

# PyQt6

...

## <desktop app>

**Desktop app** - это приложение, предназначенное для установки и использования на компьютере пользователя, обычно запускаемое с рабочего стола.

**Tkinter** - библиотека Python для создания графических интерфейсов.

**PyQt6** - библиотека Python для создания GUI, основанная на фреймворке Qt.

# Что такое ПО, GUI?

**Программное обеспечение (ПО)** — это совокупность программных инструкций и данных, которые управляют работой компьютера или другого устройства. Оно предназначено для выполнения определенных функций или задач на компьютере пользователя или другом устройстве.

## Характеристики:

- **Функциональность:** ПО может выполнять различные функции, от обработки данных и управления ресурсами до предоставления пользовательского интерфейса для взаимодействия с пользователем.
- **Типы ПО:** Включают операционные системы, прикладное программное обеспечение (например, текстовые редакторы, игры, браузеры), устройственное программное обеспечение (драйверы), и т.д.
- **Распространение:** ПО может быть распространяемым как коммерческим, так и свободным (open-source).

**Графический пользовательский интерфейс (GUI)** — это способ взаимодействия пользователя с компьютерной программой, использующий графические элементы (виджеты) и манипуляцию с ними с помощью мыши и клавиатуры.

# import tkinter

```
import tkinter as tk
```

Создание экземпляра основного окна (главного контейнера) с помощью класса Tk() из Tkinter.

```
root = tk.Tk()
```

```
root.title("Привет, мир!")
```

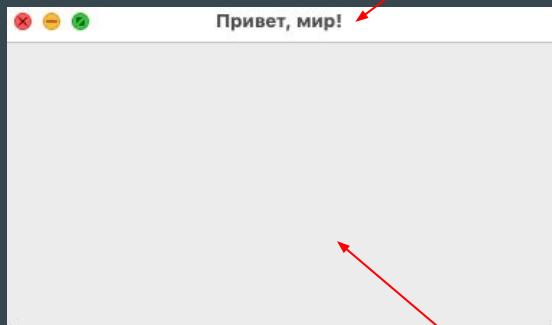
Установка заголовка (названия) окна на "Привет, мир!".

```
root.mainloop()
```

Запуск главного цикла обработки событий, который ожидает действий пользователя (например, нажатие кнопок) и отображает интерфейс окна до его закрытия.

# что получили при запуске?

наш заголовок



пусто так как ничего не  
добавлено пока еще...

# tkinter base elements

- **Текст:** `tk.Label(root, text="Текст")`
- **Кнопка:** `tk.Button(root, text="Кнопка",  
command=function_name)`
- **Поле ввода:** `tk.Entry(root)`
- **Флажок (чекбокс):** `tk.Checkbutton(root, ...)`
- **Переключатель (радиокнопка):** `tk.Radiobutton(root, ...)`
- **Список (выпадающий список):** `tk.OptionMenu(root, var, *options)`
- **Прокрутка (скроллер):** `tk.Scrollbar(root)`
- **Фрейм (панель):** `tk.Frame(root)`
- **Изображение:** `tk.PhotoImage(file="image.gif")`
- **Меню:** `tk.Menu(root)`

# tkinter-example#1

```
import tkinter as tk
```

```
root = tk.Tk()
```

```
root.title("Программа")
```

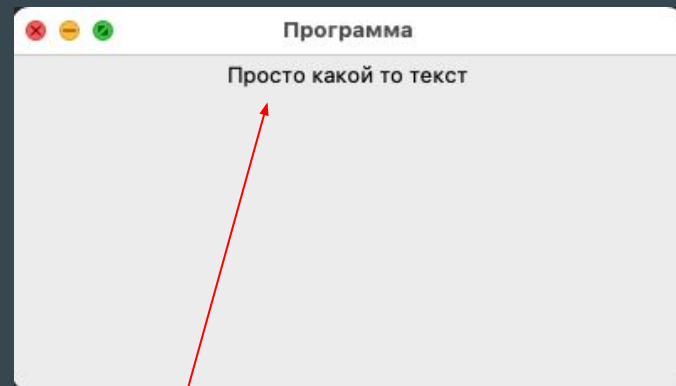
```
root.geometry("400x200")
```

*задает размер окна  
приложения...*

```
label = tk.Label(root, text="Просто какой то текст")
```

```
label.pack()
```

```
root.mainloop()
```



