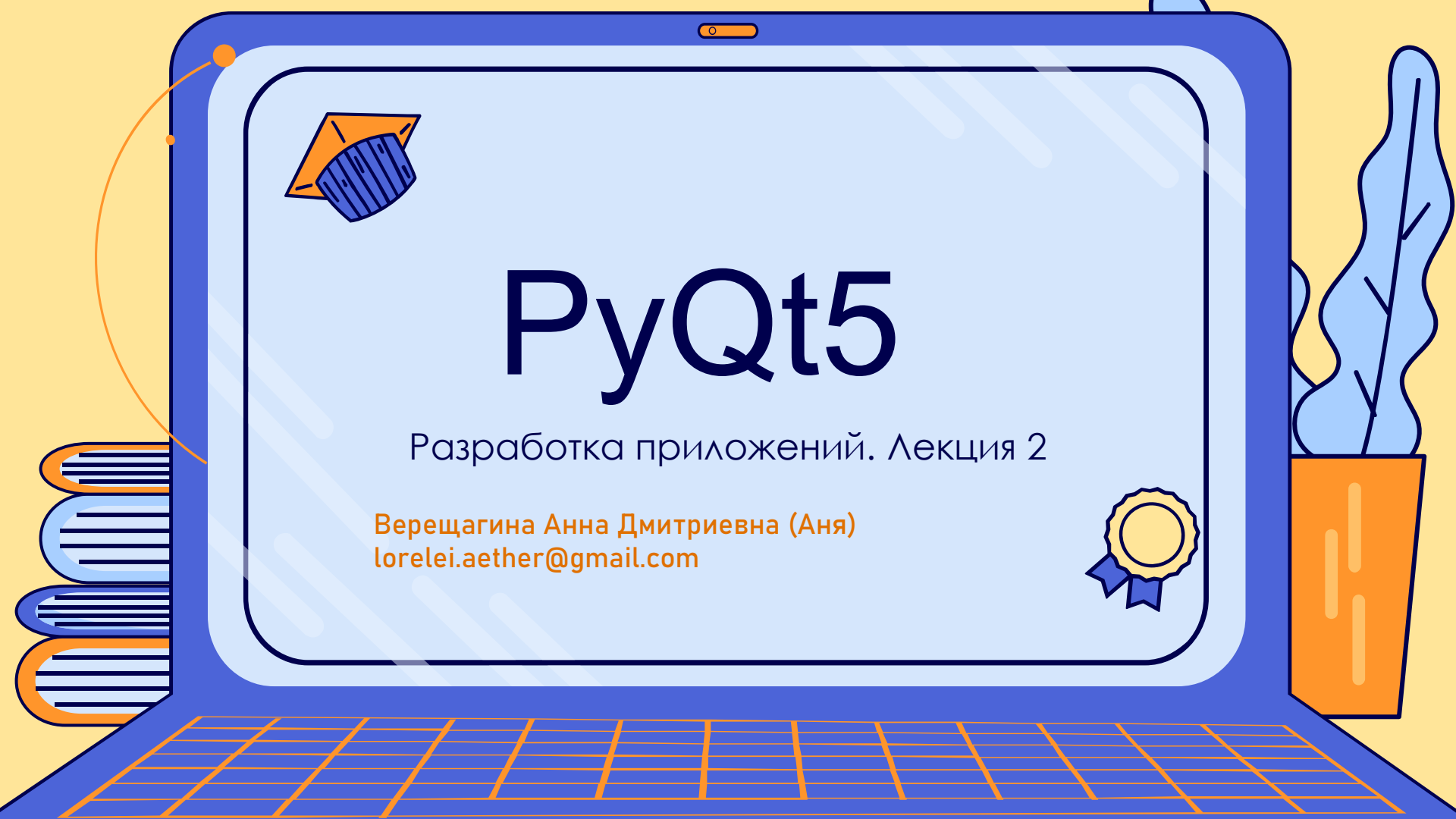


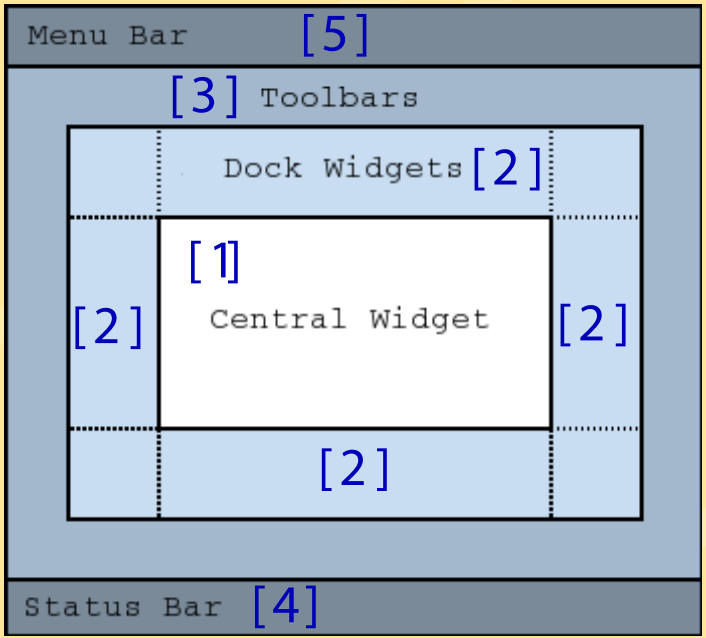
PyQt5

Разработка приложений. Лекция 2

Верещагина Анна Дмитриевна (Аня)
lorelei.aether@gmail.com

QMainWindow

Виджет, который обычно используется как **главное окно.**



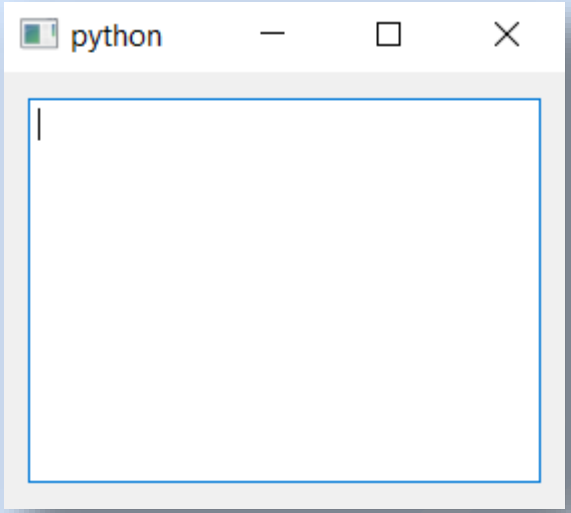
QMainWindow отличается от QWidget практически всем. Оно содержит:

- Центральный виджет [1];
- «Плавающие» боковые виджеты [2];
- Поле с инструментами [3];
- Строку состояния [4];
- Меню [5].

