TECHNISCHE UNIVERSITÄT DRESDEN

PYTHONKURS FÜR INGENIEURINNEN

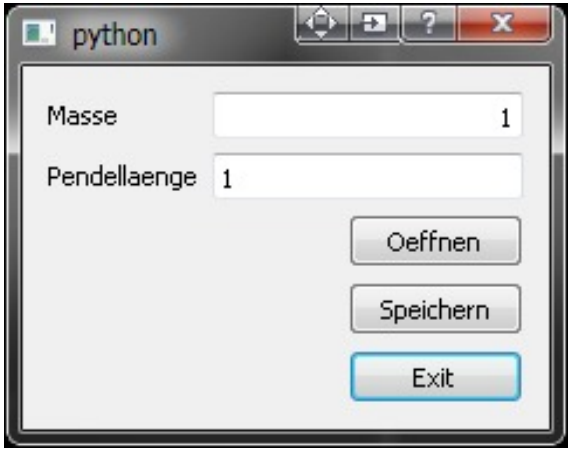
GUI Programmierung mit PyQt, Teil 2

Folien: Sebastian Voigt, Carsten Knoll, TU Dresden

Dresden, WiSe 2020/21

Wiederholung

- Widget = rechteckiger Zeichenbereich auf dem Bildschirm
  - viele Widgets sind als Steuerelemente bekannt (Buttons etc.)
- Layout = passt Größe und Anordnung der Widgets dynamisch an
- Arten von Layouts: horizontal, vertikal, grid
- Widgets und Layouts stehen in Eltern-Kind-Beziehungen zueinander
- Bisher: Anwendung als Dialogfenster (Nutzung der Klasse `QDialog`)



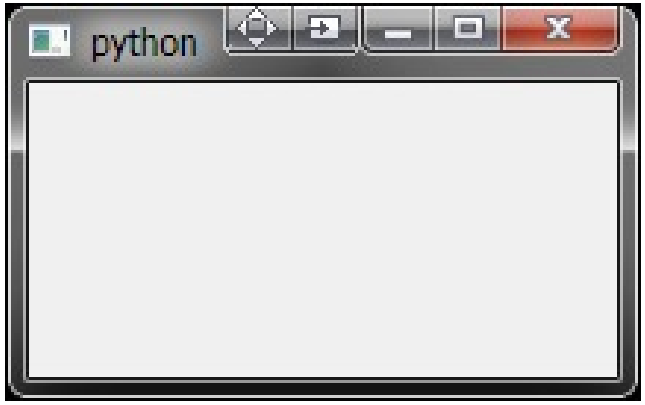
Aber: viele Elemente graphischer Benutzerschnittstellen stehen bei Verwendung von `QDialog` nicht zur Verfügung

Hauptfenster für Anwendungen

Für „echte“ Anwendungen wird `QMainWindow` verwendet:

