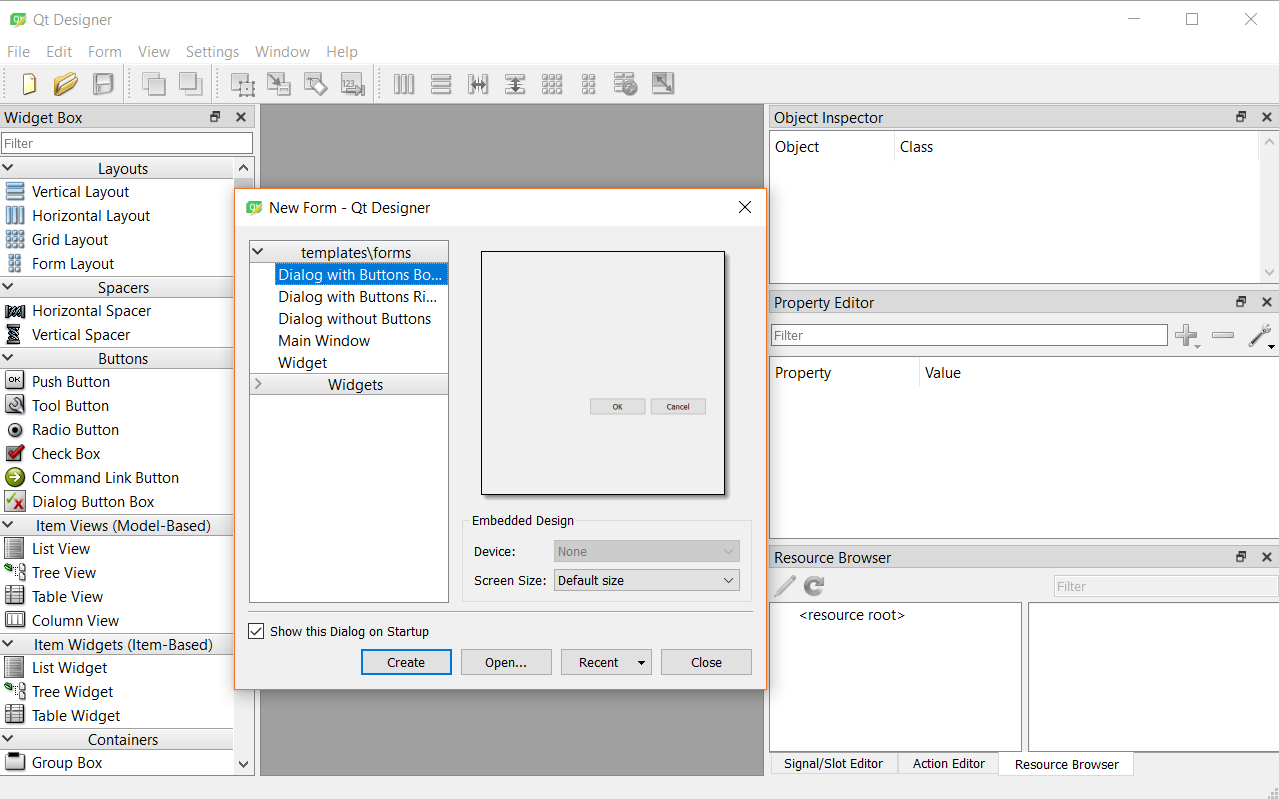
В MacOS библиотеку pyqt5-tools устанавливать не нужно.

