TECHNISCHE UNIVERSITÄT DRESDEN

python™

Carsten Knoll

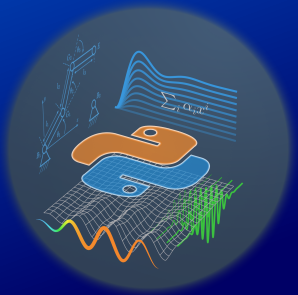
Chair of Fundamentals of Electrical Engineering

Python for Engineers
Pythonkurs für Ingenieur:innen

GUI Programming with PyQT
GUI Programmierung mit PyQT

Dresden (Online), 2024-01-09

https://tu-dresden.de/pythonkurs
https://python-fuer-ingenieure.de



“GUI” - What does it mean?

- GUI ≙ **g**raphical **u**ser **i**nterface
- ease of use, especially when program is developed for third parties
- usability (ISO 9241-11)
- multiple possibilities of interaction (mouse, gestures, etc)
- control of parallel program sequences by the user

- GUI toolkits for Python:
  - Tkinter, PyGtk, wxPython, …
  - PySide
  - PyQt4 (old)
  - PyQt5 (current, used here)
- objectives:
  - get acquainted with the basic elements and simple graphical interface

  → enable simulation control of the trolley-system (last exercises)

Background on Qt

- Engl. pronounciation like “cute”, German: mostly „Kuh-Tee“
- is extensive GUI library (C++)
- there is an automatically generated Python binding called PyQt
  - there is **no** separate reference doc for PyQt

  $\rightarrow$ C++ documentation is authoritative

  $\Rightarrow$ naming convention (e.g. `CamelCase` for methods) is rather uncommon for Python

Basics

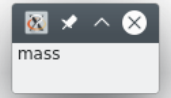
- central concept in GUI programming: **widget**
  - word origin: **wi**ndow + ga**dget**
  - original meaning: control element of a window
    e.g.: button, scroll bar, input field, …
  - collections of widgets result in a new widget e.g.: dialog box with two buttons “OK” and “Cancel”
- main part of GUI programming:
  - control creation, placement, styling and interaction of widgets

Placement (1)

window with text label:

Listing: example-code/gui-example1.py

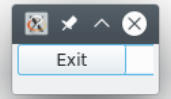
```
import PyQt5.QtWidgets as QtWidgets

app = QtWidgets.QApplication([])
dialog = QtWidgets.QDialog()
mass_label = QtWidgets.QLabel('mass', dialog)
dialog.exec()
```



add an input field and a button:

Listing: example-code/gui-example2.py (7-10)

```
# ... like gui-example1.py + two new widgets
mass_edit = QtWidgets.QLineEdit('2.5', dialog)
exit_button = QtWidgets.QPushButton('Exit', dialog)
dialog.exec()
```



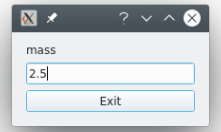
- problem: positioning one above the other → button hides the input field
- solution: define explict layout

Placement (2)

- layouts automatically adjust size and arrangement of widgets
- types: vertical, horizontal, grid
- here: vertical `QVBoxLayout` (elements are “stacked” vertically below each other)

Listing: example-code/gui-example3.py (11-16)

```
layout.addWidget(mass_label)
layout.addWidget(mass_edit)
layout.addWidget(exit_button)
dialog.setLayout(layout)
dialog.exec()
```



Placement (3)

create second `Label` and `LineEdit`:

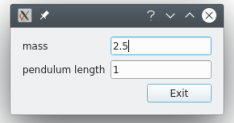
Listing: example-code/gui-example4.py (11-12)

```
len_label = QtWidgets.QLabel('pendulum length', dialog)
len_edit = QtWidgets.QLineEdit('1', dialog)
```

gridlayout:

