QInputDialog

QInputDialog обеспечивает простой удобный диалог для получения единственного значения от пользователя. Введённое значение может быть строкой, числом или пунктом из списка.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
from PyQt5.QtWidgets import (QWidget, QPushButton, QLineEdit, QInputDialog,
QApplication)

class Example(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()


    def initUI(self):
        self.btn = QPushButton('Dialog', self)
        self.btn.move(20, 20)
        self.btn.clicked.connect(self.showDialog)

        self.le = QLineEdit(self)
        self.le.move(130, 22)

        self.setGeometry(300, 300, 290, 150)
        self.setWindowTitle('Input dialog')
        self.show()


    def showDialog(self):
        text, ok = QInputDialog.getText(self, 'Input Dialog', 'Enter your
name:')

        if ok:
            self.le.setText(str(text))


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Пример имеет кнопку и виджет редактирования строки. Кнопка показывает диалог ввода для получения текстовых значений. Вводимый текст может быть отображён в виджете редактирования строки.

```
text, ok = QInputDialog.getText(self, 'Input Dialog', 'Enter your name:')
```

Эта строка показывает диалог ввода. Первая строка – это заголовок диалога, вторая – сообщение внутри диалога. Диалог возвращает введённый текст и логическое значение. Если мы кликаем кнопку «OK», то логическое значение является правдой.

```
if ok:
    self.le.setText(str(text))
```

Текст, который мы получили из диалога, устанавливается в виджет редактирования строки.

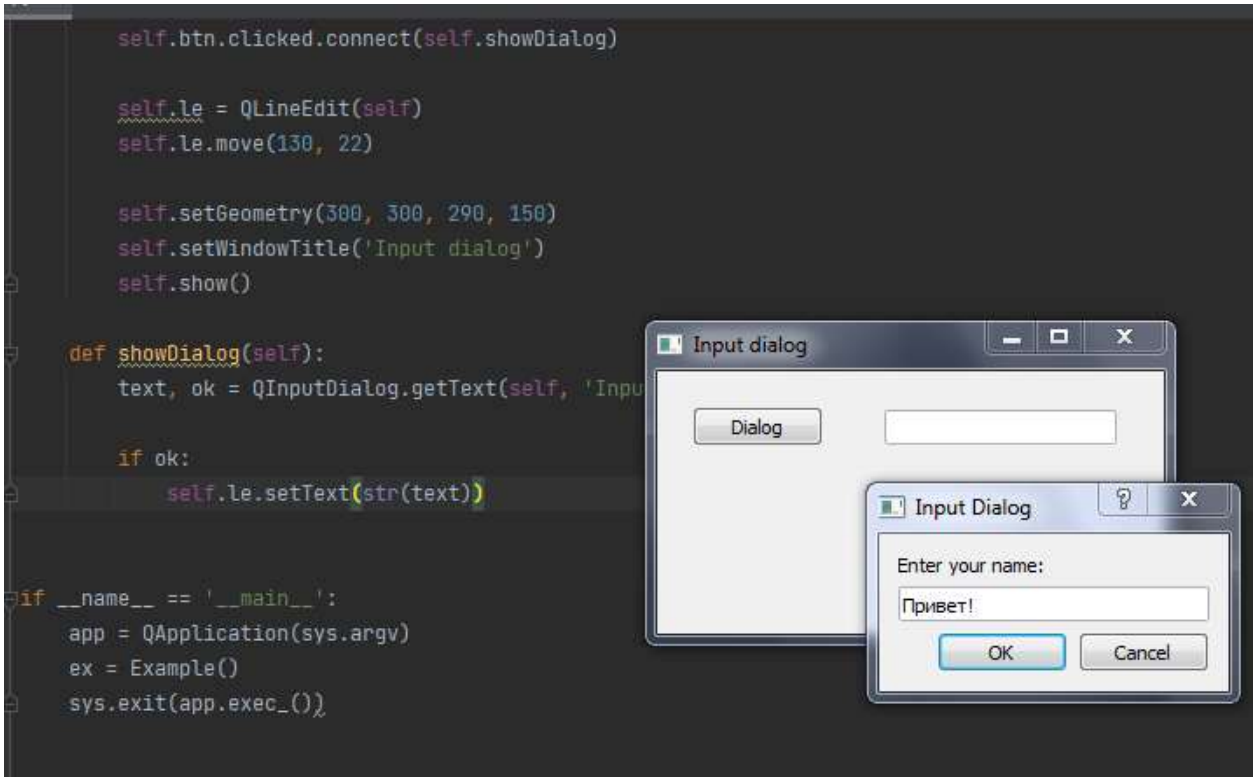


Рисунок: Ввод диалога

QColorDialog

QColorDialog обеспечивает виджет диалога для выбора цветовых значений.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import sys
from PyQt5.QtWidgets import (QWidget, QPushButton, QFrame, QColorDialog,
QApplication)
from PyQt5.QtGui import QColor


class Example(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        col = QColor(0, 0, 0)
        self.btn = QPushButton('Dialog', self)
```