*наш собственно Label который  
мы и создали.*

# tkinter-example#2

```
import tkinter as tk
```

```
root = tk.Tk()
```

```
def foo():
```

```
    label = tk.Label(root, text="Просто какой то текст")
```

```
    label.pack()
```

```
root.title("Программа")
```

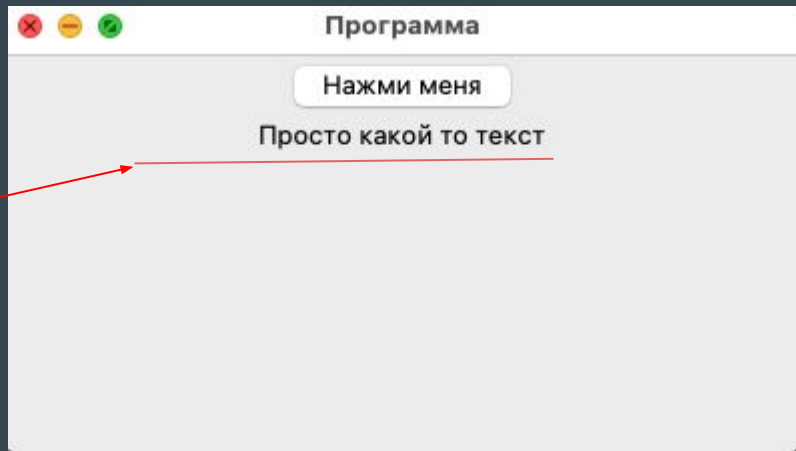
```
root.geometry("400x200")
```

```
button = tk.Button(root, text="Нажми меня", command=foo)
```

```
button.pack()
```

```
root.mainloop()
```

при клике  
появился  
текст



при клике вызовется  
эта функция.



# tkinter-example#3; 1 часть кода

```
import tkinter as tk

def check_credentials():
    username = username_entry.get()
    password = password_entry.get()

    if username == "admin" and password == "password":
        result_label.config(text="Вход выполнен успешно")
    else:
        result_label.config(text="Неверные учетные данные")

root = tk.Tk()
root.title("Проверка учетных данных")
...
```

# tkinter-example#3; 2 часть кода

...

```
username_label = tk.Label(root, text="Имя пользователя:")
username_label.pack()

username_entry = tk.Entry(root)
username_entry.pack()

password_label = tk.Label(root, text="Пароль:")
password_label.pack()

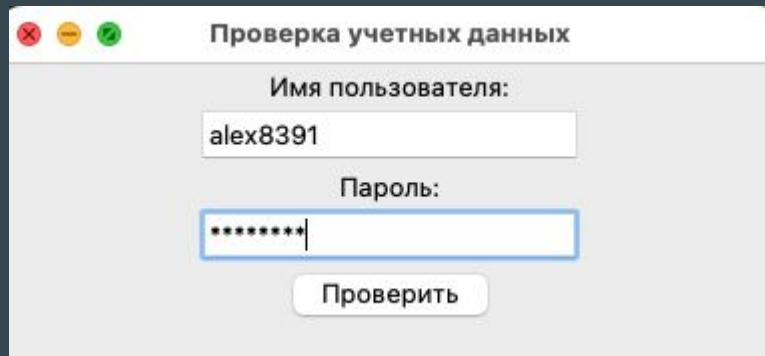
password_entry = tk.Entry(root, show="*") # show="*" скрывает вводимые символы
password_entry.pack()

check_button = tk.Button(root, text="Проверить", command=check_credentials)
check_button.pack()

result_label = tk.Label(root, text="")
result_label.pack()

root.mainloop()
```