pip install pyqt5-tools

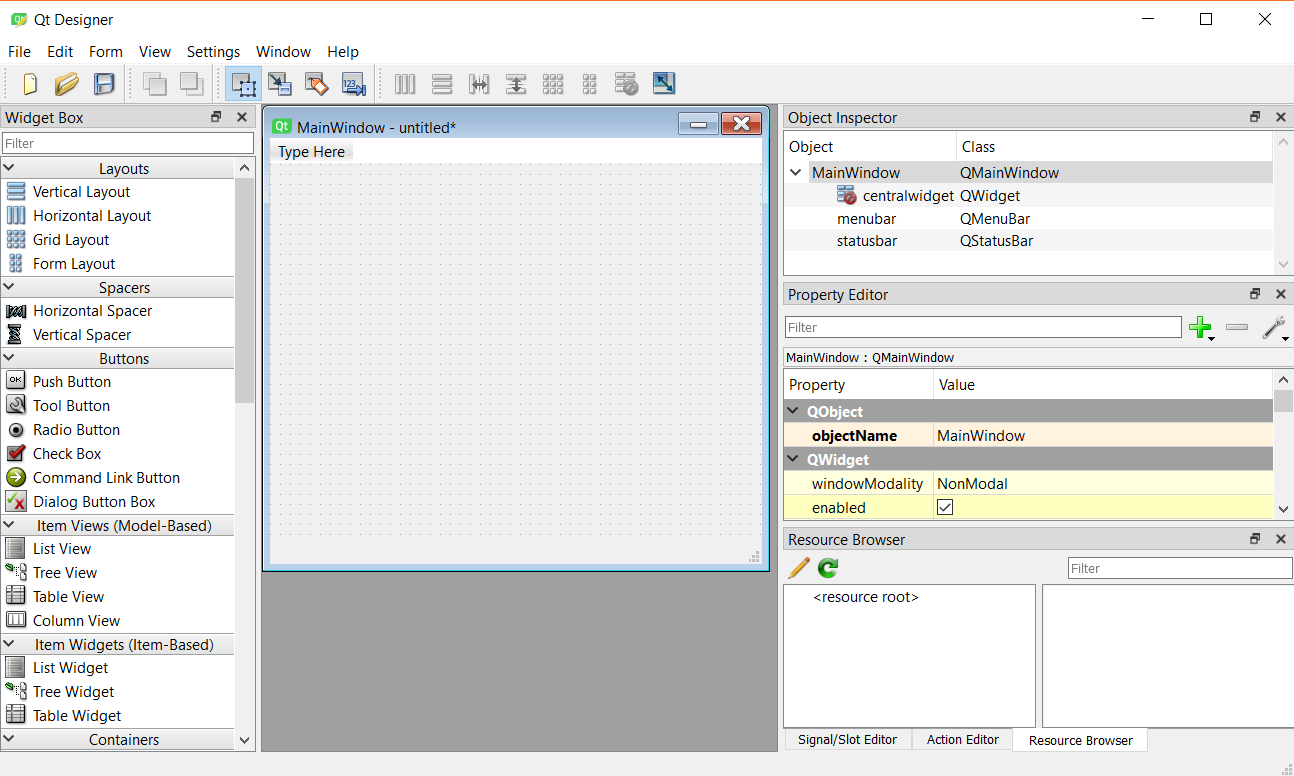
Если пакет установлен, то в консоли набираем команду

pyqt5-tools designer

Так же можно скачать и установить программу QtDesigner в виде отдельного приложения. Для этого перейдите по [ссылке](https://build-system.fman.io/qt-designer-download) и скачайте установщик приложения для вашей операционной системы.



При запуске открывается окно с предложением выбрать шаблон для формы или виджет, на основе которого мы будем делать свой интерфейс. Выберем **Main Window**. Откроется пустое окно.



Слева  располагается меню виджетов, **Widget Box**. В нем они сгруппированы в зависимости от их функциональности. Отдельно кнопки, отдельно виджеты для ввода данных и так далее.

Справа — Инспектор объектов (**Object Inspector**), Редактор свойств (**Property Editor**) и Браузер ресурсов (**Resourse Browser**). Остановимся на первых двух. В **Инспекторе объектов** отображается информация об используемых виджетах. Для каждого виджета указывается его имя и класс. Кроме того, можно увидеть иерархическую структуру всего интерфейса.

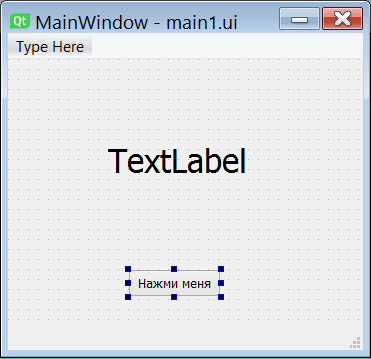
Чтобы разместить виджет на форме, его надо просто перетащить из меню виджетов. При этом информация об этом виджете автоматически появится в Инспекторе объектов.