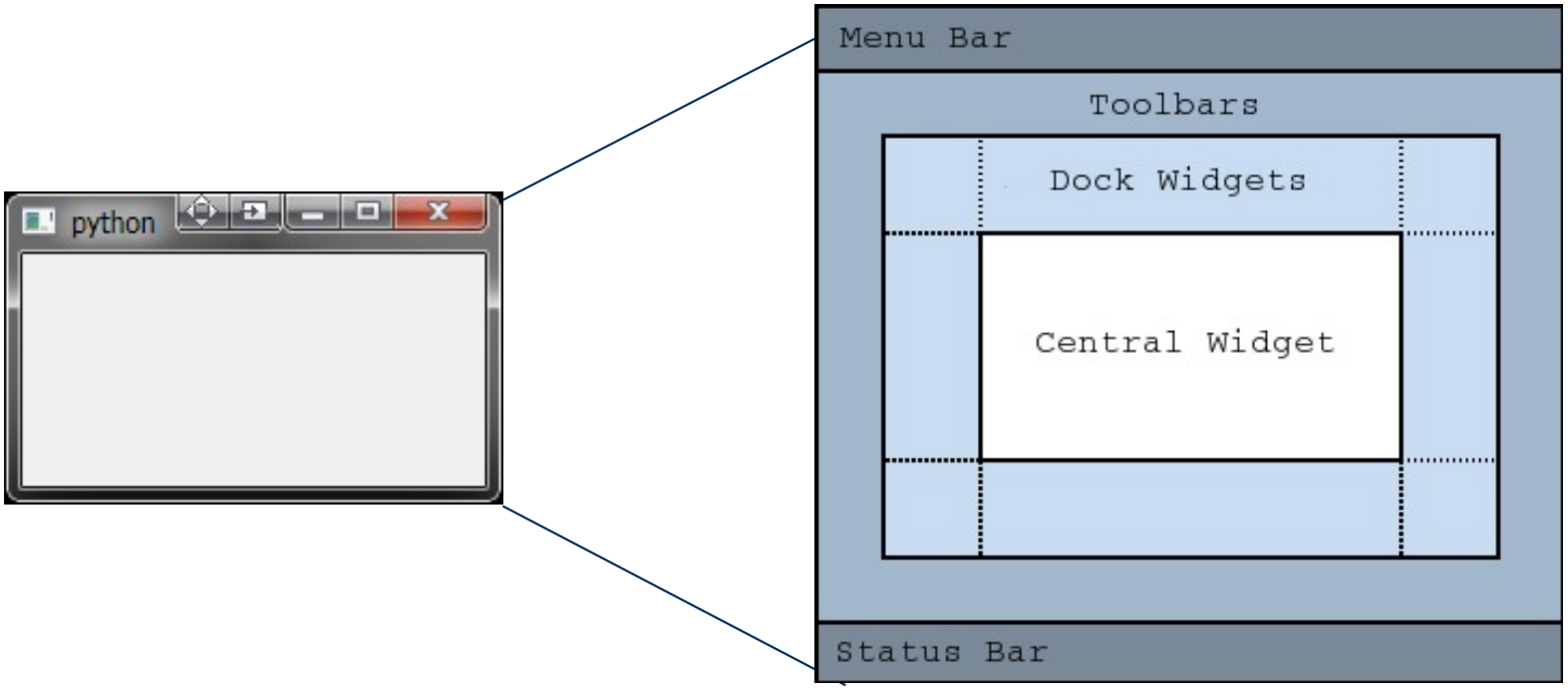
Listing: example-code/main-example1.py

```
import PyQt5.QtWidgets as QtWidgets

class Gui(QtWidgets.QMainWindow):
    """
    Eigene Klasse (abgeleitet von QMainWindow)
    Diese Klasse tut (erstmal) noch nichts.
    """
    def __init__(self):
        # Konstruktor der Basis-Klass aufrufen
        QtWidgets.QMainWindow.__init__(self)


app = QtWidgets.QApplication([])

gui = Gui() # Instanz von obiger Klasse anlegen
gui.show()
app.exec_()
```



QMainWindow

`QMainWindow` sieht bereits Platzhalter bzw. definierte Bereiche innerhalb des Anwendungsfensters für Menüs, Toolbars usw. vor:



Quelle:
http://qt-project.org/doc/qt-4.8/QMainWindow.html

Menüs

- Menüs können direkt über die Menu Bar von QMainWindow angelegt werden:

Listing: example-code/main-example2.py (12-13)

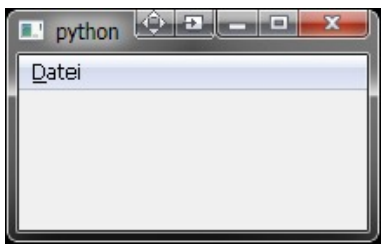
```
        # self.menuBar ist eine Methode der Basis-Klasse
        self.menu_file = self.menuBar().addMenu('&Datei')
```

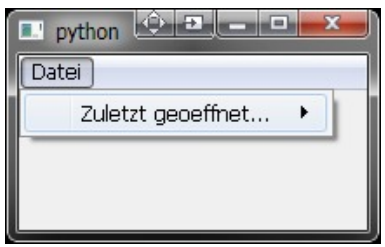
- Menüs können auch ineinander geschachtelt werden:

Listing: example-code/main-example2.py (15)

```
        self.menu_recent = self.menu_file.addMenu('Zuletzt geöffnet...')
```

- Menüs definieren nur die Struktur, keine klickbaren Einträge
- Das „&“ im Namensstring definiert Shortcurt (Alt+D → öffnet Datei-Menü)





Actions

- Instanzen der Klasse `QAction`
- Vorteil: können in verschiedener Gestalt in Erscheinung treten
  - Menüeintrag, Button, Tasten-Kombination, ...
- Action zum Beenden des Programms:

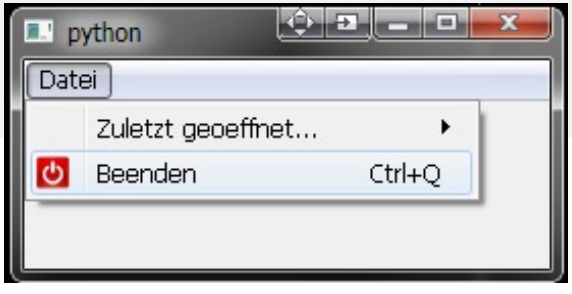
Listing: example-code/main-example3.py (17-20)

```
        self.act_exit = QtWidgets.QAction(self)
        self.act_exit.setText('Beenden')
        self.act_exit.setIcon(QtGui.QIcon('../uebung/data/exit.png'))
        self.act_exit.setShortcut('Ctrl+Q')
```