# tkinter-example#3; работоспособность кода

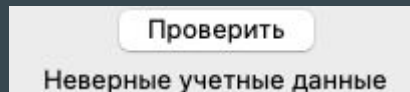


Проверка учетных данных

Имя пользователя:  
alex8391

Пароль:  
\*\*\*\*\*

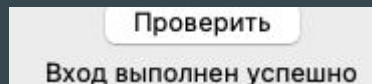
Проверить



Проверить

Неверные учетные данные

*если данные введены неверно*



Проверить

Вход выполнен успешно

*если данные введены верно*

# tkinter vs pyqt

## 1. Мощность и гибкость:

- **Tkinter:** Простая и легкая в освоении библиотека для создания базовых GUI приложений.
- **PyQt:** Предоставляет более широкие возможности для создания сложных и профессиональных пользовательских интерфейсов с помощью Qt.

## 2. Функциональность:

- **Tkinter:** Ограниченный набор виджетов и функций, но достаточен для базовых приложений.
- **PyQt:** Обширный набор инструментов, включая анимации, графику, стилизацию и множество других возможностей.

## 3. Кроссплатформенность:

- **Tkinter:** Поддерживается на всех основных операционных системах, но может иметь различия в отображении и поведении интерфейса.
- **PyQt:** Обеспечивает высокую кроссплатформенность и консистентность интерфейса между различными ОС.

## 4. Интеграция с Qt Designer:

- **Tkinter:** Не имеет визуального конструктора, все интерфейсы создаются программно.
- **PyQt:** Имеет интеграцию с Qt Designer, позволяющий создавать интерфейсы методом "перетаски и отпусти" без необходимости написания кода.

## 5. Язык программирования:

- **Tkinter:** Использует только Python.
- **PyQt:** Поддерживает как Python, так и C++.

# PyQt

**PyQt** — это набор связующих (bindings) для языка программирования Python, который позволяет использовать библиотеку Qt, написанную на C++, для создания графических пользовательских интерфейсов (GUI). PyQt включает в себя модули для работы с виджетами, событиями, сетями, базами данных и другими функциями Qt.

## Сферы применения:

1. **Разработка ПО с GUI:**
  - PyQt широко используется для создания интерактивных и мощных пользовательских интерфейсов на Python.
  - Применяется в различных областях, от настольных приложений до инструментов администрирования и управления данными.
2. **Научные вычисления и визуализация данных:**
  - PyQt в сочетании с библиотеками для научных вычислений (например, NumPy, SciPy) используется для создания приложений для анализа данных и визуализации.
3. **Инструменты автоматизации и управления процессами:**
  - PyQt позволяет разрабатывать инструменты для автоматизации задач, управления процессами и взаимодействия с пользователем.
4. **Игровая индустрия:**
  - PyQt может использоваться для создания административных и конфигурационных интерфейсов игровых приложений.
5. **Образовательные и тренировочные приложения:**
  - PyQt подходит для создания образовательных и тренировочных приложений с удобным интерфейсом для пользователей.

# Версии и отличия (pyqt)

1. **PyQt4:**
  - Поддерживает Qt 4.x.
  - PyQt4 был популярным до выпуска Qt 5 и PyQt5.
  - Устарел и больше не поддерживается.
2. **PyQt5:**
  - Поддерживает Qt 5.x.
  - Современная версия PyQt, активно поддерживаемая и развиваемая.
  - Предоставляет полный доступ к функциональности Qt 5, включая Qt Widgets, QML и другие модули.
  - Поддерживает Python 2.7 и Python 3.x.
3. **PyQt6:**
  - Поддерживает Qt 6.x.
  - PyQt6 разрабатывается с учетом новых функций и изменений в Qt 6.
  - Включает обновленный API и улучшенные возможности для разработчиков.
  - Поддерживает Python 3.x (Python 2.7 более не поддерживается).