**Редактор свойств** помогает изменять значения тех или иных атрибутов виджета (например, текст или размер). Расположение свойств меняется кнопкой с изображением гаечного ключа.

Сначала отображаются свойства, унаследованные от самого базового класса QObject, потом те, что получены от QWidget, и так далее, пока мы не придем к конкретному классу. Например, у PushButton, кроме вышеперечисленных, будут еще свойства QAbstractButton и QPushButton.

Попробуйте создать простейший интерфейс из кнопки (PushButton) и текстового поля (TextLabel), поиграйте с **Редактором свойств**, разберитесь, где поменять название и размеры кнопки, как изменить шрифт в TextLabel. Для сохранения в меню **Файл** выберите вкладку **Сохранить как**, найдите папку проекта и впишите имя.

Помните: название объекта (атрибут objectName) и текст, который может быть на нем показан (атрибут text), — это разные вещи.



Посмотрим, как выглядит наш дизайн с точки зрения компьютера. Для этого откроем созданный нами файл с помощью любого текстового редактора. Лучше использовать не просто Блокнот, а, например, SublimeText, Notepad++ или VS Code. Содержимое файла будет примерно вот таким:

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>554</width>

<height>379</height>

</rect>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="centralwidget">

<widget class="QPushButton" name="pushButton">

<property name="geometry">

<rect>

<x>210</x>

<y>200</y>

<width>93</width>

<height>28</height>

</rect>

</property>

<property name="text">

<string>Нажми меня</string>

</property>

<property name="checkable">

<bool>true</bool>

</property>

</widget>

<widget class="QLabel" name="label">

<property name="geometry">

<rect>

<x>130</x>

<y>80</y>

<width>281</width>

<height>61</height>

</rect>

</property>

<property name="font">

<font>

<pointsize>24</pointsize>

</font>

</property>

<property name="text">

<string>Текст на метке</string>

</property>

<property name="textFormat">

<enum>Qt::AutoText</enum>

</property>

</widget>

</widget>

<widget class="QMenuBar" name="menubar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>554</width>

<height>26</height>

</rect>

</property>

</widget>

<widget class="QStatusBar" name="statusbar"/>

</widget>

<resources/>

<connections/>

</ui>

Конечно, это не HTML, а язык разметки XML. Если присмотреться, мы увидим, что внутри этого документа описаны все наши виджеты и их свойства, а также показана их вложенность друг в друга.

**Подключение дизайна к программе**

Для подключения дизайна к программе есть два способа.

**Способ первый: загрузка ui-файла**

import sys

from PyQt5 import uic # Импортируем uic

from PyQt5.QtWidgets import QApplication, QMainWindow

class MyWidget(QMainWindow):

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

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

uic.loadUi('01.ui', self) # Загружаем дизайн

self.pushButton.clicked.connect(self.run)

# Обратите внимание: имя элемента такое же как в QTDesigner

def run(self):

self.label.setText("OK")

# Имя элемента совпадает с objectName в QTDesigner

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

app = QApplication(sys.argv)

ex = MyWidget()

ex.show()

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

Для загрузки ui-файла импортируйте класс uic, а затем в конструкторе вызовите метод loadUi, где одним из параметров указывается файл с интерфейсом. В  примере он называется 01.ui и лежит в той же папке, что и запускаемый скрипт.

После выполнения метода loadUi все виджеты становятся полями класса, имена для которых мы задали в редакторе свойств. Затем можно работать с ними (в примере, подключается обработчик нажатия кнопки).

**Способ второй: использование pyuic**

Второй способ — конвертирование ui-файла в класс Python. Для этого нужна консольная утилита **pyuic5**.