```
        self.btn.move(20, 20)
        self.btn.clicked.connect(self.showDialog)
        self.frm = QFrame(self)
        self.frm.setStyleSheet("QWidget { background-color: %s }" %
col.name())
        self.frm.setGeometry(130, 22, 100, 100)

        self.setGeometry(300, 300, 250, 180)
        self.setWindowTitle('Color dialog')
        self.show()



    def showDialog(self):
        col = QColorDialog.getColor()
        if col.isValid():
            self.frm.setStyleSheet("QWidget { background-color: %s }" %
col.name())

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Пример приложения показывает кнопку и **QFrame**. Фон виджета устанавливается чёрным цветом. Используя QColorDialog, мы можем менять фон.

```
col = QColor(0, 0, 0)
```

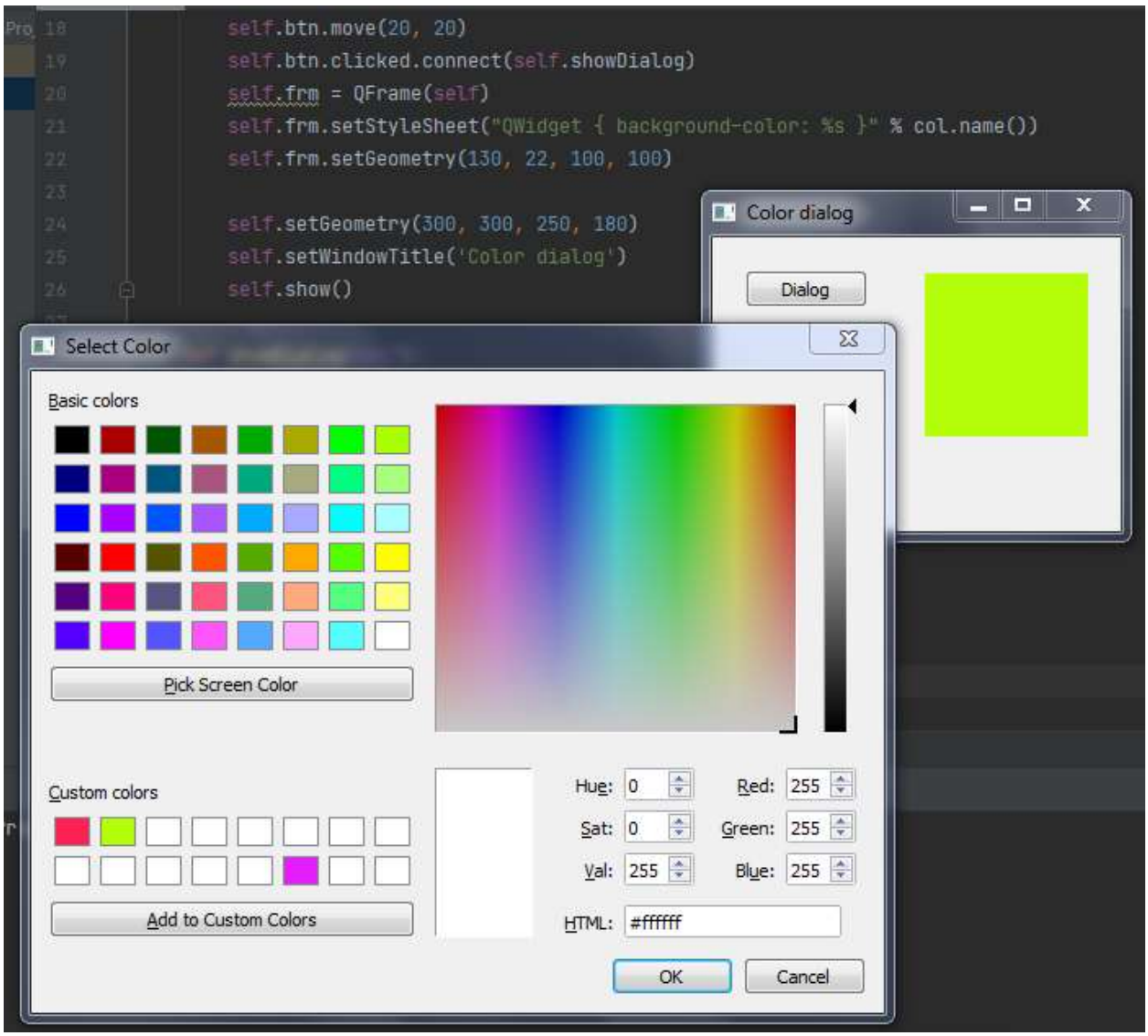
Это первоначальный цвет фона **QtGui.QFrame**.

```
col = QColorDialog.getColor()
```

