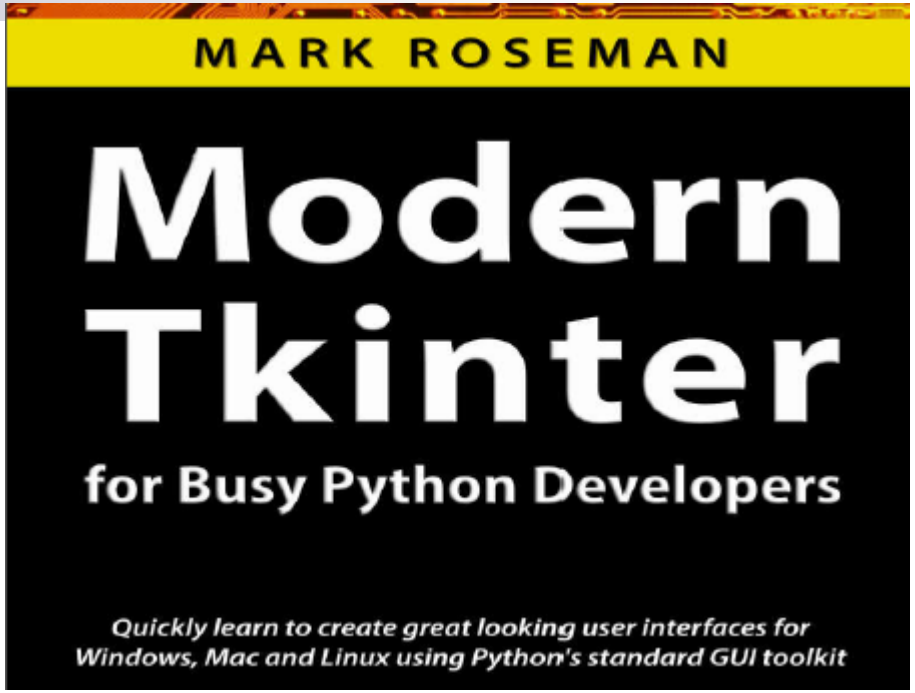
Объектно-ориентированное программирование на Python

Осипов Никита Алексеевич

ЛЕКЦИЯ 12. СОЗДАНИЕ GUI В PYTHON



Учебные вопросы:

1. Создание GUI на основе библиотеки Tkinter.
2. Применение ООП при создании компонентов.

https://python-scripts.com/category/gui

tkinter — Python interface to Tcl/Tk

- Tkinter – это пакет для Python, предназначенный для работы с библиотекой Tk.
- Библиотека Tk содержит компоненты графического интерфейса пользователя (graphical user interface – GUI), написанные на языке программирования Tcl
- Tkinter реализован как оболочка Python для Tcl интерпретатора, встроенного в интерпретатор Python

➢ Существует и несколько других популярных графических инструментов для Python:
Наиболее популярными являются wxPython, PyQt , PyGTK , PyGame

Понятие программы GUI

- Tkinter импортируется стандартно для модуля Python любым из способов:

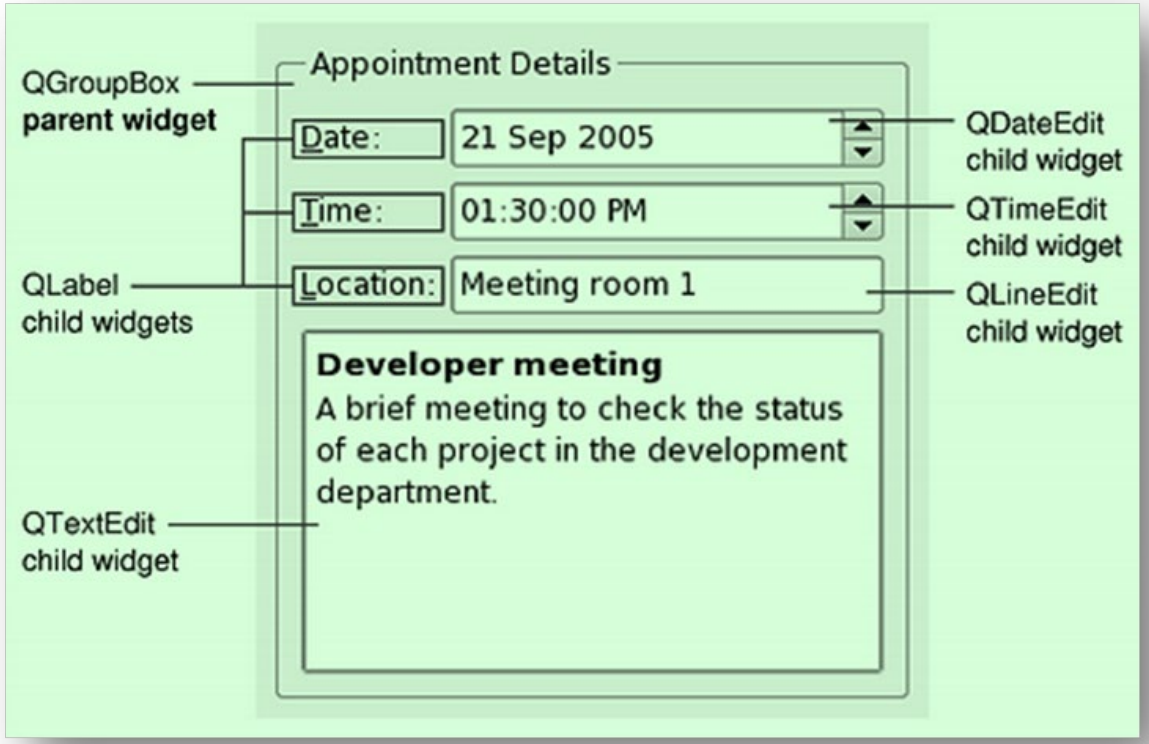
```
import tkinter, from tkinter import *, import tkinter as tk
```