Чтобы ею воспользоваться, нужно открыть командную строку (терминал), перейти в ту папку, где лежит ваш ui-файл, и выполнить следующую команду:

pyuic5 ui\_file.ui -o ui\_file.py

Посмотрим, что находится внутри получившегося файла. (В зависимости от того, что вы сделали в QtDesigner, содержимое файла будет отличаться, но смысл останется неизменным.)

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file '01.ui'

#

# Created by: PyQt5 UI code generator 5.15.1

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(554, 379)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.pushButton = QtWidgets.QPushButton(self.centralwidget)

self.pushButton.setGeometry(QtCore.QRect(180, 200, 111, 28))

self.pushButton.setCheckable(True)

self.pushButton.setObjectName("pushButton")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(130, 80, 281, 61))

font = QtGui.QFont()

font.setPointSize(24)

self.label.setFont(font)

self.label.setTextFormat(QtCore.Qt.AutoText)

self.label.setObjectName("label")

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 554, 28))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.pushButton.setText(\_translate("MainWindow", "Нажми меня"))

self.label.setText(\_translate("MainWindow", "Текст на метке"))

Весь код написан не в инициализаторе, а в методе setupUi(). Утилита pyuic5 конвертирует XML-описание разметки из ui-файла в класс Python с кодом, создающим точно такой же интерфейс.

В получившемся файле нет кода для запуска приложения и никаких обработчиков событий. Есть соблазн дописать код в получившийся файл, но делать этого не стоит, и вот почему: никто из нас не идеален и не может написать интерфейс сразу так, чтобы в него никогда потом не пришлось вносить изменения. Поэтому если мы перемешаем код интерфейса и наш код с логикой, а потом поправим дизайн, после конвертации вся наша работа пропадет.

Ситуацию с разделением интерфейса и логики исполнения можно решить при помощи множественного наследования.

Создадим новый py-файл рядом с классом, который получился после конвертации интерфейса, вот с таким кодом:

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow

from ui\_01 import Ui\_MainWindow

# Наследуемся от виджета из PyQt5.QtWidgets и от класса с интерфейсом

class MyWidget(QMainWindow, Ui\_MainWindow):

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

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

# Вызываем метод для загрузки интерфейса из класса Ui\_MainWindow,

# остальное без изменений

self.setupUi(self)

self.pushButton.clicked.connect(self.run)

def run(self):

self.label.setText("OK")

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

app = QApplication(sys.argv)

ex = MyWidget()

ex.show()

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

Предками нашего класса MyWidget являются и QMainWindow, и Ui\_MainWindow. От первого унаследованы методы, а от второго — дизайн. В остальном работа схожа: мы вызываем метод setupUi из Ui\_MainWindow, а затем просто работаем с полями.

**Какой способ лучше?**

Оба метода уместны в определенных ситуациях.

Загрузка ui-файла очень удобна, когда приложение находится на стадии разработки и нам постоянно надо вносить какие-либо изменения в интерфейс. В таких случаях постоянная конвертация файла только замедляет процесс разработки. На самом деле конвертация происходит и в этом случае: метод uic.loadUi() выполняет эту конвертацию каждый раз при запуске приложения (а точнее каждый раз, когда выполняется эта строчка кода), что может сильно снизить производительность приложения.

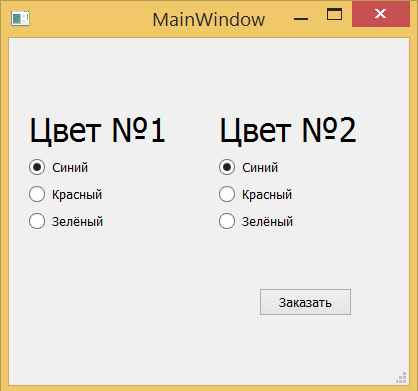
* На этапе разработки подключайте интерфейс с помощью ui-файлов
* Для релиза сконвертируйте весь интерфейс в классы Python

**Размещение виджетов**

Простое «накидывание» виджетов в QtDesigner или из кода работает неплохо. Проблемы начинаются, когда мы попытаемся каким-либо образом изменять размеры окна нашего приложения. В этом случае какие-то виджеты перестают частично или полностью попадать в поле зрения пользователя, и приложением становится неудобно пользоваться. Однако и с этой проблемой мы можем легко справиться, как программно, так и при помощи QtDesigner.