QTextEdit

```
from PyQt5.QtWidgets import
QTextEdit

textedit = QTextEdit()
```

Output:



Полезные методы QTextEdit

textedit.setPlainText(str)	Установить текст (строку) в QTextEdit.
textedit.toPlainText()	Получить текст (строку) из QTextEdit.
textedit.clear()	Очистить QTextEdit.
textedit.append(str)	Присоединить текст (строку) в QTextEdit.
textedit.setAlignment(Qt.Align…)	Установить выравнивание QTextEdit.
textedit.textCursor()	Создание данных о курсоре. Открывает множество полезных опций.

Построение QMainWindow

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QPixmap, QFont, QIcon
from PyQt5.QtCore import Qt

class Window1(QMainWindow): # Создаем класс от родителя QMainWindow - это будет наше первое окно.
    def __init__(self):
        super().__init__() # Наследуем методы и атрибуты класса QMainWindow.

        self.setWindowTitle("QMainWindow и переход между окнами") # Называем наше окошко. Оно будет высвечиваться в верхней полосе.
        self.resize(800, 400) # Назначаем размер окна. Оно будет растягиваться само, если добавлять туда большие элементы.
        self.setWindowIcon(QIcon('icon.png')) # Назначаем иконку для окна.
        self.setFont(QFont("Arial", 12)) # Назначаем шрифт.
        self.w = None # В эту переменную мы поместим наше следующее окно.

        self.initUI()

    def initUI(self): # Основная функция для построения нашего окна.

        self.textedit = QPlainTextEdit() # Создаем простейший редактор текста.
        self.setCentralWidget(self.textedit) # Назначаем редактор текста главным виджетом нашего окна QMainWindow.


def open_window(): # Наша функция для открытия приложения.
    app = QApplication(sys.argv) # Создаем наше приложение с аргументами из командной строки.
    wind = Window1() # Создаем экземпляр первого окна.
    wind.show() # Показываем наше окно.
    sys.exit(app.exec_()) # Заканчиваем работу приложения в случае выхода.

open_window()
```

Создание простого действия

exitAction = QAction(QIcon('exit.png'), 'Выход', self)	Оформляем действие, кладем его в переменную. Ставим иконку и называем его.
exitAction.setShortcut('Ctrl+Q')	Задаем горячие клавиши для этого действия.
exitAction.setStatusTip('Выход из приложения')	Назначаем подсказку, которая будет высвечиваться в строке состояния.
exitAction.triggered.connect(self.close)	Привязываем действие к функции.

Создание основных элементов QMainWindow

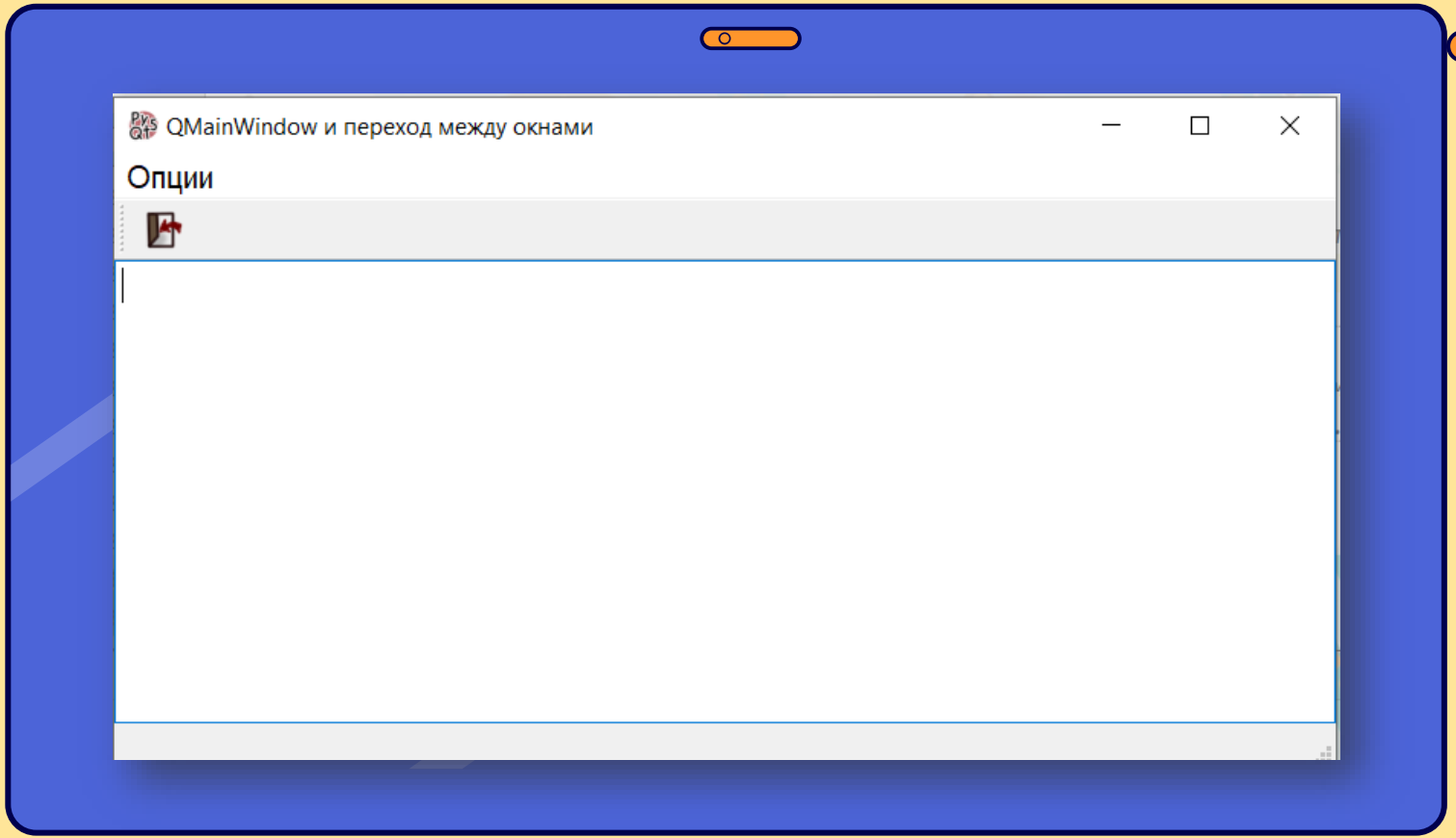
self.statusBar()	Создание строки состояния.
menubar = self.menuBar()	Создание меню.
optionMenu = menubar.addMenu('&Опции')	Создание в меню категорию «Опции».
optionMenu.addAction(exitAction)	Добавление нашего действия в меню «Опции».
toolbar = self.addToolBar('Выход')	Создание инструмента в тул-баре «Выход».
toolbar.addAction(exitAction)	Добавление нашего действия к инструменту «Выход».

Построение QMainWindow