Чтобы написать GUI-программу, надо выполнить следующее:

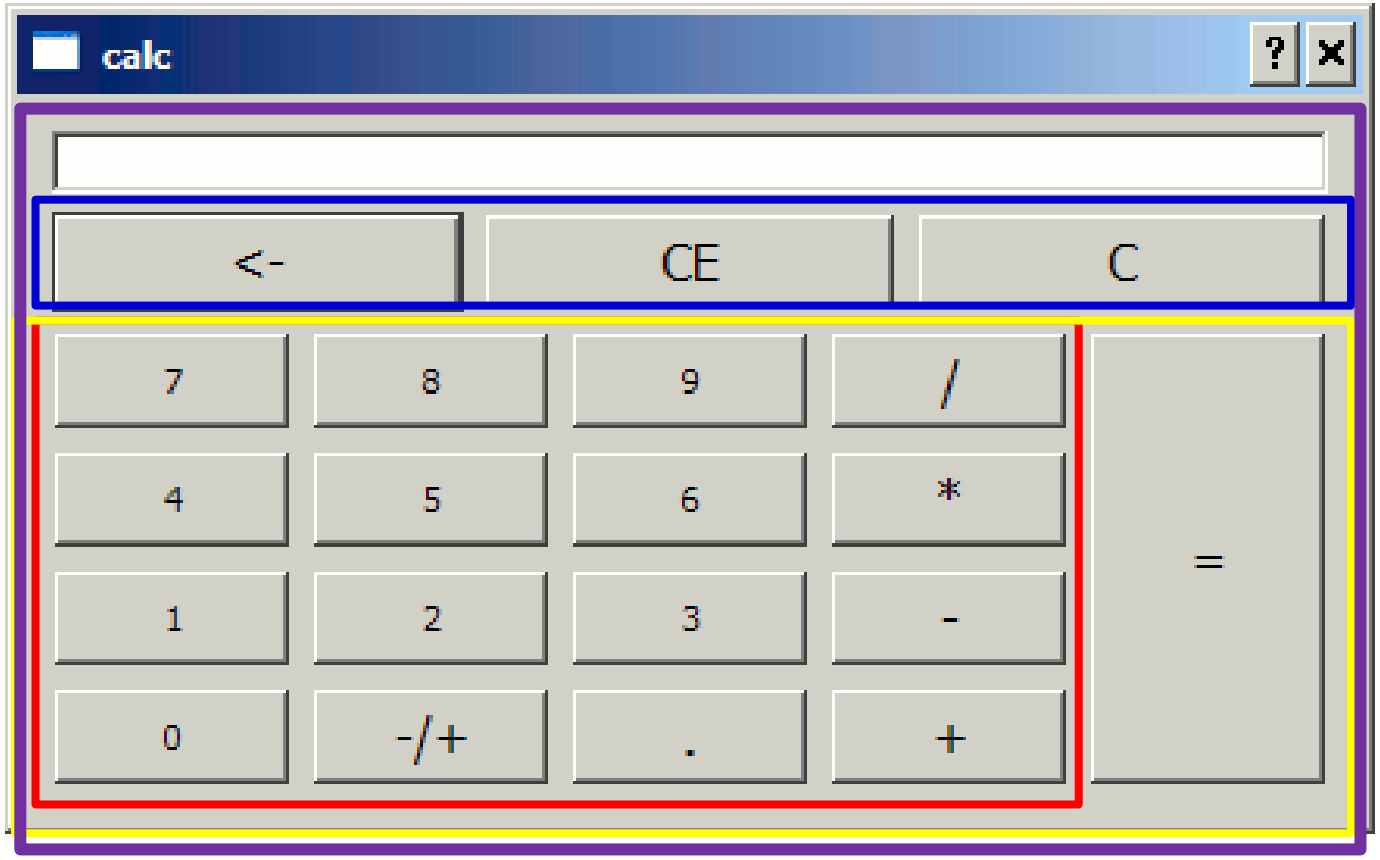
1. Создать главное окно.
2. Создать виджеты и выполнить конфигурацию их свойств (опций).
3. Определить события, то есть то, на что будет реагировать программа.
4. Определить обработчики событий, то есть то, как будет реагировать программа.
5. Расположить виджеты в главном окне.
6. Запустить цикл обработки событий.