Это строка высветит **QColorDialog**.

```
if col.isValid():
    self.frm.setStyleSheet("QWidget { background-color: %s }" % col.name())
```

Мы проверяем, является ли цвет валидным. Если мы нажимаем кнопку «Cancel», возвращается невалидный цвет. Если цвет валиден, мы меняем цвет фона, используя таблицы стилей (CSS).



QFontDialog

QFontDialog – это виджет диалога для выбора шрифта.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import sys
from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QPushButton, QSizePolicy,
QLabel, QFontDialog, QApplication)

class Example(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        vbox = QVBoxLayout()

        btn = QPushButton('Dialog', self)
```

```
        btn.setSizePolicy(QSizePolicy.Fixed, QSizePolicy.Fixed)

        btn.move(20, 20)
        vbox.addWidget(btn)
        btn.clicked.connect(self.showDialog)

        self.lbl = QLabel('Knowledge only matters', self)
        self.lbl.move(130, 20)
        vbox.addWidget(self.lbl)
        self.setLayout(vbox)

        self.setGeometry(300, 300, 250, 180)
        self.setWindowTitle('Font dialog')
        self.show()

    def showDialog(self):
        font, ok = QFontDialog.getFont()
        if ok:
            self.lbl.setFont(font)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

В нашем примере, мы имеем кнопку и метку. С помощью QFontDialog, мы меняем шрифт метки.

```
font, ok = QFontDialog.getFont()
```

Здесь мы высвечиваем диалог со шрифтами. Метод **getFont**() возвращает имя шрифта и параметр ok. Он равен «Истине», если пользователь кликнул «OK»; в противном случае, параметр – «Ложь».

```
if ok:
    self.label.setFont(font)
```

Если бы мы кликнули «OK», шрифт метки б изменился.