```
def initUI(self): # Основная функция для построения нашего окна.

    self.textedit = QPlainTextEdit() # Создаем простейший редактор текста.
    self.setCentralWidget(self.textedit) # Назначаем редактор текста главным виджетом нашего окна QMainWindow.

    exitAction = QAction(QIcon('exit.png'), 'Выход', self) # Создаем действие "выход". Выбираем для него иконку и название.
    exitAction.setShortcut('Ctrl+Q') # Выбираем горячие клавиши для выхода.
    exitAction.setStatusTip('Выход из приложения') # Создаем подсказку к нашему действию.
    exitAction.triggered.connect(self.close) # Привязываем функцию к действию. В данном случае это уже существуюший метод .close.

    self.statusBar() # Добавляем в окно строку состояния.
    menubar = self.menuBar() # Добавляем в окно верхнее меню.

    optionMenu = menubar.addMenu('&Опции') # Содаем категорию "опции" в меню.
    optionMenu.addAction(exitAction) # Добавляем в категорию "опции" наше действие.

    toolbar = self.addToolBar('Выход') # Добавляем в поле с инструментами кнопку "Выход"...
    toolbar.addAction(exitAction) # ...И действие к нему.
```

Сохраните эту иконку в папку с кодом под названием «exit.png», и она появится!

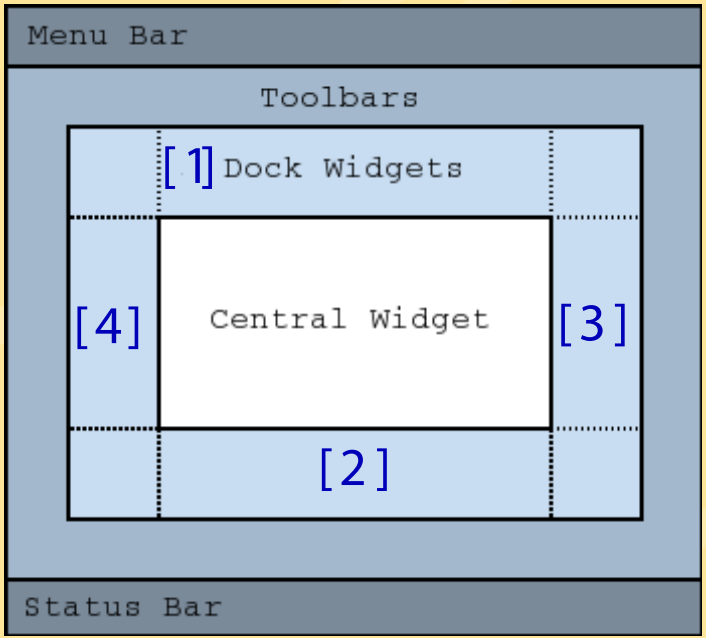
Output:



Создание QDockWidget

labelwidget = QDockWidget(self)	Создаем «плавающий» виджет. Сюда поместим надпись.
labelwidget.setWindowTitle(«Подсказка»)	Называем «плавающий» виджет.
label = QLabel(«Введите сюда текст!»)	Создаем нашу надпись.
labelwidget.setWidget(label)	Устанавливаем надпись в «плавающий» виджет.
self.addDockWidget(Qt.BottomDockWidgetArea, labelwidget)	Добавляем «плавающий» виджет в нижнюю область.

Области QDockWidget



[1] Верхняя - Qt.TopDockWidgetArea

[2] Нижняя - Qt.BottomDockWidgetArea

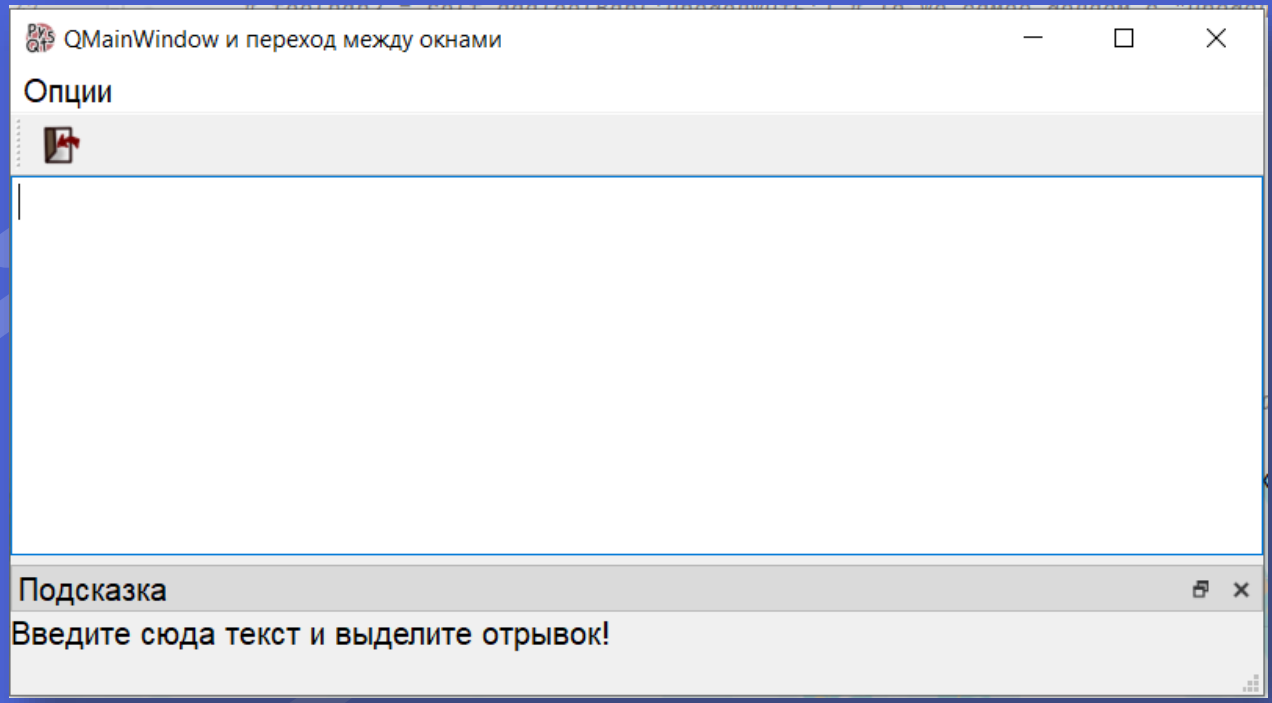
[3] Правая - Qt.RightDockWidgetArea

[4] Левая - Qt.LeftDockWidgetArea

Построение QMainWindow