Размещение на форме

- Визуальные компоненты могут динамически изменять размер, исчезать или появляться в следствии работы логики программы
- Удобно размещать группы виджетов вертикально либо горизонтально на форме
- Форма, в которой размещают виджеты, может динамически изменять размер при работе программы



Пример. UI калькулятора



Макеты представляют собой контейнеры для виджетов, которые будут удерживать их на определённой позиции относительно других элементов

Создание программы GUI

Demo_Tk01.py

```
from tkinter import *
root = Tk()
root.title("Заголовок главного окна")

e = Entry(width=20)
b = Button(text="Выполнить задачу")
l = Label(bg='black', fg='white', width=20)

def strToSortlist(event):
    s = e.get()
    s = s.split()
    s.sort()
    l['text'] = ' '.join(s)

b.bind('<Button-1>', strToSortlist)
e.pack()
b.pack()
l.pack()
root.mainloop()
```

Объект окна верхнего уровня создается от класса Tk модуля tkinter

текстовое поле (entry), метка (label) и кнопка (button) – объекты создаются от соответствующих классов модуля tkinter

У функций, которые вызываются при наступлении события с помощью метода bind(), должен быть один параметр - event (событие)

Связь вызова функции с событием

Размещение элементов с помощью менеджера геометрии

Метод mainloop() объекта Tk запускает главный цикл обработки событий

Настройка свойств виджетов

TkButLabEntry.py

- Существует три способа конфигурирования свойств виджетов:
  - в момент создания объекта

```
b1 = Button(text="Изменить", width=15, height=3)
```

  - с помощью метода config(), он же configure()

```
label1.config(bd=20, bg='#ffaaaa')
```

  - путем обращения к свойству как к элементу словаря

```
def change():
    b1['text'] = "Изменено"
    b1['bg'] = '#000000'
    b1['activebackground'] = '#555555'
    b1['fg'] = '#ffffff'
    b1['activeforeground'] = '#ffffff'
```

Добавление функциональности элементу UI

Tk04.py

Tk05.py

- Виджет, событие и действие связываются между собой с помощью метода bind()

```
def change(event):
    b['fg'] = "red"
    b['activeforeground'] = "red"

b = Button(text='RED', width=10,
height=3)
b.bind('<Button-1>', change)
b.bind('<Return>', change)
```

Действие оформляется как функция (или метод), которая вызываются при наступлении события

- Примеры:
  - виджет – кнопка, событие – клик по ней левой кнопкой мыши, действие – отправка сообщения.
  - виджет – текстовое поле, событие – нажатие Enter, действие – получение текста из поля методом get() для последующей обработки программой.

Tk03Frame_pack.py | Tk06.py | Tk06a.py | Demo_Tk03_OOP.py | Tk07Frame.py

Создание рамки – Frame

- Класс Frame играет роль контейнера для других виджетов

```
class Example(Frame):

    def __init__(self, parent):
        Frame.__init__(self,
parent, background="white")
```

Фреймы размещают на главном окне, а уже в фреймах – виджеты

```
f_top = Frame(root)
f_bot = Frame(root)
l1 = Label(f_top, width=7, height=4, bg='yellow', text="1")
l2 = Label(f_top, width=7, height=4, bg='orange', text="2")
l3 = Label(f_bot, width=7, height=4, bg='lightgreen', text="3")
l4 = Label(f_bot, width=7, height=4, bg='lightblue', text="4")

f_top.pack()
f_bot.pack()
```

Виджеты добавляются в конкретный фрейм

Tk08Menu.py

Создание меню – Menu

- Класс Menu объединяет группу команд

```
menubar = Menu(self.parent)
self.parent.config(menu=menubar)

fileMenu = Menu(menubar)

fileMenu.add_command(label="Exit",
command=self.onExit)

menubar.add_cascade(label="File",
menu=fileMenu)

submenu = Menu(fileMenu)
submenu.add_command(label="New")
```