- Actions stellen eine abstrakte Interaktionsmöglichkeit mit den Benutzer dar
- `self.act_exit` bisher nur erstellt also noch zum Menü hinzufügen:

Listing: example-code/main-example3.py (22)

```
        self.menu_file.addAction(self.act_exit)
```



Actions II

- Wenn Actions „getriggert“ werden, kann darauf mit einem Funktionsaufruf reagiert werden:

Listing: example-code/main-example3.py (24)

```
        self.act_exit.triggered.connect(self.close)
```

- Actions können ebenso in Toolbars auftauchen. Dazu neue Toolbar anlegen:

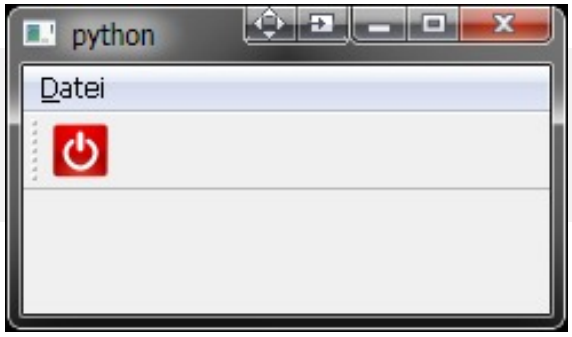
Listing: example-code/main-example4.py (26-28)

```
        self.toolbar = QtWidgets.QToolBar('Datei')
        self.toolbar.setIconSize(QtCore.QSize(24, 24))
        self.addToolBar(self.toolbar)
```

- ... und die Action hinzufügen:

Listing: example-code/main-example4.py (30)

```
        self.toolbar.addAction(self.act_exit)
```

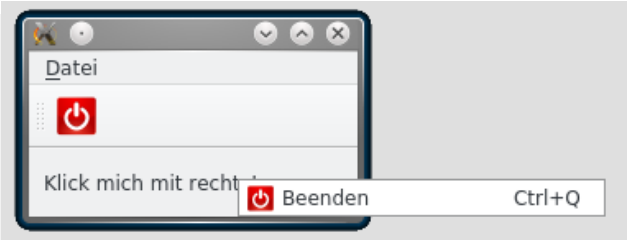


Actions III

- Actions können auch im Kontextmenü auftauchen:

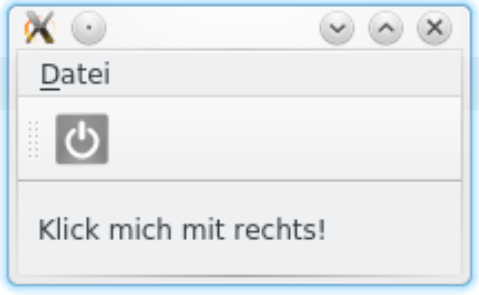
Listing: example-code/main-example5.py (32-41)

```
        self.cw = QtWidgets.QWidget()
        self.setCentralWidget(self.cw)

        self.vBox = QtWidgets.QVBoxLayout(self.cw)

        self.label = QtWidgets.QLabel('Klick mich mit rechts!')
        self.label.setContextMenuPolicy(QtCore.Qt.ActionsContextMenu)
        self.label.addAction(self.act_exit)

        self.vBox.addWidget(self.label)
```



- Actions können deaktiviert werden:

```
self.act_exit.setDisabled(True)
```



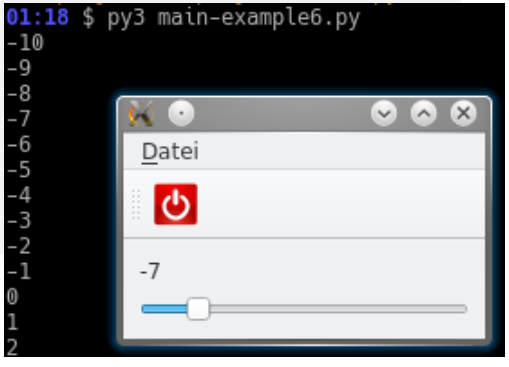
Signals und Slots

- Kommunikationsmechanismus innerhalb der Anwendung:
  - Widgets emittieren „Signale“ (z.B. bei Klick auf Button)
  - Funktionen/Methoden (sog. „Slots“) können darauf reagieren
  - Voraussetzung: Entsprechendes Signal wurde entsprechendem Slot zugeordnet (mit `connect`)

- Bsp. Ausgabe des Wertes eines Sliders (per QT-Label und auf Kommandozeile):