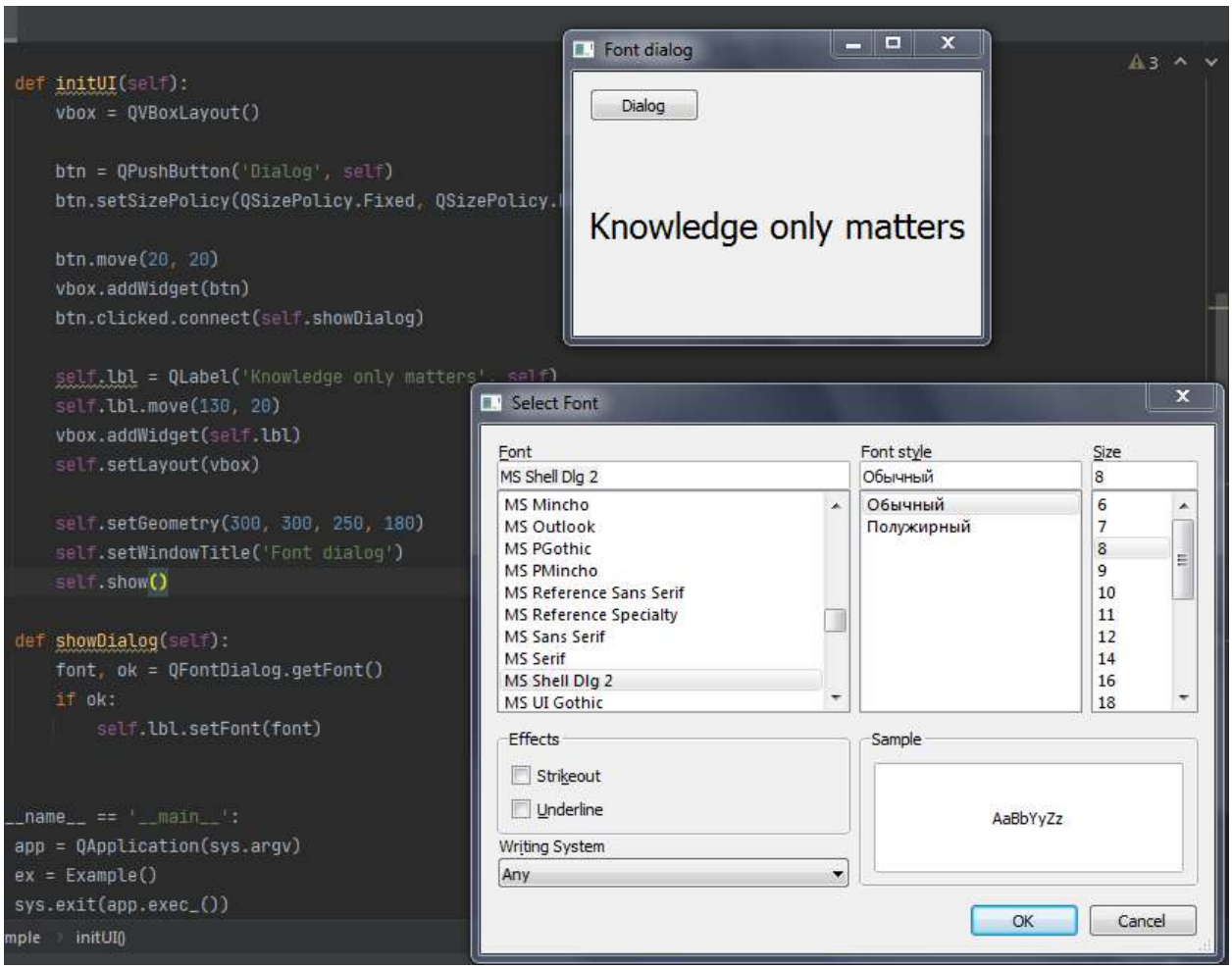


Рисунок: Диалог со шрифтами

QFileDialog

QFileDialog – это диалог, который позволяет пользователям выбирать файлы или папки. Файлы могут быть выбраны и для открытия, и для сохранения.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import sys
from PyQt5.QtWidgets import (QMainWindow, QTextEdit, QAction, QFileDialog,
QApplication)
from PyQt5.QtGui import QIcon

class Example(QMainWindow):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.textEdit = QTextEdit()
        self.setCentralWidget(self.textEdit)
        self.statusBar()
```

```
        openFile = QAction(QIcon('open.png'), 'Open', self)
        openFile.setShortcut('Ctrl+O')
        openFile.setStatusTip('Open new File')
        openFile.triggered.connect(self.showDialog)

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(openFile)

        self.setGeometry(300, 300, 350, 300)
        self.setWindowTitle('File dialog')
        self.show()

    def showDialog(self):
        fname = QFileDialog.getOpenFileName(self, 'Open file', '/home')

        f = open(fname, 'r')

        with f:
            data = f.read()
            self.textEdit.setText(data)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Пример показывает строку меню, центрально-установленный виджет редактирования текста и строку состояния. Пункт меню показывает **QtGui.QFileDialog**, который используется для выбора файла. Содержание файла загружается в виджет редактирования текста.

```
class Example(QMainWindow):

    def __init__(self):
        super().__init__()
        self.initUI()
```

Пример основывается на виджете **QMainWindow**, поскольку мы установили по центру виджет редактирования текста.

```
fname = QFileDialog.getOpenFileName(self, 'Open file', '/home')
```

Мы высвечиваем QFileDialog. Первая строка в методе getOpenFileName() – это заголовок. Вторая строка указывает диалог рабочей папки. По умолчанию, файловый фильтр установлен в положение «Все файлы (*)».

```
f = open(fname, 'r')
with f:
    data = f.read()
    self.textEdit.setText(data)
```

Выбранное имя файла читается и содержание файла устанавливается в виджет редактирования текста.