```
def initUI(self): # Основная функция для построения нашего окна.

    self.textedit = QPlainTextEdit() # Создаем простейший редактор текста.
    self.setCentralWidget(self.textedit) # Назначаем редактор текста главным виджетом нашего окна QMainWindow.
    labelwidget = QDockWidget(self)  # Создаем экземляр док-виджета. Это перемещаемый объект, уникальный для MainWindow.
    labelwidget.setWindowTitle("Подсказка")  # Даем название нашему док-виджету.
    label = QLabel("Введите сюда текст и выделите отрывок!")  # Создаем нашу надпись.
    labelwidget.setWidget(label)  # Устанавливаем наш готовый виджет в экземпляр док-виджета.
    self.addDockWidget(Qt.BottomDockWidgetArea, labelwidget)  # Добавляем в окно наш док-виджет и выбираем нижнюю область для него.

    exitAction = QAction(QIcon('exit.png'), 'Выход', self) # Создаем действие "выход". Выбираем для него иконку и название.
    exitAction.setShortcut('Ctrl+Q') # Выбираем горячие клавиши для выхода.
    exitAction.setStatusTip('Выход из приложения') # Создаем подсказку к нашему действию.
    exitAction.triggered.connect(self.close) # Привязываем функцию к действию. В данном случае это уже существуюший метод .close.

    self.statusBar() # Добавляем в окно строку состояния.
    menubar = self.menuBar() # Добавляем в окно верхнее меню.

    optionMenu = menubar.addMenu('&Опции') # Содаем категорию "опции" в меню.
    optionMenu.addAction(exitAction) # Добавляем в категорию "опции" наше действие.

    toolbar = self.addToolBar('Выход') # Добавляем в поле с инструментами кнопку "Выход"...
    toolbar.addAction(exitAction) # ...И действие к нему.
```

Output:


QMainWindow и переход между окнами
Опции
Подсказка
Введите сюда текст и выделите отрывок!

QMessageBox

```
from PyQt5.QtWidgets import QMessageBox

reply = QMessageBox.question(self, 'Выход', "Вы уверены, что хотите выйти?", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
```

reply = QMessageBox.question	Вызываем сообщение и кладем ответ юзера в переменную reply.
«Выход»	Заголовок QMessageBox.
«Вы уверены, что хотите выйти?»	Текст вопроса в QMessageBox.
QMessageBox.Yes | QMessageBox.No	Варианты ответа юзера.
QMessageBox.No	Ответ по умолчанию.

Warning & information

```
from PyQt5.QtWidgets import QMessageBox

QMessageBox.warning(self, 'Ошибка', 'Что-то пошло не так.')
QMessageBox.information(self, 'Получилось', 'Успешно обработано!')
```

QMessageBox.warning	Вызывается сообщение с предупреждением.
«Ошибка»	Заголовок QMessageBox.
«Что-то пошло не так»	Текст сообщения в QMessageBox.

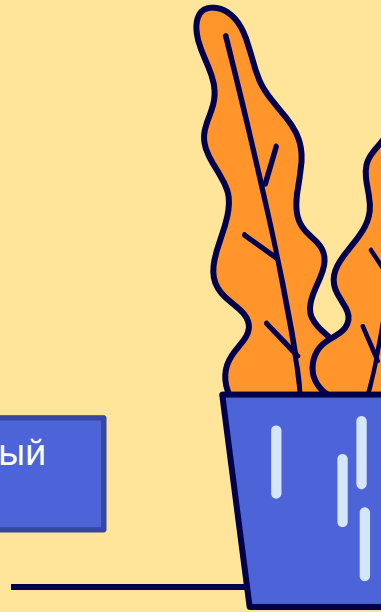
QMessageBox.information	Вызывается сообщение с информацией.
«Получилось»	Заголовок QMessageBox.
«Успешно обработано!»	Текст сообщения в QMessageBox.

Кастомный выход из приложения

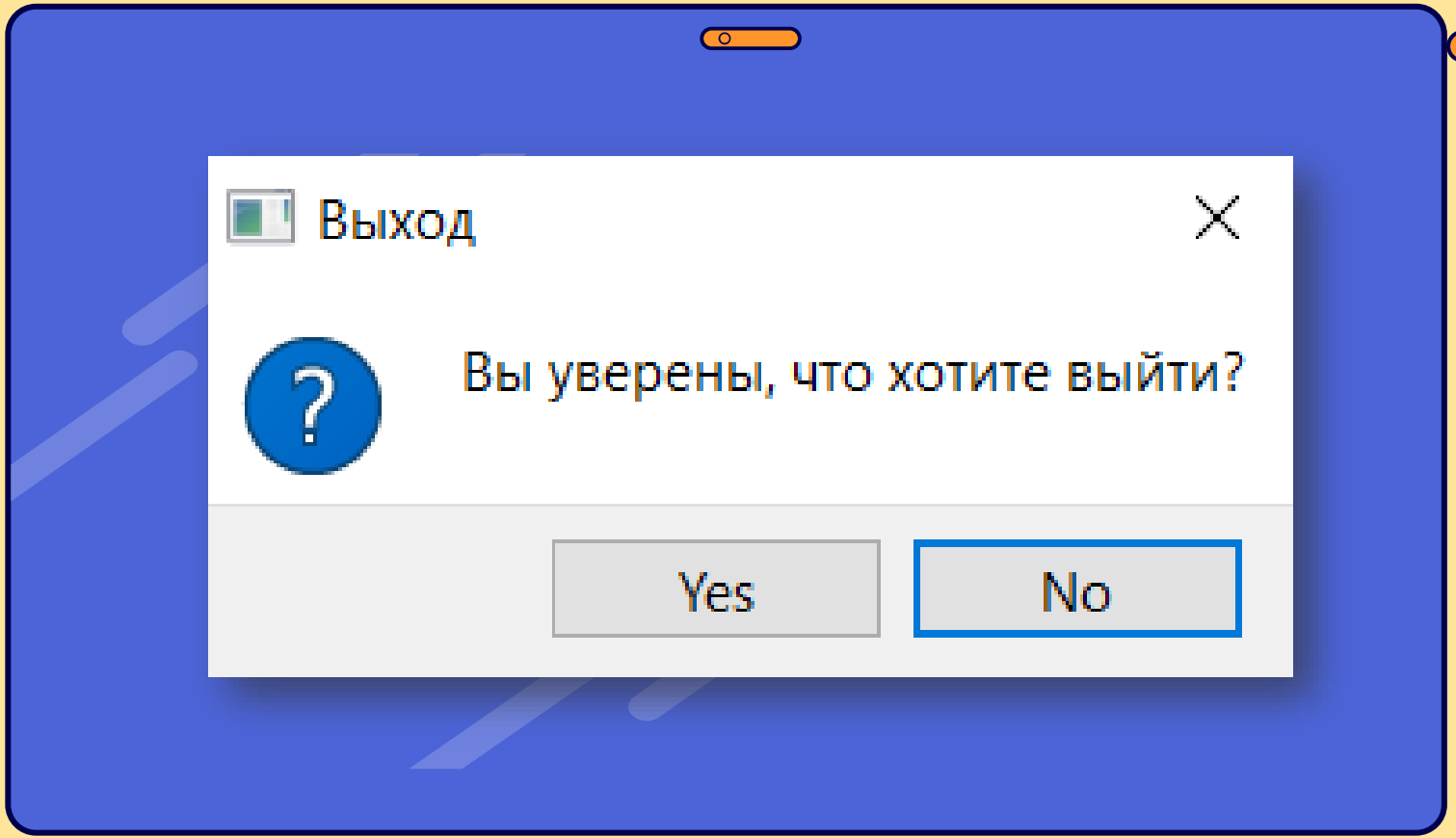
Переписываем существующий closeEvent