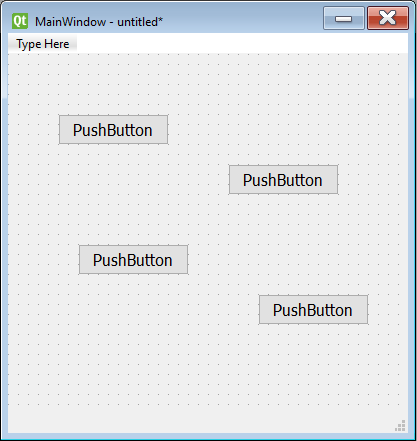
В QtDesigner можно размещать виджеты на экране не хаотично, а упорядоченно. Этот процесс называется разметкой. Для этого есть виджеты, которые называются Layout (Разметка): Vertical, Horizontal, Grid и Form Layout.

Но они нужны не только для красивого размещения элементов интерфейса, но и для создания групп из виджетов, например, для Radio Button (элемент интерфейса, который позволяет пользователю выбрать одну опцию (пункт) из предопределенного набора (группы)). Когда мы работаем с радиокнопками, можем выбрать только одну из них. А как поступить в том случае, когда у нас несколько логических групп, в каждой их которых нужно сделать выбор? Если мы просто разместим все Radio Button на нашем виджете, то никак не сможем выбрать два. А вот если часть из них поместить в какой-нибудь Layout — легко.

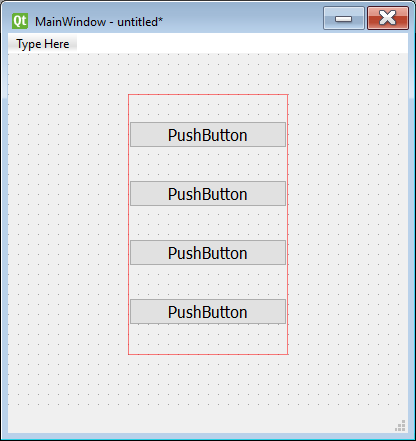


Посмотрим на то, как будут выглядеть привычные нам PushButton при применении к ним различных layout-ов.

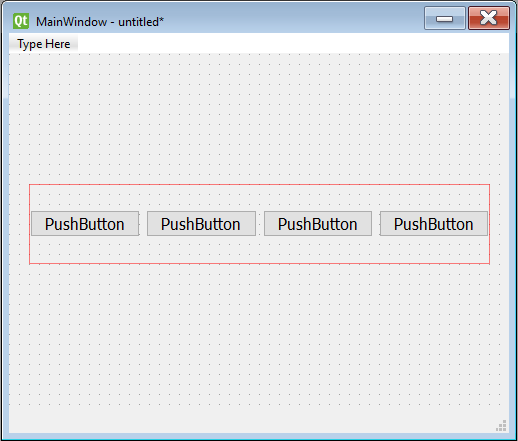
Разместим изначально кнопки на форме в случайном порядке.



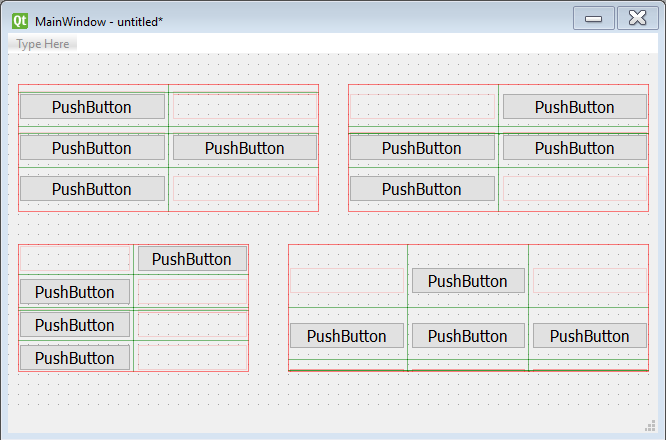
Затем поместим их внутрь **Vertical Layout**.