## Отличия:

- **Совместимость с Qt:** Каждая версия PyQt соответствует конкретной версии Qt.
- **Функциональность:** Новые версии обычно включают расширенные возможности и улучшения производительности.
- **Поддержка:** PyQt5 и PyQt6 активно поддерживаются и обновляются, в то время как PyQt4 устарел и не рекомендуется для новых проектов.

# create pyqt on tkinter

Базовый класс для графического интерфейса.

QtGui

Основной объект приложения

QApplication

```
sys.exit(app.exec())
```

QWidget(QtGui)

Базовый класс для всех виджетов (окон)

QMainWindow (QWidget)

Главное (основное) окно приложения

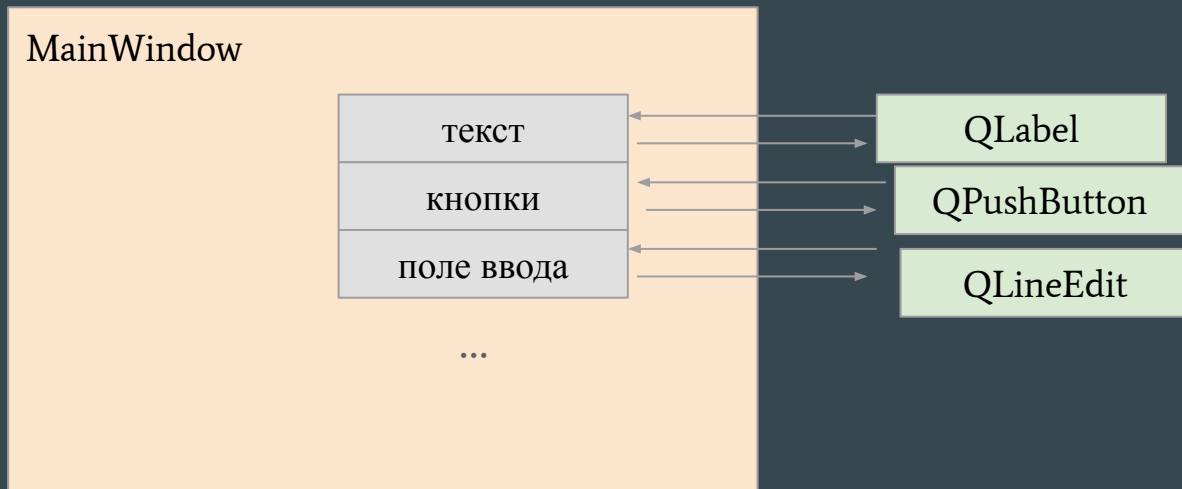
MainWindow

QDialog (QWidget)

Модальное, диалоговое окно (для вторичных окон например)

DialogWindow

# main-window.





# import base lib.

```
import sys
```

```
import tkinter as tk
```

```
if __name__ == '__main__':
```

```
    app = QApplication(sys.argv)
```

```
    window = MainWindow()
```

```
    window.show()
```

```
    sys.exit(app.exec())
```

# main-window

```
class MainWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle('Простое приложение PyQt6')  
        self.setGeometry(100, 100, 300, 200)
```



*размеры окна*

# ./QtGui/QtGui

```
import tkinter as tk
```

```
class QtGui:
```

```
    def __init__(self, argv: list[str]) -> None:
```

```
        self.root = tk.Tk()
```

```
    def event(self, a0: any) -> bool:
```

```
        pass
```

```
    def show(self):
```

```
        self.root.mainloop()
```

# ./QtWidgets/QApplication

```
class QApplication(QtGui):  
    def __init__(self, *argv, **kwargs):  
        pass  
  
    @staticmethod  
    def exec() -> int:  
        return -1
```

# ./QtWidgets/QWidget

```
class QWidget(QtGui):
```

```
    def __init__(self, *argv, **kwargs):  
        super().__init__(list(argv))
```