Listing: example-code/gui-example4.py (14-19)

```
layout = QtWidgets.QGridLayout()
layout.addWidget(mass_label, 0, 0)  # widget, row, column
layout.addWidget(mass_edit, 0, 1)
layout.addWidget(len_label, 1, 0)
layout.addWidget(len_edit, 1, 1)
layout.addWidget(exit_button, 2, 1, QtCore.Qt.AlignRight)
```



- last line: adjust alignment within layout
- more info: https://doc.qt.io/qt-5/qgridlayout.html
  - note: C++ doc → take polymorphism into account
    (i. e. method names appear multiple times with different signatures)

Data Processing and Buttons

- get value from `LineEdit` and set changed value (for demonstration)

Listing: example-code/gui-example4.py (21-28)

```
dialog.exec() # run dialog for the first time

m = float(mass_edit.text())
# write text to command line
print("mass input in the dialog:", m)

mass_edit.setText(str(m*2))
dialog.exec() # run dialog for the second time
```

- customize input field: alignment and input filter:
  → with the `QDoubleValidator` only numbers are allowed as input

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

- connect button with the **function object** `dialog.close` (without brackets):

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

Buttons (2)

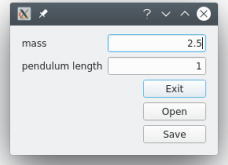
- connect buttons with custom functions (e.g. loading/opening a file)

Listing: example-code/gui-example5.py (24-31)

```
def openfile():
    path, type_filter = QtWidgets.QFileDialog.getOpenFileName()
    print(path, type_filter)

def savefile():
    path, type_filter = QtWidgets.QFileDialog.getSaveFileName()
    print(path, type_filter)
```

Listing: example-code/gui-example5.py (32-40)

```
open_button = QtWidgets.QPushButton('Open', dialog)
save_button = QtWidgets.QPushButton('Save', dialog)

layout.addWidget(open_button, 3, 1, QtCore.Qt.AlignRight)
layout.addWidget(save_button, 4, 1, QtCore.Qt.AlignRight)

open_button.clicked.connect(openfile)
save_button.clicked.connect(savefile)
exit_button.clicked.connect(dialog.close)
```



Create Configuration Files

- module `configparser`: part of Python standard library

Listing: example-code/cfgparser-example1.py

```
import configparser

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

with open('parameters.ini', 'w') as fid:
    c.write(fid)
```

- sections (appear in square brackets in the file) and the key-value pairs can be created
- attention: data type for writing: `str`

Read Configuration Files

- reading configuration files: just as simple
- value: retrieve via `get()`-method of `SafeConfigParser`-class:

Listing: example-code/cfgparser-example2.py

```
import configparser

c = configparser.SafeConfigParser()
c.read('parameters.ini')
print(c.sections())

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

print(m1)
print(m2)
```

- attention: data type of reading result: `str` → convert if needed

TOML – A Modern Configuration File Format

advantages of `.ini`-format:

- simple
- established since decades

disadvantages of `.ini`

- only string-values ($\rightarrow$ conversion neccessary)
- difficult to store complex structures (lists, dicts, ...)
- no standard (subtle differences between implementations)

$\exists$ many formats to store data as text e.g. JSON, YAML, ...

rising star with best advantages-to-disadvantages-ratio: TOML (Tom's Obvious, Minimal Language)

- `tomllib`: part of Python standard library since 3.11
- `https://realpython.com/python-toml/`
- `https://learnxinyminutes.com/docs/toml/`

Links

Links:

- PyQt5 overview:
  http://pyqt.sourceforge.net/Docs/PyQt5/index.html
- PyQt5 modules:
  http://pyqt.sourceforge.net/Docs/PyQt5/modules.html
- PyQt5 widgets module (most important module):
  http://pyqt.sourceforge.net/Docs/PyQt5/QtWidgets.html

  (references C++ docs https://doc.qt.io/qt-5/qtwidgets-module.html)

- several short Qt5 tutorials:
  https://pythonspot.com/en/gui/
- configparser module reference:
  https://docs.python.org/3/library/configparser.html