```
def closeEvent(self, event): # Создаем кастомное событие выхода. Пользователю будет задан вопрос.
    reply = QMessageBox.question(self, 'Выход',
                                 "Вы уверены, что хотите выйти?", QMessageBox.Yes |
                                 QMessageBox.No, QMessageBox.No)
    if reply == QMessageBox.Yes: # Если юзер отвечает "да" -
        event.accept() # то мы выходим.
    else: # Если юзер отвечает "нет" -
        event.ignore() # ничего не происходит.
```

Вставляем все это дело как отдельный метод в классе (после initUI)



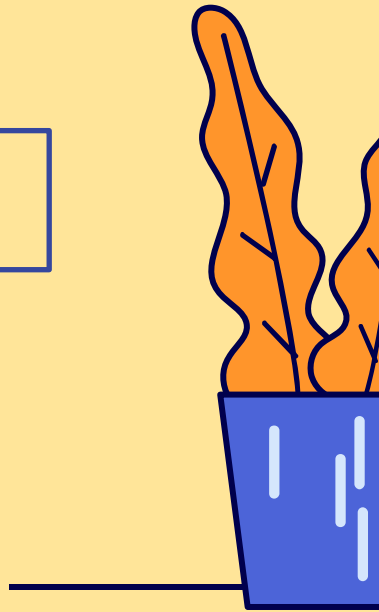
Output:



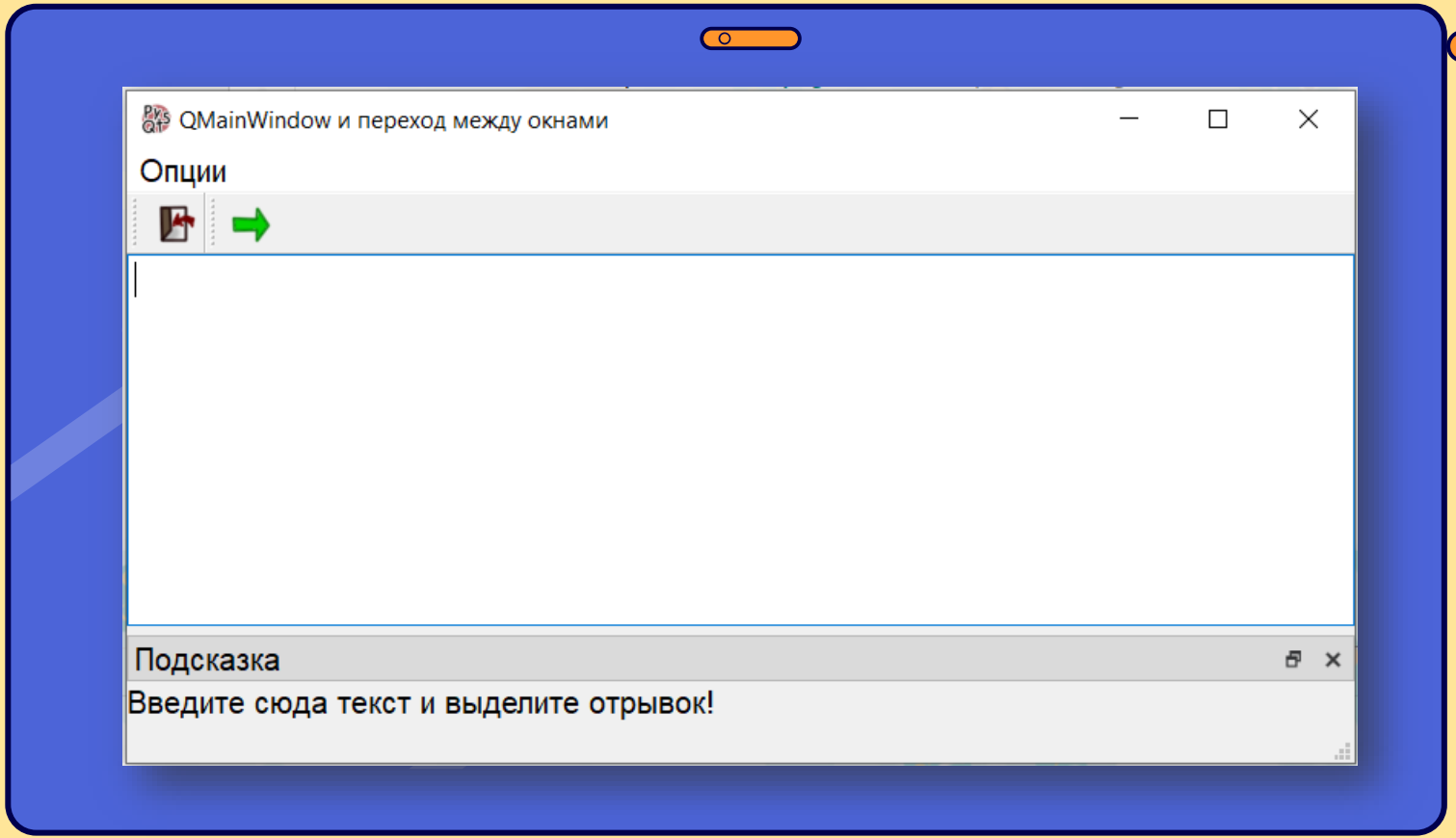
Добавляем действие «Продолжить» в QMainWindow