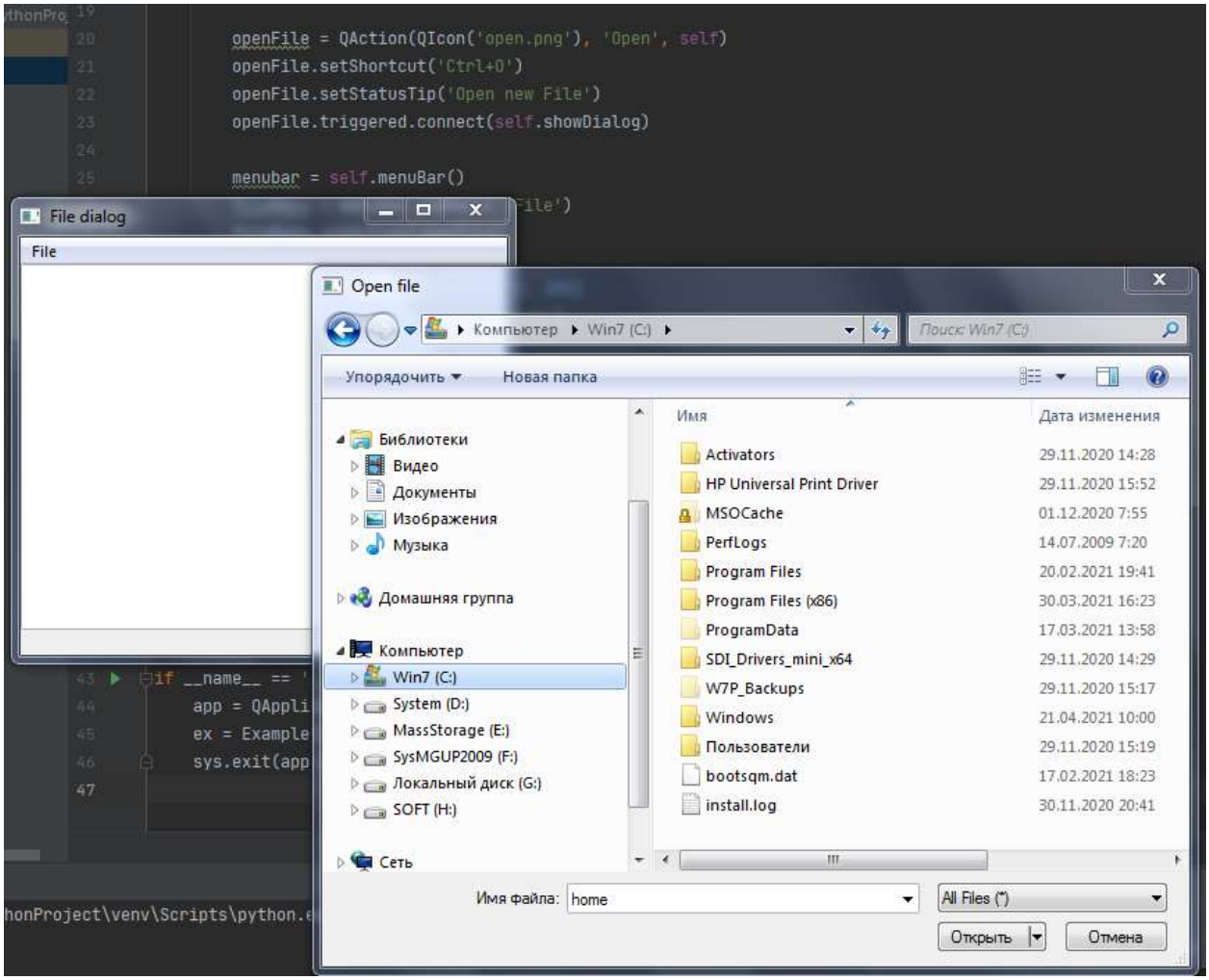


Рисунок: Файловый диалог

В последнем примере этой части, мы создадим меню, панель инструментов и строку состояния. Мы также создадим центральный виджет.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import sys
from PyQt5.QtWidgets import QMainWindow, QTextEdit, QAction, QApplication
from PyQt5.QtGui import QIcon
class Example(QMainWindow):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        textEdit = QTextEdit()
        self.setCentralWidget(textEdit)
        exitAction = QAction(QIcon('exit24.png'), 'Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(self.close)
        self.statusBar()
        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(exitAction)
        toolbar = self.addToolBar('Exit')
        toolbar.addAction(exitAction)

        self.setGeometry(300, 300, 350, 250)
        self.setWindowTitle('Main window')
        self.show()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

Этот пример кода создаёт каркас классического графического приложения состоящей из меню, панелью инструментов и строкой состояния.

```
textEdit = QTextEdit()
self.setCentralWidget(textEdit)
```

Здесь мы создаём виджет редактирования текста. Мы устанавливаем его так, чтобы он был центральным виджетом QMainWindow. Центральный виджет займёт всё оставшееся пространство.