# ./QtWidgets/QLabel

```
class QLabel:
```

```
    def __init__(self, text, parent):  
        self.label = ...
```

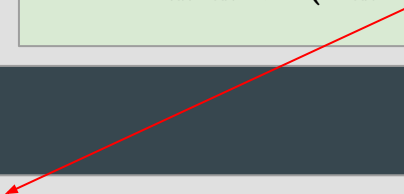
```
    def move(self, x, y):  
        self.label.place(x=x, y=y)
```

# ./QtWidgets/QPushButton

MainWindow

```
button = QPushButton(..)
```

```
button.clicked(self.callback)
```



send `callback()` -> QPushButton -> `clicked(..)`

## QPushButton

```
def __init__(self, text, parent=None):
    self.parent = parent.root if parent else tk.Tk()
    self.text = text
    self.callback = None
    self.button = tk.Button(self.parent, text=self.text,
command=self._on_click)


def setGeometry(self, x, y, width, height):
def move(self, x, y):
def setText(self, text):
def setEnabled(self, enabled: bool):
def clicked(self, callback):
    self.callback = callback
```

# ./QtWidgets/QLineEdit

MainWindow

```
username = QLineEdit(self)
```

```
username.setGeometry(50, 50, 200, 30)
```



-> QLineEdit -> **setGeometry(..)**

## QLineEdit

```
def __init__(self):  
    ...  
def setGeometry(self, x, y, width, height):  
    self.entry.place(x=x, y=y, width=width,  
height=height)  
def move(self, x, y):  
def setText(self, text):  
def text(self):
```



# Разбиение на отдельные файлы.

Давайте вынесем в отдельную папку **MyPyQt** <- папка нашего модуля. Внутри этой папки мы создадим:

- **/QtGui**
  - QtGui
- **/QtWidgets**
  - QApplication
  - QWidget
  - QLabel
  - QPushButton
  - QLineEdit

## `__init__.py`

Для корректной работы нашего модуля рекомендовано также добавить по пустому `__init__.py` в:

- **`/QtGui`**
- **`/QtWidgets`**

## new imports in main.py

```
import sys

from MyPyQt.QtWidgets.QApplication import QApplication
from MyPyQt.QtWidgets.QMainWindow import QMainWindow
from MyPyQt.QtWidgets.QLabel import QLabel
from MyPyQt.QtWidgets.QLineEdit import QLineEdit
from MyPyQt.QtWidgets.QPushButton import QPushButton
from MyPyQt.QtGui.QtGui import QtGui
```

# Простой проект

Давайте реализуем простой проект. Напишем desktop приложение по проверке вводимых данных - например вход в личный кабинет.

Проверять будем с этими данными:

```
DATA_USER = {  
    "username": "alex2834",  
    "password": "qwerty"  
}
```

# переход на реальный PyQt6

Давайте для начала установим pyqt6

```
pip install pyqt6
```

После установки удалим наш созданный ранее модуль MyPyQt... и посмотрим на наш код...

```
from MyPyQt.QtWidgets.QApplication import QApplication
from MyPyQt.QtWidgets.QMainWindow import QMainWindow
from MyPyQt.QtWidgets.QLabel import QLabel
from MyPyQt.QtWidgets.QLineEdit import QLineEdit
from MyPyQt.QtWidgets.QPushButton import QPushButton
from MyPyQt.QtGui.QtGui import QtGui
```

ПОЧЕМУ ТУТ  
ПИШЕТ ОШИБКА???

# переделка imports

Ну в целом логично - ошибка возникает из-за того, что у нас больше нету модуля MyPyQt, НО есть PyQt6, который мы ранее установили.