Меню (панель меню) размещают на главном окне

В объект меню добавляются команды

Меню добавляется на панель меню при помощи метода add_cascade()

Подменю – меню, встроенные в другие объекты меню

Tk09D.py

Создание диалогов

- Окна сообщений – это удобные диалоги, которые показывают пользователю сообщения приложения

```
from tkinter import  messagebox as mbox

 def onError(self):
        mbox.showerror("Error", "Could not open file")

    def onWarn(self):
        mbox.showwarning("Warning", "Deprecated
function call")

    def onQuest(self):
        mbox.askquestion("Question", "Are you sure to
quit?")

    def onInfo(self):
        mbox.showinfo("Information", "Download
```

Используются функции для вывода диалога

Применение стандартных диалогов

Tk09Dcolor.py

Tk09Dfile.py

- Стандартные диалоги – это удобные диалоги, которые показывают пользователю привычные окна

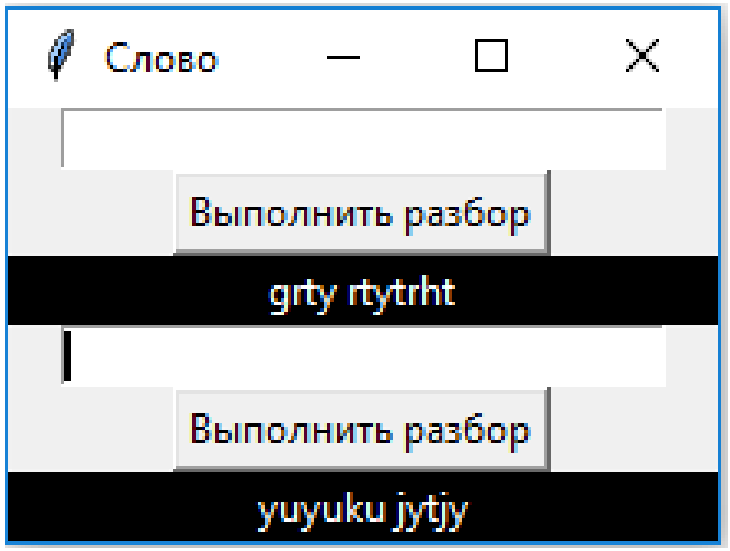
```
from tkinter import colorchooser as tkColorChooser
from tkinter import filedialog as tkFileDialog

def onChoose(self):
        (rgb, hx) = tkColorChooser.askcolor()
        self.frame.config(bg=hx)

def onOpen(self):
        ftypes = [('Python files', '*.py'), ('All files', '*')]
        dlg = tkFileDialog.Open(self, filetypes = ftypes)
        fl = dlg.show()
```