Строку состояния создают с помощью виджета **QMainWindow**.

```
self.statusBar().showMessage('Ready')
```

Чтобы получить строку состояния, мы вызываем метод **statusBar**() класса QtGui.QMainWindow. Первый вызов метода создаёт строку состояния. Последующие вызовы возвращают объект строки состояния. **showMessage**() отображает сообщение в строке состояния.

В примере мы создаём строку меню с одним набором команд. Этот набор команд будет содержать одно действие, завершающее приложение при его выборе. Строка состояния тоже создаётся. Действие доступно с горячей клавишей Ctrl+Q.

```
exitAction = QAction(QIcon('exit.png'), '&Exit', self)
exitAction.setShortcut('Ctrl+Q')
exitAction.setStatusTip('Exit application')
```

QAction – это абстракция для действий, выполняемых из меню, панелью инструментов или с использованием горячей клавишей. В приведённых выше трёх строках, мы создаём действие с определённой иконкой и меткой «Exit». Кроме того, для этого действия определена горячая клавиша. Третья строка создаёт подсказку, которая показывается в строке состояния, когда мы наводим курсор мыши на пункт меню.

```
exitAction.triggered.connect(qApp.quit)
```

Когда мы выбираем конкретное действие, срабатывает инициирующий сигнал. Сигнал присоединяют к методу quit() виджета QApplication. Это завершает приложение.

```
menubar = self.menuBar()
fileMenu = menubar.addMenu('&File')
fileMenu.addAction(exitAction)
```

Метод **menuBar**() создаёт строку меню. Мы создаём меню «File» и добавляем в него действие вых .

В вышеприведённом примере, мы создаём простую панель инструментов. Она имеет один инструмент, действие выхода, которое завершает приложение, будучи инициированным.

```
exitAction = QAction(QIcon('exit24.png'), 'Exit', self)
exitAction.setShortcut('Ctrl+Q')
exitAction.triggered.connect(qApp.quit)
```

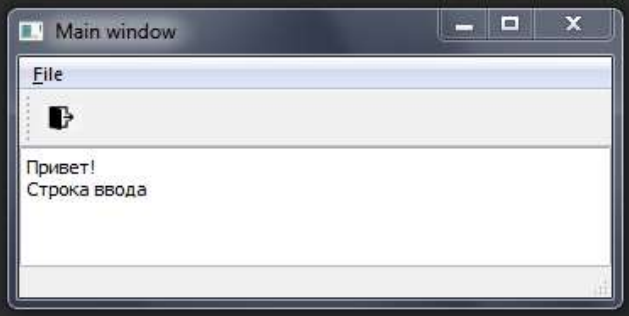
В аналогичном примере как с созданием меню выше, мы создаём объект действия. Объект имеет метку, иконку и горячую клавишу. Метод quit() из **QtGui.QMainWindow** присоединяется к инициирующему сигналу.

```
self.toolbar = self.addToolBar('Exit')
self.toolbar.addAction(exitAction)
```

Здесь мы создаём панель инструментов и подключаем объект действия к ней.

Не забудьте разместить иконку exit24.png в папке рядом с вашей программой на Python.

```
def initUI(self):

    textEdit = QTextEdit()
    self.setCentralWidget(textEdit)

    exitAction = QAction(QIcon('exit24.png'), 'Exit', self)
    exitAction.setShortcut('Ctrl+Q')
    exitAction.setStatusTip('Exit application')
    exitAction.triggered.connect(self.close)

    self.statusBar()

    menubar = self.menuBar()
    fileMenu = menubar.addMenu('&File')
    fileMenu.addAction(exitAction)

    toolbar = self.addToolBar('Exit')
    toolbar.addAction(exitAction)

    self.setGeometry(300, 300, 350, 250)
    self.setWindowTitle('Main window')
    self.show()
```



Задание.

Создать полноценное оконное приложение со всеми изученными меню, окнами, диалогами и прочими элементами оконного интерфейса.

Программа должна вызывать и выполнять некоторые ранее сделанные в консоли лабораторные работы (не менее 4).