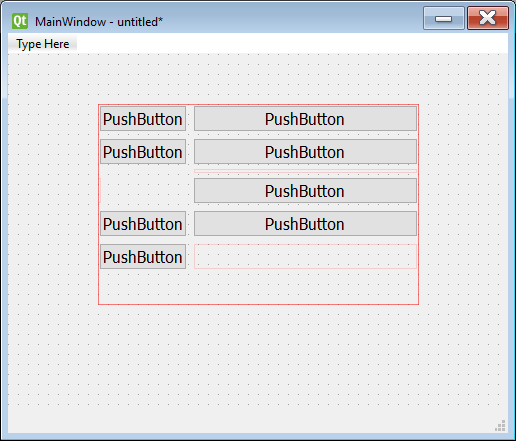
Теперь — в Horizontal Layout.



В случае использования **Grid Layout** появляются различные варианты:



А Form Layout является подвидом Grid Layout. У него фиксированное число столбцов, равное двум. Как можно догадаться по названию, такой layout удобнее всего использовать для создания форм, где в первый столбец размещаются подсказки для полей, а во второй — виджеты для ввода значения.



Создавать **Layout** можно и из кода напрямую. Для добавления элементов в**Layout** у них всех существует метод addWidget(), который принимает виджет, который надо разместить, и дополнительные настройки размещения. Например, чтобы добавить виджет в **Grid Layout**, можно написать такой код: grid.addWidget(elem, x, y). В качестве параметров метода выступают виджет, а также координаты ячейки, по которым его нужно расположить. Подробнее можно почитать в [документации](https://doc.qt.io/qtforpython/PySide2/QtWidgets/QLayout.html).

Кроме **Layout**, существуют и виджеты для создания виртуальных групп (вроде QButtonGroup). Чаще всего они нужны, когда у нас есть много виджетов с одинаковыми функциями. Например, у нас много **Radio Button**, но они все отвечают за один и тот же параметр — цвет. Чтобы объединить **Radio Button** в группу в QtDesigner, необходимо выделить их, нажать на правую кнопку мыши, кликнуть пункт меню **Assign to button group** и выбрать либо пункт создания новой группы (**New button group**), либо уже созданную. Чтобы «повесить» на всю группу обработчик событий, нужно вызвать метод connect.

self.buttonGroup.buttonClicked.connect(self.run)

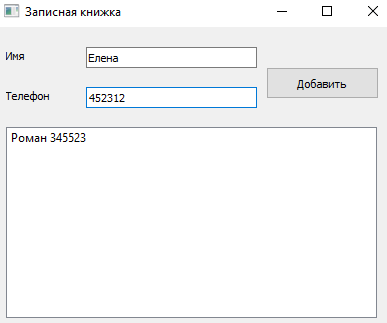
А чтобы получить список всех кнопок группы, применить метод .buttons().

**Задание 1.**

Вам дан файл calc.ui, в котором реализован интерфейс калькулятора. Изучите его, подключите к своей программе любым из способов и реализуйте все его функции. Если какое-то выражение вычислить не получается (например, квадратный корень для отрицательного числа), должно выводиться соответствующее сообщение.

**Задание 2.**

Напишите программу «Записная книжка» с графическим пользовательским интерфейсом на PyQT, используя необходимые виджеты. Пользователь должен иметь возможность ввести имя контакта и его номер. После добавление данные должны отображаться в List Widget.



**Задание 3.**

Используя виджеты [QCalendar](https://doc.qt.io/qt-5/qcalendarwidget.html" \t "_blank), [QTimeEdit](https://doc.qt.io/qt-5/qtimeedit.html" \t "_blank), [QListWidget](https://doc.qt.io/qt-5/qlistwidget.html" \t "_blank), и другие (при необходимости), напишите программу-ежедневник с графическим пользовательским интерфейсом на PyQT.

Пользователь должен иметь возможность ввести название события, выбирать дату и время. После нажатия на кнопку «Добавить» событие должно добавляться в QListWidget. События в QListWidget должны быть отсортированы по возрастанию даты.