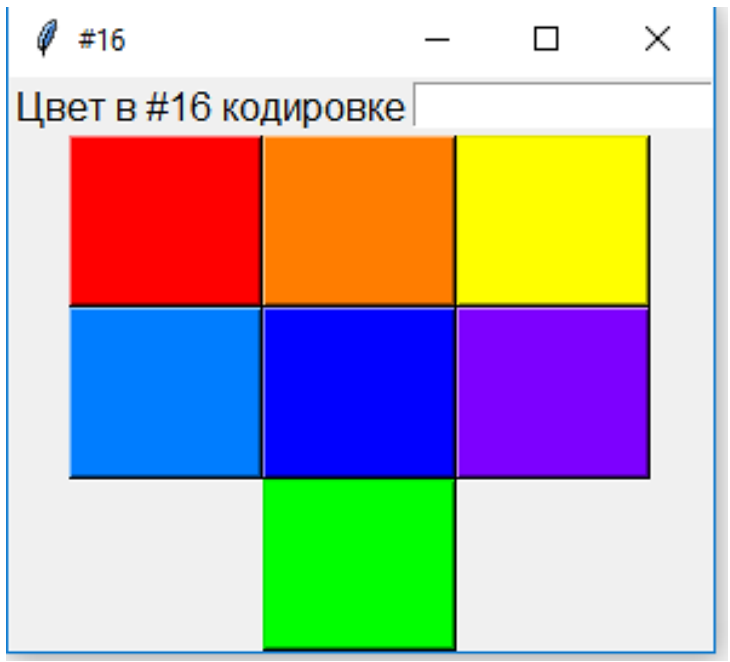
Используются функции для вывода диалога

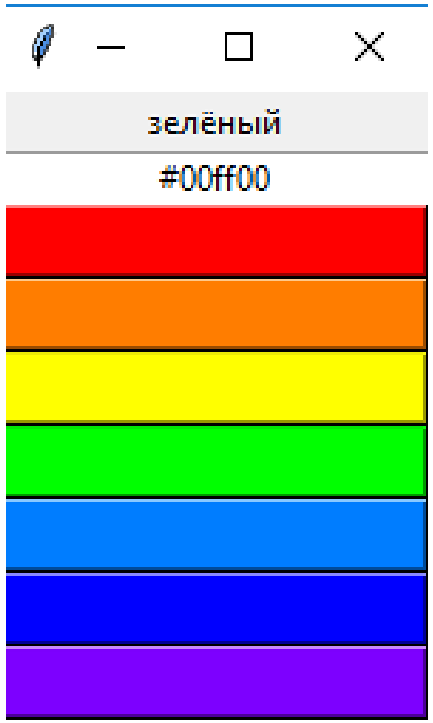
Demo_Tk02_OOP.py

Применение ООП при создании компонентов

- Требуется несколько похожих объектов-блоков, состоящих из метки, кнопки, поля.
  - У кнопки каждой группы должна быть своя функция-обработчик клика







Demo_Tk03_OOP.py

Demo_Tk031_OOP.py

PyQt

- PyQt5 – это набор Python библиотек для создания графического интерфейса на базе платформы Qt5
- Чтобы установить PyQt5 при помощи pip, выполните команду:

```
sudo pip3 install PyQt5
pip install PyQt5
```

Выбор базового класса для создания класса формы

- **QMainWindow** — в большинстве случаев, наиболее функциональный.
  - Наследуясь от данного класса получаем уже готовые средства для размещения меню, строки статуса и центрального поля, которое можно реализовать как в стиле SDI, так и в стиле MDI
- **QWidget** — этот класс является простейшим виджетом.
  - В терминологии Qt это простейший элемент, с которым связана какая-то графическая область на экране.
  - Как базовый класс для главного окна, используется, как правило, при создании простых одноформенных приложений
- **QDialog** — базовый класс для создания модальных диалоговых окон

Создание программы GUI

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget


if __name__ == '__main__':

    app = QApplication(sys.argv)

    w = QWidget()
    w.resize(250, 150)
    w.move(300, 300)
    w.setWindowTitle('Simple')

    w.show()

    sys.exit(app.exec_())
```

Основные виджеты расположены в PyQt5.QtWidgets

Каждое приложение PyQt5 должно создать объект приложения (экземпляр QApplication)

Виджет QWidget это базовый класс для всех объектов интерфейса пользователя в PyQt5

Настройка окна-виджета

Отображение виджета на экране

главный цикл обработки событий

```
import sys
from PyQt5.QtWidgets import QWidget, QPushButton, QApplication
from PyQt5.QtCore import QCoreApplication

class Example(QWidget):

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

    def initUI(self):

        qbtn = QPushButton('Quit', self)
        qbtn.clicked.connect(QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(50, 50)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Quit button')
        self.show()
```

Класс Example наследуется от класса QWidget

Создание GUI делегируется методу initUI()

Настройка окна-виджета

```
if __name__ == '__main__':