```
def initUI(self): # Основная функция для построения нашего окна.

    self.textedit = QPlainTextEdit() # Создаем простейший редактор текста.
    self.setCentralWidget(self.textedit) # Назначаем редактор текста главным виджетом нашего окна QMainWindow.
    labelwidget = QDockWidget(self)  # Создаем экземляр док-виджета. Это перемещаемый объект, уникальный для MainWindow.
    labelwidget.setWindowTitle("Подсказка")  # Даем название нашему док-виджету.
    label = QLabel("Введите сюда текст!")  # Создаем нашу надпись.
    labelwidget.setWidget(label)  # Устанавливаем наш готовый виджет в экземпляр док-виджета.
    self.addDockWidget(Qt.BottomDockWidgetArea, labelwidget)  # Добавляем в окно наш док-виджет и выбираем нижнюю область для него.

    exitAction = QAction(QIcon('exit.png'), 'Выход', self) # Создаем действие "выход". Выбираем для него иконку и название.
    exitAction.setShortcut('Ctrl+Q') # Выбираем горячие клавиши для выхода.
    exitAction.setStatusTip('Выход из приложения') # Создаем подсказку к нашему действию.
    exitAction.triggered.connect(self.close) # Привязываем функцию к действию. В данном случае это уже существуюший метод .close.

    continueAction = QAction(QIcon('continue.png'), 'Продолжить', self) # Делаем аналогичную работу с действием "продолжить".
    continueAction.setShortcut('Ctrl+W')
    continueAction.setStatusTip('Открыть следующее окно')
    continueAction.triggered.connect(self.next_window)

    self.statusBar() # Добавляем в окно строку состояния.
    menubar = self.menuBar() # Добавляем в окно верхнее меню.

    optionMenu = menubar.addMenu('&Опции') # Содаем категорию "опции" в меню.
    optionMenu.addAction(exitAction) # Добавляем в категорию "опции" наше действие.
    optionMenu.addAction(continueAction)

    toolbar = self.addToolBar('Выход') # Добавляем в поле с инструментами кнопку "Выход"...
    toolbar.addAction(exitAction) # ...И действие к нему.
    toolbar2 = self.addToolBar('Продолжить') # То же самое делаем с "Продолжить".
    toolbar2.addAction(continueAction)
```

Иконка 



Output:


QMainWindow и переход между окнами
Опции
Подсказка
Введите сюда текст и выделите отрывок!

Добавляем метод «Продолжить» в Window1

```
def next_window(self): # Метод для открытия следующего окна.
    cursor = self.textedit.textCursor() # Создаем курсор, который вернет нам выделенный текст.
    text = cursor.selectedText() # Получаем выделенный текст из QTextEdit.
    self.w = Window2(text) # Создаем следующее окно и передаем аргумент,
    self.w.show() # показываем его,
    self.hide() # и скрываем это окно.
```

Все это вставляется как отдельный метод класса нашего окошка (после initUI).

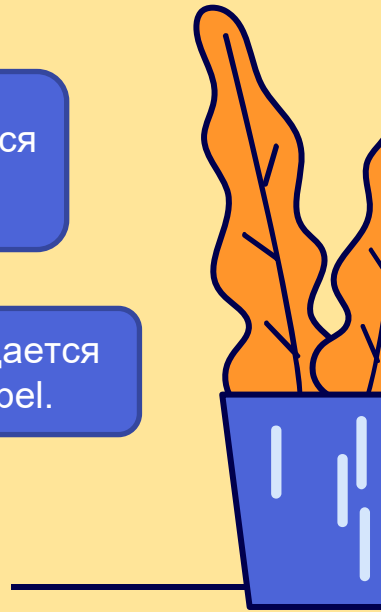
Построение второго окна

```
class Window2(QWidget):  # Создаем класс от родителя QWidget - это будет наше второе окно.
    def __init__(self, txt):
        super().__init__()  # Наследуем методы и атрибуты класса QWidget.

        self.setWindowTitle("QMainWindow и переход между окнами")  # Называем наше окно.
        self.resize(500, 200)  # Назначаем размер окна.
        self.setWindowIcon(QIcon('icon.png'))  # Выбираем иконку для нашего окна.
        self.setFont(QFont('Arial', 20))  # Назначаем шрифт и размер шрифта.
        self.w = None  # В эту переменную мы поместим наше предыдущее окно.
        self.text = txt # Кладем полученный текст в атрибут класса.

        self.initUI()

    def initUI(self):  # Основная функция для построения нашего окна.

        label1 = QLabel("Вы выделили такой текст:")  # Создаем нашу надпись.
        label1.setAlignment(Qt.AlignCenter)  # Выравниваем по центру.
        label2 = QLabel("") # Создаем надпись, которую потом заполним текстом.
        label2.setText(self.text) # Устанавливаем полученный текст.
        button = QPushButton("Назад") # Создаем кнопку "Назад".

        layout = QVBoxLayout()  # Вертикальная раскладка, в которую мы будем класть наши виджеты.
        layout.addWidget(label1)  # Добавляем надпись.
        layout.addWidget(label2)
        layout.addWidget(button)
        self.setLayout(layout)  # Устанавливаем нашу раскладку в окно.