Исправленные импорты будут выглядеть так:

```
from PyQt6.QtWidgets import QApplication
from PyQt6.QtWidgets import QMainWindow
from PyQt6.QtWidgets import QLabel
from PyQt6.QtWidgets import QLineEdit
from PyQt6.QtWidgets import QPushButton
from PyQt6.QtGui import QtGui
```

# короткая запись imports

При этом есть более короткая запись всех импортов.

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QLabel, QLineEdit, QPushButton
```

QtGui - в целом нам не нужно импортировать.

используйте ту запись, которая вам больше удобна.. *(но рекомендована короткая)*

# fix QLineEdit

необходимо заменить:

```
self.password = QLineEdit(self, input_type="password")
```

на:

```
self.password = QLineEdit(self)
```



# fix QPushButton

необходимо заменить:

```
button.clicked(self.buttonClicked)
```

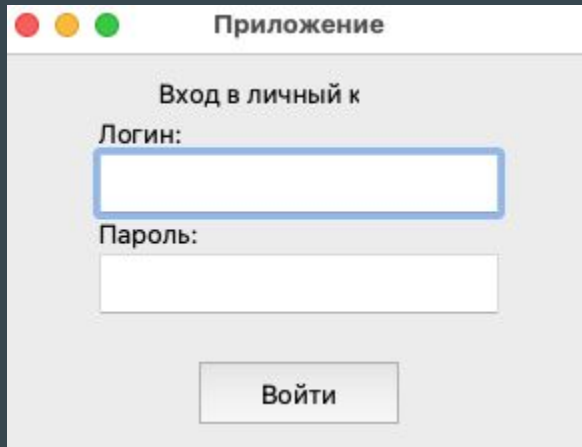
на:

```
button.clicked.connect(self.buttonClicked)
```

**connect()** - это способ сообщить программе, что когда произойдет определенное событие, нужно выполнить определенное действие.

Например, когда пользователь нажимает на кнопку, connect() указывает, какая функция или метод должны быть вызваны в ответ на это действие.

# test work app



Приложение

Вход в личный к

Логин:

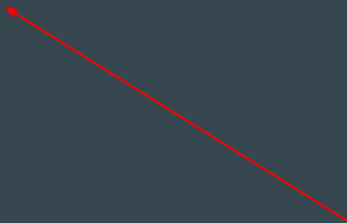
Пароль:

Войти

# QTextEdit.

**QTextEdit** используется для отображения и редактирования текста.

```
self.text_edit = QTextEdit()  
self.text_edit.setPlainText(new_text)
```



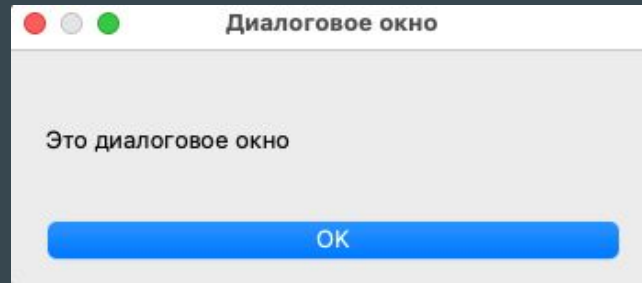
*для установки  
(изменения) текста*

# QDialog.

**QDialog** - это специальное окно, которое используется для отображения диалоговых сообщений или запросов к пользователю. Оно может быть использовано для вывода предупреждений, запросов подтверждения, ввода данных или любого другого взаимодействия с пользователем в отдельном окне, которое блокирует основное окно приложения до его закрытия.

# QDialog. Пример.