    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```

События и сигналы в PyQt5

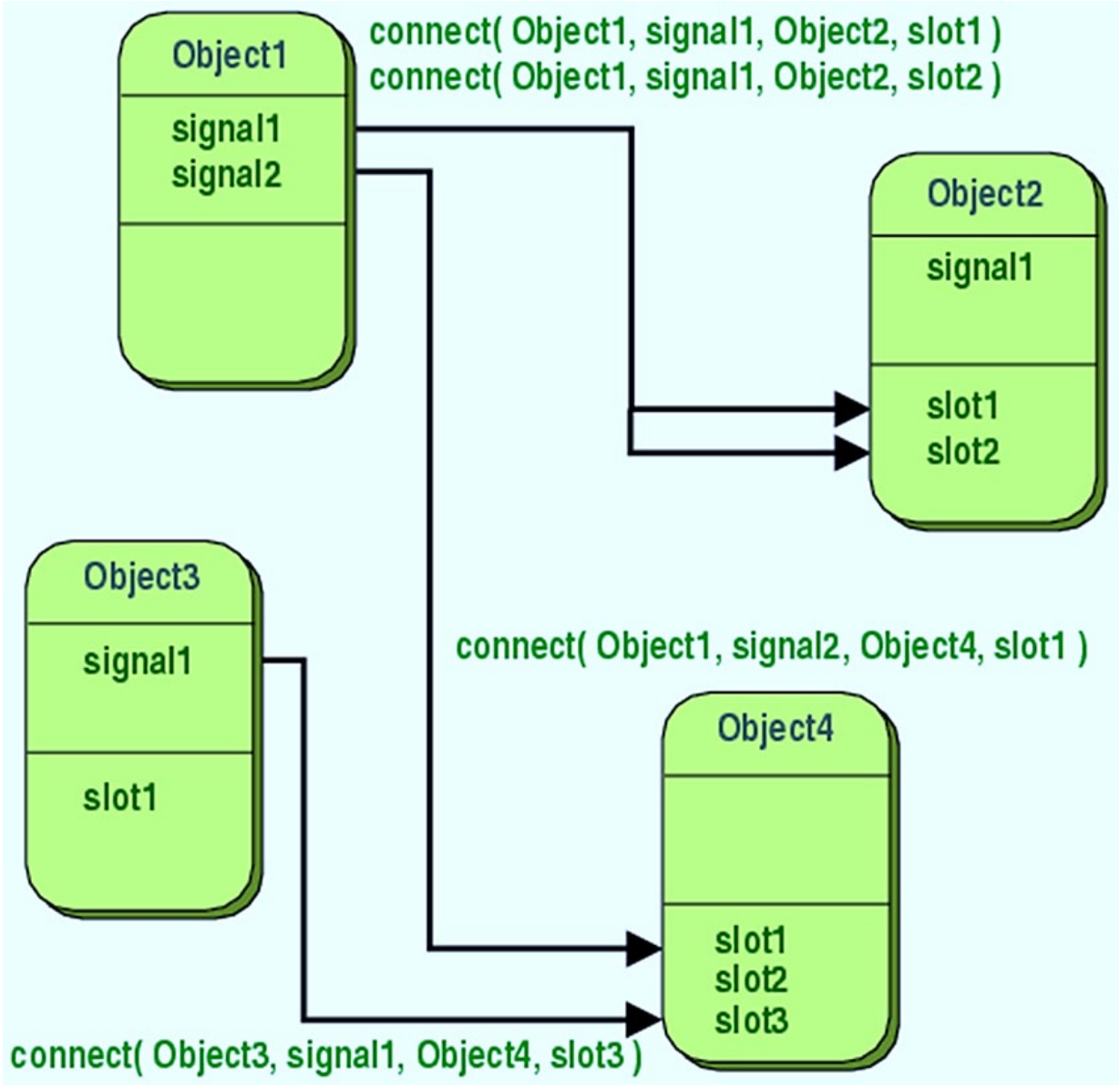
- UI-приложения являются событийно-ориентированными
- Вызываем метод exec() – приложение входит в главный цикл – главный цикл получает события и отправляет их объектам
- Модель событий:
  - **Источник события** – это объект, состояние которого меняется. Он вызывает событие. Событие инкапсулирует изменение состояния в источнике события.
  - **Цель события** – это объект, которому требуется уведомление.
  - **Объект источника** события делегирует задачу обработки события цели события.

Механизм сигналов и слотов

- Сигналы и слоты используются для связи между объектами
- Сигнал срабатывает тогда, когда происходит конкретное событие
- Слот может быть любой функцией – вызывается, когда срабатывает его сигнал

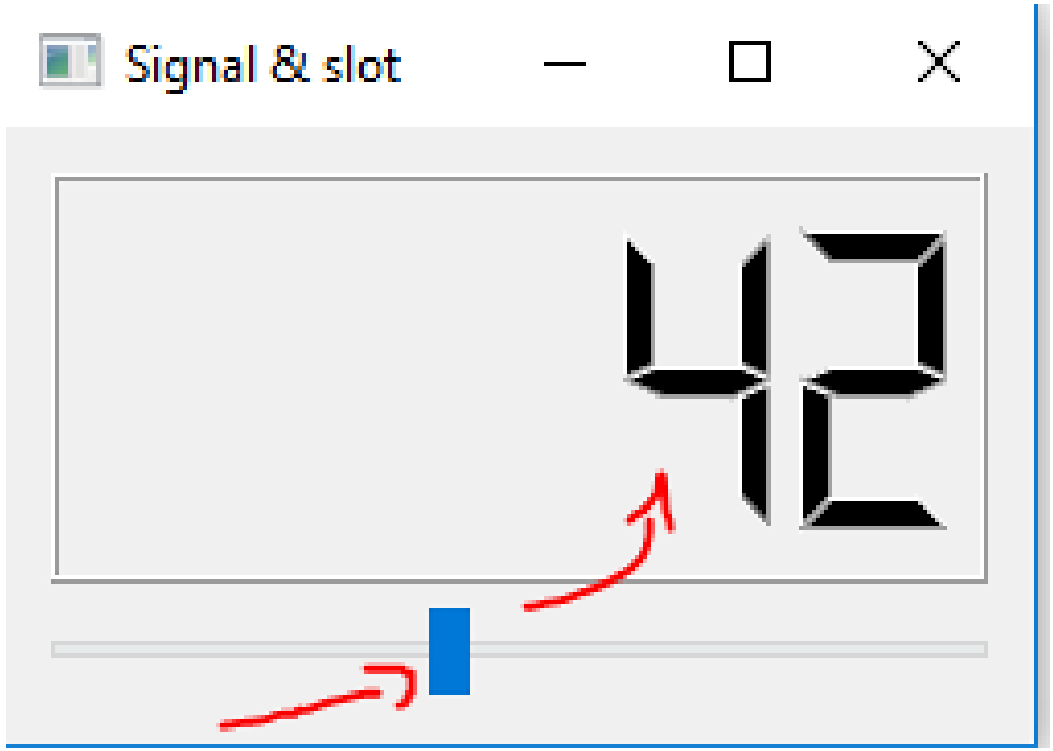
```
sld.valueChanged.connect(lcd.display)
```