        button.clicked.connect(self.previousWindow) # Привязываем кнопку к функции «Назад».
```

Помимо self., наше окно принимает еще один аргумент – выделенный текст.

Этот текст затем кладется в атрибут класса…

И помещается в QLabel.

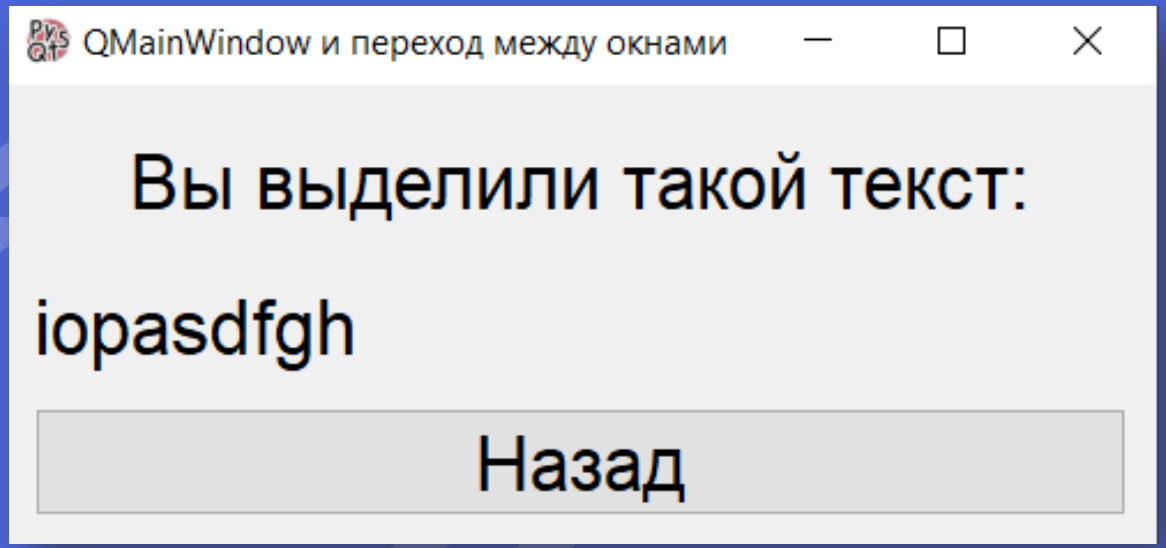


Добавляем метод «Назад» в Window2

```
def previousWindow(self):
    self.w = Window1() # Создаем экземпляр первого окна,
    self.w.show() # Показываем его,
    self.hide() # И скрываем это.
```

Все это вставляется как отдельный метод класса нашего окошка (после initUI).

Output:


QMainWindow и переход между окнами
Вы выделили такой текст:
iopasdfgh
Назад

Немного священных знаний о PyQt5

Можно перехватить мистическую ошибку интерпретатора. Так как мы не знаем, какая именно ошибка возникла, придется все делать через исключение BaseExeption.

```
try:
        # Что-нибудь сделать
except BaseException:
        return QMessageBox.warning(self, "Ошибка", "Не удалось обработать запрос")
```

Таким образом, при возникновении ошибки интерпретатора программа не «вылетает», а лишь выдает ошибку и перестает выполнять вызванную функцию, возвращая нас к окну.

Немного священных знаний о PyQt5

Можно заставить PyQt5 читать эксельки и выводить таблицы из них. Для этого надо переписать класс QAbstractTableModel, который сможет обрабатывать DataFrame. Как читать эксельки вы, наверное, знаете.

```
class TableModel(QAbstractTableModel):

    def __init__(self, data):
        super(TableModel, self).__init__()
        self._data = data

    def data(self, index, role):
        if role == Qt.DisplayRole:
            value = self._data.iloc[index.row(), index.column()]
            return str(value)

    def rowCount(self, index):
        return self._data.shape[0]

    def columnCount(self, index):
        return self._data.shape[1]

    def headerData(self, section, orientation, role):
        if role == Qt.DisplayRole:
            if orientation == Qt.Horizontal:
                return str(self._data.columns[section])

            if orientation == Qt.Vertical:
                return str(self._data.index[section])
```

Переписать класс

Сделать табличку из датафрейма

```
from PyQt5.QtWidgets import QTableView

tableview = QTableView()

model = TableModel(df)
tableview.setModel(model)
```

Немного священных знаний о PyQt5

В PyQt5 можно показывать графики из Matplotlib, Seaborn или Pandas.
Для этого нужно:

```
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas,
NavigationToolbar2QT as NavigationToolbar
```

Импортировать необходимые вещи из модуля бэкенда matplotlib

```
fig, ax = plt.subplots()