```
class DialogWindow(QDialog):  
    def __init__(self):  
        super().__init__()  
        self.setWindowTitle('Диалоговое окно')  
  
        layout = QVBoxLayout()  
  
        label = QLabel('Это диалоговое окно', self)  
        layout.addWidget(label)  
  
        ok_button = QPushButton('OK', self)  
        ok_button.clicked.connect(self.accept)  
        layout.addWidget(ok_button)  
        self.setLayout(layout)
```



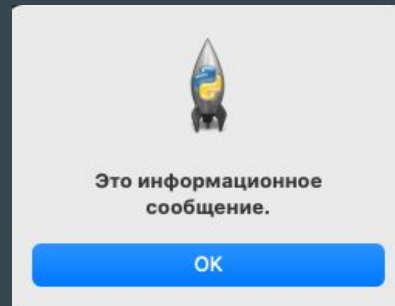
# QMessageBox.

**QMessageBox** - это стандартное диалоговое окно, которое используется для отображения сообщений различных типов, таких как информационные сообщения, предупреждения, ошибки или запросы подтверждения. Оно предоставляет удобные методы для создания и настройки различных видов сообщений, что позволяет взаимодействовать с пользователем в приложении.

# QMessageBox. Пример.

```
# Создание и настройка сообщения
message_box = QMessageBox()
message_box.setWindowTitle('Информация')
message_box.setText('Это информационное сообщение.')
message_box.setIcon(QMessageBox.Icon.Information)
message_box.setStandardButtons(QMessageBox.StandardButton.Ok)

# Отображение сообщения и ожидание ответа пользователя
message_box.exec()
```



# QVBoxLayout.

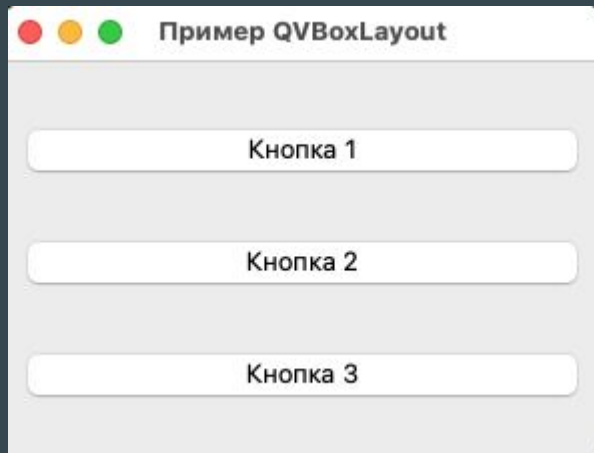
**QVBoxLayout** - это менеджер компоновки, который располагает виджеты вертикально, один под другим. Он используется для организации интерфейса приложения по вертикали, упорядочивая виджеты в порядке их добавления в макет.

```
from PyQt6.QtWidgets import QApplication, QMainWindow, ..., QVBoxLayout
```



# QVBoxLayout. Пример.

```
class MainWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()   
        self.setWindowTitle('Пример QVBoxLayout')  
        self.setGeometry(100, 100, 300, 200)  
  
        layout = QVBoxLayout()  
  
        button1 = QPushButton('Кнопка 1')  
        button2 = QPushButton('Кнопка 2')  
        button3 = QPushButton('Кнопка 3')  
        layout.addWidget(button1)  
        layout.addWidget(button2)  
        layout.addWidget(button3)  
        central_widget = QWidget()  
        central_widget.setLayout(layout)  
        self.setCentralWidget(central_widget)
```

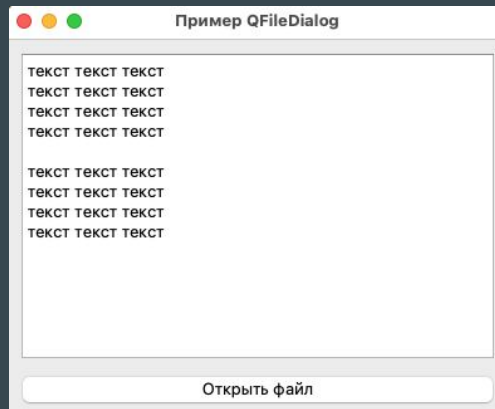
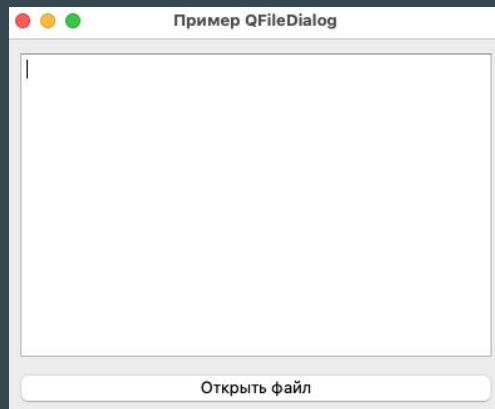


# QFileDialog.

**QFileDialog** - это стандартное диалоговое окно, которое позволяет пользователю выбирать файлы или директории на компьютере. Оно предоставляет интерфейс для открытия и сохранения файлов, а также для выбора папок. QFileDialog удобно использовать, когда требуется работа с файлами в приложении, например, для открытия или сохранения документов.

# QFileDialog. Пример.