Listing: example-code/main-example6.py (43-53)

```
        self.slider = QtWidgets.QSlider(self)
        self.slider.setMinimum(-10)
        self.slider.setMaximum(10)
        self.slider.setOrientation(QtCore.Qt.Horizontal)
        self.vBox.addWidget(self.slider)

        self.slider.valueChanged.connect(self.label.setNum)
        self.slider.valueChanged.connect(self.print_value)

    def print_value(self, x):
        print(x)
```



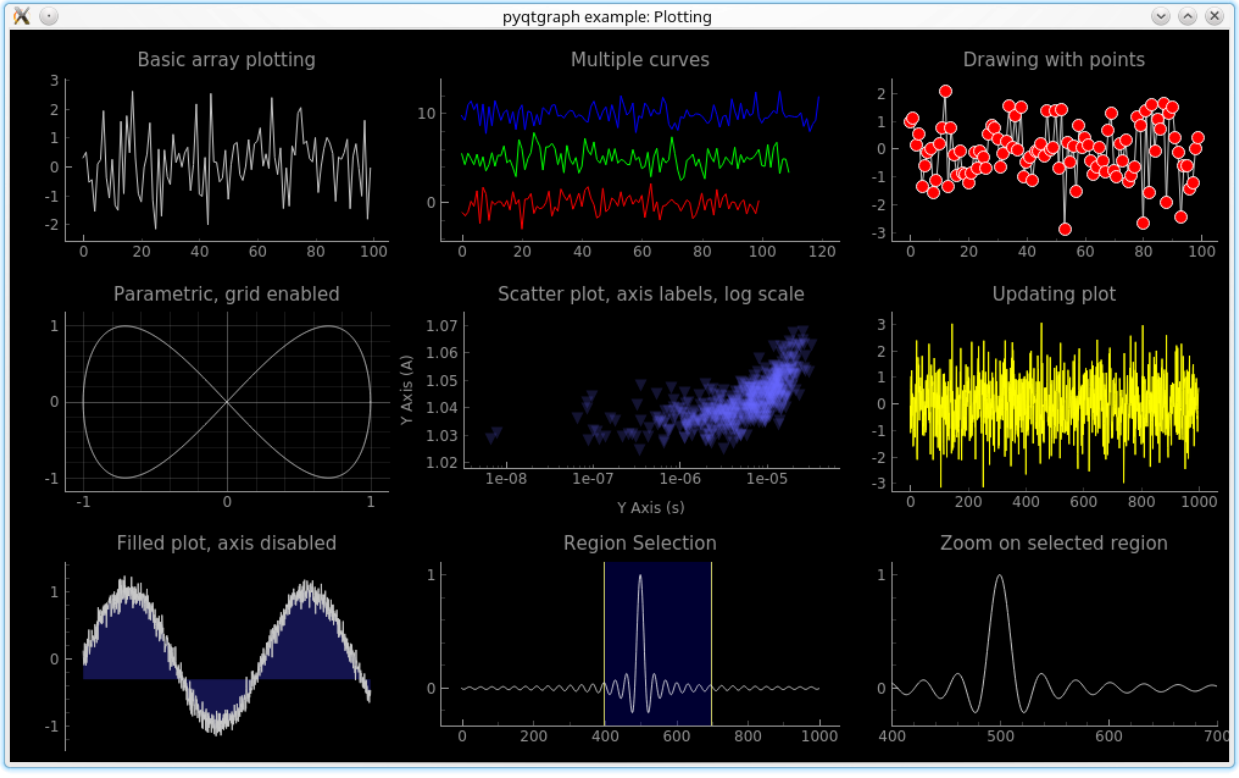
TECHNISCHE UNIVERSITÄT DRESDEN

PyQtGraph

- Plot-Bibliothek ($\approx$ matplotlib), gut in Qt-Anwendungen integrierbar + deutlich schneller

$\rightarrow$ Bei *interaktiven* Plot-Anwendungen von Vorteil

- Nachteil: Einarbeitungsaufwand
- Installation (im PC-Pool evtl. nicht notwendig): `pip install --user pyqtgraph`
- Demo-Anzeige: `python -m pyqtgraph.examples`



Links

- PyQt5 Übersicht:
  http://pyqt.sourceforge.net/Docs/PyQt5/index.html
- PyQt5 Module:
  http://pyqt.sourceforge.net/Docs/PyQt5/modules.html
- PyQt5 Widgets-Modul (das wichtigste Modul):
  http://pyqt.sourceforge.net/Docs/PyQt5/QtWidgets.html
- PyQtGraph-Projekt-Webseite:
  https://github.com/pyqtgraph/pyqtgraph