## Ваш код по настройке графика


canvas = FigureCanvas(fig)
toolbar = NavigationToolbar(canvas, self)
```

Настроить график и установить его в FigureCanvas. После этого он станет виджетом

Немного священных знаний о PyQt5

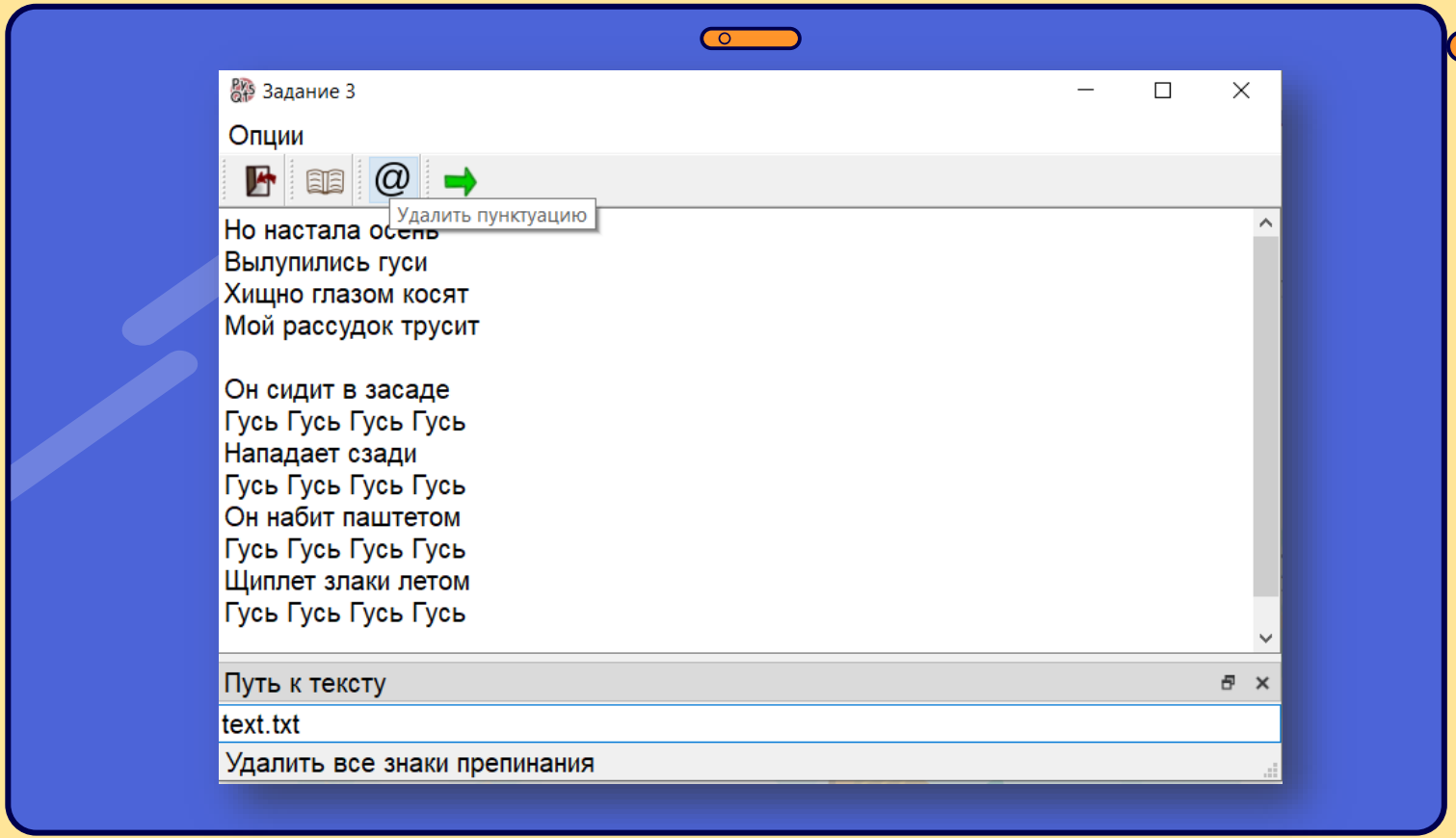
Чтобы компилировать Ваш код Python (в принципе любой) в файл .exe (на Windows), надо:

1. Скачать PyInstaller: pip install pyinstaller
2. Открыть командную строку.
3. Перейти в папку с кодом, который Вы хотите скомпилировать, через cd. Пример:
cd /D d:\Desktop\Folder\MyScript.py
4. Ввести команду:
PyInstaller --onefile --icon=icon.ico --noconsole MyScript.py
5. Дождаться завершения компиляции и найти свой дистрибутив в папке dist (папку build можно удалить).
6. Теперь Ваша программа будет открываться на любом компьютере с операционной системой Windows!
7. Для того, чтобы поделиться программой, можно создать архив, содержащий только дистрибутив .exe и папки с дополнительными материалами (файлами, картинками и т.д.).

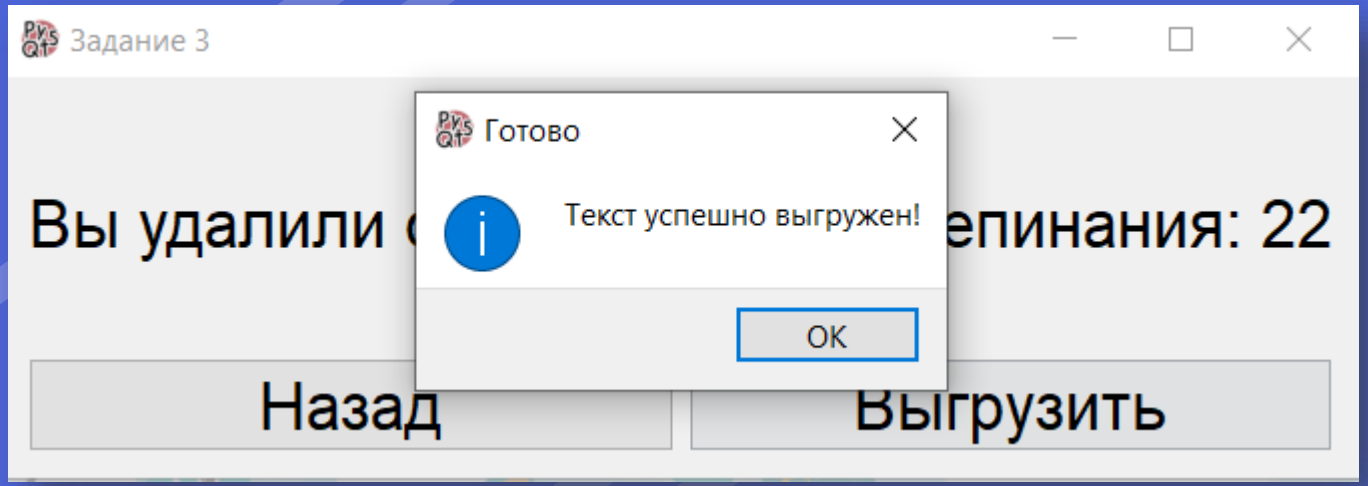
Сложное задание!

Принимаются ваши собственные задумки с использованием двух окон.
Если идей нет, то попробуйте выполнить следующее:
1. Создайте QMainWindow с QTextEdit как главным виджетом.
2. Создайте текстовый файл, поместив в него какой-нибудь текст.
3. Добавьте в QMainWindow QDockWidget и положите в него QLineEdit, который будет служить для того, чтобы юзер ввел туда путь к текстовому файлу.
4. Создайте действие, привязав к нему созданный Вами метод: пусть при выборе этого действия в QTextEdit открывается файл по пути, введенному в QLineEdit.
5. Создайте действие, которое бы удалило из полученного текста все знаки препинания. Запомните количество удаленных знаков.
6. Создайте действие «Продолжить» и передайте следующему окну (QWidget) готовый текст и количество удаленных знаков препинания.
7. В новом окне: пусть надпись QLabel скажет, сколько знаков препинания было удалено, а кнопка QPushButton позволит выгрузить полученный текст в файл.
8. В функции выгрузки файла вызовите QMessageBox.information, который объявит об окончании выгрузки.
8. Создайте кнопку «Назад» и подключите ее к функции возврата к первому окну.

Как примерно это будет выглядеть:



Как примерно это будет выглядеть:



Спасибо за внимание!

Присылайте домашку:
lorelei.aether@gmail.com
Telegram: @lorelei_ether