Присоединяем сигнал *valueChanged* слайдера к слоту *display* числа lcd



Пример

Signal_Slot_01.py



Переопределение обработчика события

- События в PyQt5 могут обрабатываться путём переопределения обработчиков, например:
  - Если мы нажимаем клавишу Esc, то приложение завершается

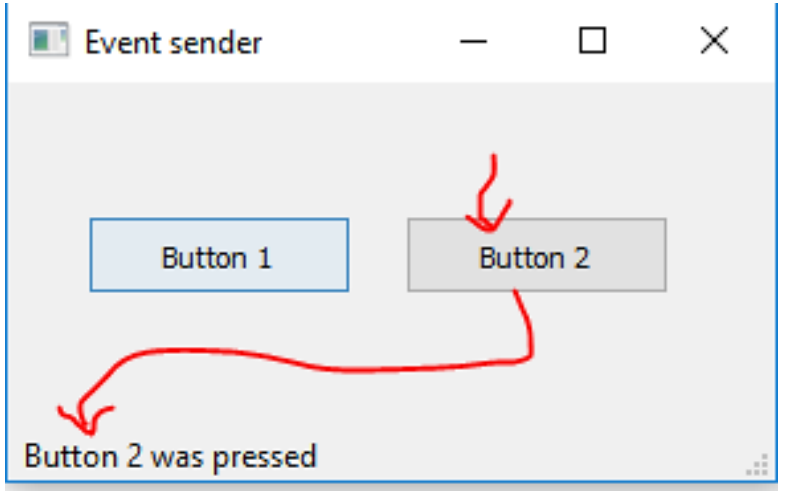
```
def keyPressEvent(self, e):

    if e.key() == Qt.Key_Escape:
        self.close()
```

Контроль отправителя события

Signal_Slot_02.py

- Для определения какой именно виджет является отправителем сигнала используется метод sender(), например:
  - есть две кнопки (подключаются к одному слоту)

```
btn1.clicked.connect(self.buttonClicked)
btn2.clicked.connect(self.buttonClicked)
```



  - в методе button Clicked() определяем, какую из кнопок нажали с помощью метода sender()

```
def buttonClicked(self):

    sender = self.sender()
    self.statusBar().showMessage (sender.text() + ' was pressed')
```

