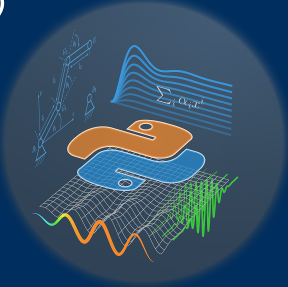
TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Electrical and Computer Engineering Institute of Control Theory

PYTHON FOR ENGINEERS
PYTHONKURS FÜR INGENIEUR:INNEN

GUI Programming with PyQT (Part 2)
GUI Programmierung mit PyQT (Teil 2)



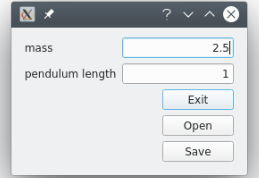
Slides: Sebastian Voigt, Carsten Knoll

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

Dresden, 2023-01-20

Review of Part 1

- “widget” ≙ rectangular area on the screen
  - many widgets serve as control elements (buttons, etc.)
- layout: adjusts the size and arrangement of widgets dynamically
- types of layouts: horizontal, vertical, grid
- widgets and layouts are in parent-child relationships to each other
- so far: application as dialog window (use of the `QDialog` class)



mass: 2.5

pendulum length: 1

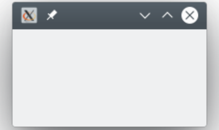
But: many elements of graphical user interfaces are not available when using `QDialog`

Main Window for Applications

For “real” applications, `QMainWindow` is used:

Listing: example-code/main-example1.py

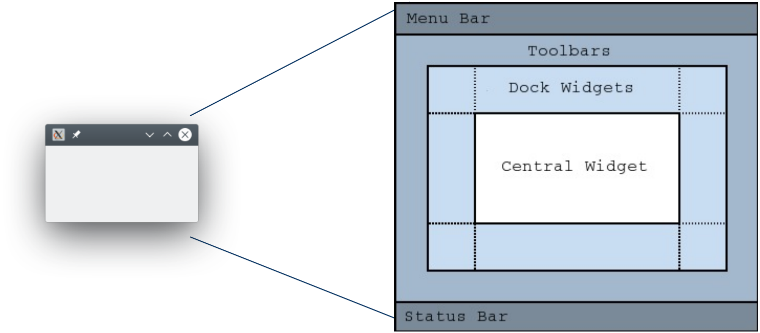
```
import PyQt5.QtWidgets as QtWidgets


class Gui(QtWidgets.QMainWindow):
    """
    Own class (derived from QMainWindow).
    This class (yet) does not do anything.
    """

    def __init__(self):
        # call the "constructor" of the base-class
        QtWidgets.QMainWindow.__init__(self)


app = QtWidgets.QApplication([])

gui = Gui()  # create an instance of the new class
gui.show()
app.exec_()
```



TECHNISCHE
UNIVERSITÄT
DRESDEN

QMainWindow

`QMainWindow` already provides placeholder areas for menus, toolbars, etc.:



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

Menus

- menus can be created directly through the menu bar of QMainWindow:

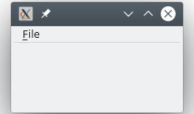
Listing: example-code/main-example2.py (14-15)

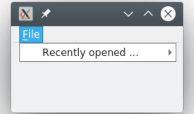
```
        # self.menuBar is a method of the base class
        self.menu_file = self.menuBar().addMenu("&File")
```

- menus can also be nested:

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

- menus only define the structure, not clickable entries
- the `&` in the name string defines shortcut (Alt+F → opens file menu)





Actions

- instances of the `QAction` class can appear in different places: menu entry, button, key combination, ...
- define action to end the program:

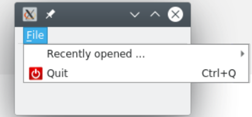
Listing: example-code/main-example3.py (17-20)

```
        self.menu_recent = self.menu_file.addMenu("Recently opened ...")

        self.act_exit = QtWidgets.QAction(self)
        self.act_exit.setText("Quit")
```

- actions represent an abstract interaction possibility with the user
- so far: `self.act_exit` only created; still needs to be added to the menu (and get associated with a shortcut):

Listing: example-code/main-example3.py (22-23)

```
        self.menu_file.addAction(self.act_exit)
        self.act_exit.setShortcut("Ctrl+Q")
```



Actions (2)

- when actions are “triggered” → function can be executed

Listing: example-code/main-example3.py (25)

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

- actions can also appear in toolbars; create a new toolbar:

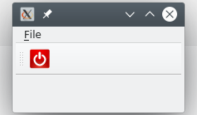
Listing: example-code/main-example4.py (28-30)

```
        self.toolbar = QtWidgets.QToolBar("File")
        self.toolbar.setIconSize(QtCore.QSize(24, 24))
        self.addToolBar(self.toolbar)
```

- ...and add the action:

Listing: example-code/main-example4.py (32)

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

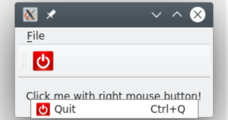


Actions (3)

- actions can appear in the context menu:

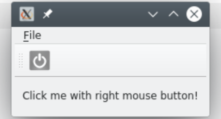
Listing: example-code/main-example5.py (32-41)

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

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

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

        self.label = QtWidgets.QLabel("Click me with right mouse button!")
        self.label.setContextMenuPolicy(QtCore.Qt.ActionsContextMenu)
        self.label.addAction(self.act_exit)
```



- actions can be deactivated:

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



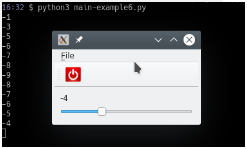
Signals and Slots

- communication mechanism within the application:
  - widgets emit “signals” (e.g. when button is clicked)
  - functions/methods (so called “slots”) can react to them
  - requirement: corresponding signal has been assigned to corresponding slot (with `connect`)

- example: output the value of a slider (via QT label and on command line):

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

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

        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)
```



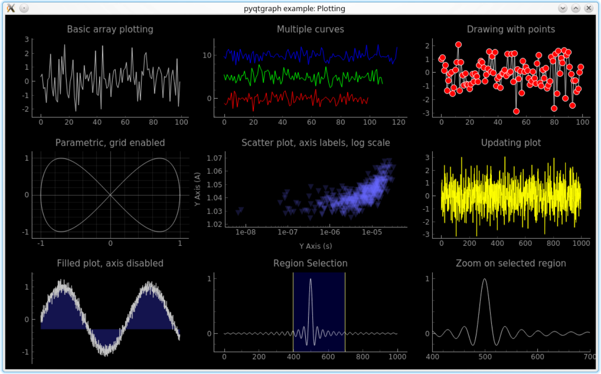
```
16:32 $ python3 main-example6.py
-1
-3
-5
-6
-7
-8
-9
-8
-7
-6
-5
-4
```

PyQtGraph

- plot library (≈ matplotlib), integrates well with Qt applications + much faster

→ advantageous for *interactive* plotting applications

- Disadvantage: additional learning effort required
- installation: `pip install pyqtgraph` or `pip install --user pyqtgraph`
- demo display: `python -m pyqtgraph.examples`



Links

- PyQt5 Overview:
  `http://pyqt.sourceforge.net/Docs/PyQt5/index.html`
- PyQt5 Module:
  `http://pyqt.sourceforge.net/Docs/PyQt5/modules.html`
- PyQt5 Widgets-Modul (most important module):
  `http://pyqt.sourceforge.net/Docs/PyQt5/QtWidgets.html`
- PyQtGraph project:
  `https://github.com/pyqtgraph/pyqtgraph`