



TECHNISCHE
UNIVERSITÄT
DRESDEN

EINFÜHRUNG GUI PROGRAMMIERUNG MIT PYQT4

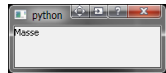
Sebastian Voigt, TU Dresden

Dresden, Wintersemester 2013/14

- GUI = Graphical User Interface
- Benutzerfreundlichkeit, insbesondere wenn Programm für Dritte entwickelt wird
- Gebrauchstauglichkeit (ISO 9241-11)
- Vielfältige Möglichkeiten der Interaktion (Maus, Gesten, etc)
- Kontrolle paralleler Programmabläufe durch den Nutzer
- GUI Toolkits für Python:
 - Tkinter
 - PyGtk
 - wxPython
 - PyQt4
 - PySide
 - ...
- Ziel: Kennenlernen der Basiselemente und einfache graphische Oberfläche zur Simulationssteuerung der Laufkatze

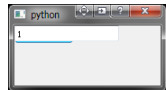
- Fenster mit Textelement

```
dialog = QtGui.QDialog()  
massLabel = QtGui.QLabel('Masse',  
    dialog)  
dialog.exec()
```



- Button und Eingabefeld hinzufügen

```
exitButton = QtGui.QPushButton('Exit',  
    dialog)  
massEdit = QtGui.QLineEdit('1',  
    dialog)
```



- Problem: Unterordnung
- Lösung: Layout

- Layouts passen Größe und Anordnung der Widgets automatisch an
- Typen: vertical, horizontal, grid
- hier vertikales QVBoxLayout

```
layout = QtGui.QVBoxLayout()  
layout.addWidget(massLabel)  
layout.addWidget(exitButton)  
layout.addWidget(massEdit)  
dialog.setLayout(layout)
```



- Wert aus QLineEdit holen und wieder setzen

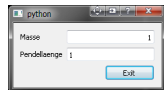
```
print massEdit.text()  
# Rueckgabewert vom Typ QString muss meist konvertiert  
  werden  
  
massEdit('10.5')
```

- Zweites Label und QLineEdit anlegen

```
lenLabel = QtGui.QLabel('Pendellaenge', dialog)
lenEdit = QtGui.QLineEdit('1', dialog)
```

- Gridlayout für Rasteranordnung

```
layout = QtGui.QGridLayout()
layout.addWidget(massLabel, 0, 0)
layout.addWidget(massEdit, 0, 1)
layout.addWidget(lenLabel, 1, 0)
layout.addWidget(lenEdit, 1, 1)
```



- Ausrichtung innerhalb des Layouts anpassen

```
layout.addWidget(exitButton, 2, 1, 1, 1,QtCore.Qt.AlignRight)
```

- Eingabefeld anpassen (Ausrichtung und Eingabefilter) - mit dem `QDoubleValidator` sind nur Zahlen als Eingabe erlaubt

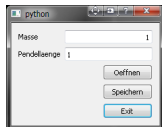
```
massEdit.setAlignment (QtCore.Qt.AlignRight)  
massEdit.setValidator (QtGui.QDoubleValidator (massEdit) )
```

- Button mit Funktion verknüpfen - wichtig: Als Argument wird immer ein Funktionsobjekt erwartet ("ohne Klammern")

```
exitButton.clicked.connect (dialog.close)
```

- Buttons mit eigener Funktion verknüpfen (am Beispiel Laden/Öffnen einer Datei)

```
def openFile():  
    print QtGui.QFileDialog.  
        getOpenFileName()  
  
def saveFile():  
    print QtGui.QFileDialog.  
        getSaveFileName()
```



```
openButton = QtGui.QPushButton('Öffnen', dialog)  
saveButton = QtGui.QPushButton('Speichern', dialog)
```

```
layout.addWidget(openButton, 3, 1, 1, 1, QtCore.AlignRight)  
layout.addWidget(saveButton, 4, 1, 1, 1, QtCore.AlignRight)
```

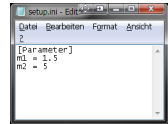
```
openButton.clicked.connect(openFile)  
saveButton.clicked.connect(saveFile)
```

- Dafür existiert in der Python Standardbibliothek das Modul `ConfigParser`

```
import ConfigParser
```

```
c = ConfigParser.SafeConfigParser()  
c.add_section('Parameter')  
c.set('Parameter', 'm1', '1.5')  
c.set('Parameter', 'm2', '5')
```

```
fid = open('setup.ini', 'w')  
c.write(fid)  
fid.close()
```



- Es können beliebige Sections (erscheinen in eckigen Klammern in der Datei) und der Schlüssel-Wert-Paare angelegt werden
- Achtung: Es müssen Strings geschrieben werden!

- Konfigurationsdateien können ebenso einfach gelesen werden, an die Werte kommt man mit dem `get()` Befehl der `SafeConfigParser` Klasse

```
import ConfigParser

c = ConfigParser.SafeConfigParser()
c.read('setup.ini')
print c.sections()

m1 = c.get('Parameter', 'm1')
m2 = c.get('Parameter', 'm2')

print m1
print m2
```

- Achtung: Auch hier werden die Werte als Strings gelesen und müssen nach Bedarf konvertiert werden!



Links:

- <http://pyqt.sourceforge.net/Docs/PyQt4/qtcore.html>
- <http://pyqt.sourceforge.net/Docs/PyQt4/qtgui.html>
- <http://docs.python.org/2/library/configparser.html>