Создание пользовательских сигналов

Signal_Slot_03.py

- Объекты, создаваемые из QObject, могут посылать сигналы

Сигнал создаётся с помощью pyqtSignal() как атрибут класса Communicate

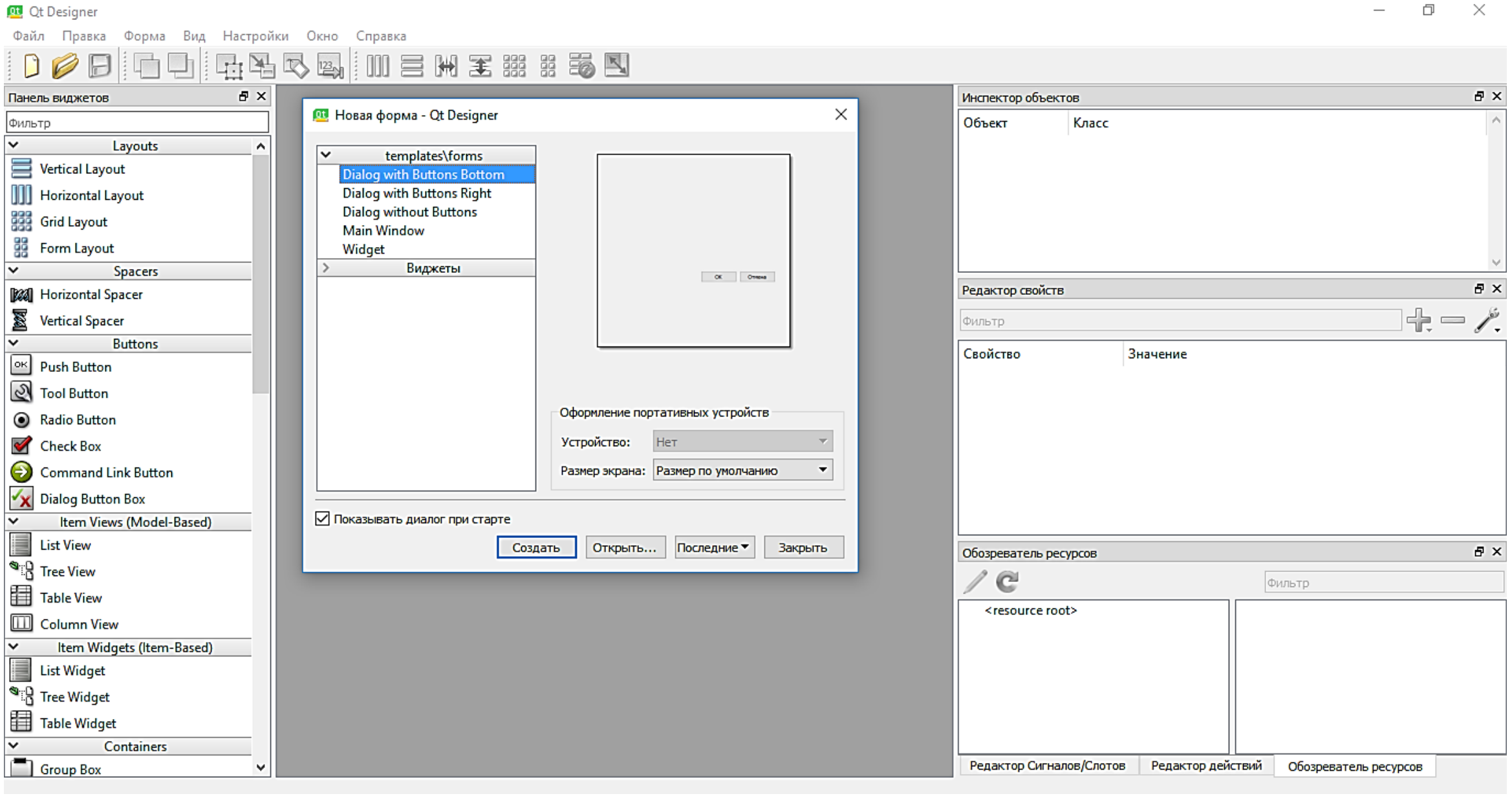
```
class Communicate(QObject):
    closeApp = pyqtSignal()
```

Пользовательский сигнал closeApp присоединяется к слоту close() класса QMainWindow

Кликаем на окне курсором мыши, посылается сигнал closeApp – приложение завершается

```
class Example(QMainWindow):

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

    def initUI(self):
        self.c = Communicate()
        self.c.closeApp.connect(self.close)
        self.show()

    def mousePressEvent(self, event):
        self.c.closeApp.emit()
```

Применение дизайнера PyQt5 Designer



Применение дизайнера PyQt5

PyQt5 Designer экспортирует форму в XML с расширением .ui

Для использования этого дизайна есть два способа:

- Загрузить файл .ui в код Python
- Конвертировать файл .ui в файл .py при помощи pyuic5

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Form</class>
 <widget class="QWidget" name="Form">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Form</string>
  </property>
  <widget class="QWidget" name="verticalLayoutWidget">
   <property name="geometry">
    <rect>
     <x>90</x>
     <y>40</y>
     <width>160</width>
     <height>80</height>
    </rect>
   </property>
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <widget class="QPushButton" name="pushButton">
      <property name="text">
       <string>PushButton</string>
      </property>
     </widget>
    </item>
   </layout>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Применение дизайнера PyQt5 Designer

PyQt5 Designer экспортирует
форму в XML с расширением .ui

Для использования этого дизайна есть
два способа:

- Загрузить файл .ui в код Python

```
from PyQt5 import QtWidgets, uic
import sys

app = QtWidgets.QApplication([])
win = uic.loadUi("mydesign.ui") # расположение вашего файла .ui

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

Применение дизайнера PyQt5 Designer

PyQt5 Designer экспортирует форму в XML с расширением .ui

Для использования этого дизайна есть два способа:

- Конвертировать файл .ui в файл .py при помощи pyuic5

```
pyuic5 mydesign.ui -o mydesign.py
```

Конвертация файла .ui в файл .py безопаснее для кодирования и быстрее для загрузки

```
from PyQt5 import QtWidgets
from mydesign import Ui_MainWindow  # импорт
import sys


class mywindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(mywindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)


app = QtWidgets.QApplication([])
application = mywindow()
application.show()

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

Визуальный редактор слота/сигнала