```
class MainWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()   
        self.setWindowTitle('Пример QFileDialog')  
        self.setGeometry(100, 100, 400, 300)  
  
        layout = QVBoxLayout()  
        self.text_edit = QTextEdit()  
        layout.addWidget(self.text_edit)  
        self.open_button = QPushButton('Открыть файл', self)  
        self.open_button.clicked.connect(self.open_file)  
        layout.addWidget(self.open_button)  
        central_widget = QWidget()  
        central_widget.setLayout(layout)  
        self.setCentralWidget(central_widget)  
  
    def open_file(self):  
        file_dialog = QFileDialog(self)  
        file_dialog.setFileMode(QFileDialog.FileMode.ExistingFile)  
        file_dialog.setNameFilter("Текстовые файлы (*.txt)")  
        if file_dialog.exec():  
            file_path = file_dialog.selectedFiles()[0]  
            with open(file_path, 'r') as file:  
                file_content = file.read()  
                self.text_edit.setPlainText(file_content)
```



# QAction.

**QAction** представляет действие, которое пользователь может выполнить в приложении, например, нажатие кнопки или выбор пункта меню.

```
from PyQt6.QtGui import QAction
```

# QAction. Пример.

```
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow, QMessageBox
from PyQt6.QtGui import QAction
```

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Пример QAction')
        self.setGeometry(100, 100, 300, 200)

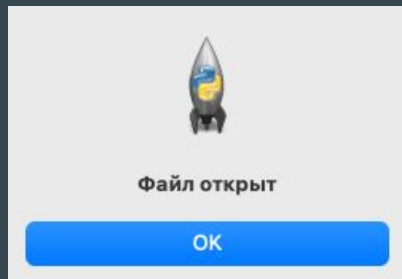
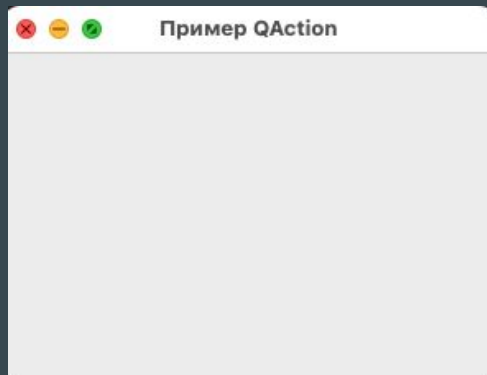
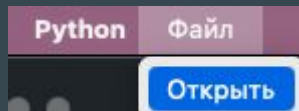
        # Создание QAction
        open_action = QAction('Открыть', self)
        open_action.triggered.connect(self.open_file)

        # Добавление QAction в меню
        file_menu = self.menuBar().addMenu('Файл')
        file_menu.addAction(open_action)

    def open_file(self):
        QMessageBox.information(self, 'Сообщение', 'Файл открыт')
```

```
if __name__ == '__main__':
```

```
...
```



# PyQt6 Documentation

<https://doc.qt.io/qtforpython-6/>

<https://pypi.org/project/PyQt6/>

<https://www.riverbankcomputing.com/static/Docs/PyQt6/>