PyQt5 – introduction

Grzegorz Bokota

May 10, 2017

Event-driven programming

Event-driven programming – overview

This is programming paradigm where programmer do not control data flow in program. Person, who wrote program prepare solution for every event which should be handle.

Main application:

- GUI
- Network programming (server client architecture)

From some point of view Object programming is version off Event-driven programing

PyQt5 – base information

Qt

- Set of multiplatform libraries
- Gain access for component for creating GUI, server and console applications
- Wrote in C++
- Has bindings in other languages
- Current version: 5
- Free (GPL, LGPL) for non commercial use.

PyQt

- Available under GNU GPL v3 and Riverbank Commercial License (More restrictive than Qt)
- There are version for Qt4 and Qt5 (strong suggest for Qt5 version)
- It is integrated with matplotlib plot engine
- Mainly use python builtin types.
- It is easy to use Qt documentation during use PyQt.

‘Event‘ type

Event – interaction connected with this object:

- Window show, close, resize, ...
- Mouse move

Signal – some event happen for other object:

- Button clicked
- Item selected
- **Custom signal**

Introduction to PyQt programing

QApplication

Main class needed for execute application. It contains *mainloop* which looks for events and execute functions connected with this events

Simplest run:

```
myApp = QApplication(sys.argv)
wind = MainWindow(...) # custom class
wind.show()
myApp.exec_()
```

QMainWindow

Base class suggested for application main window. Add place for main menu and statusbar

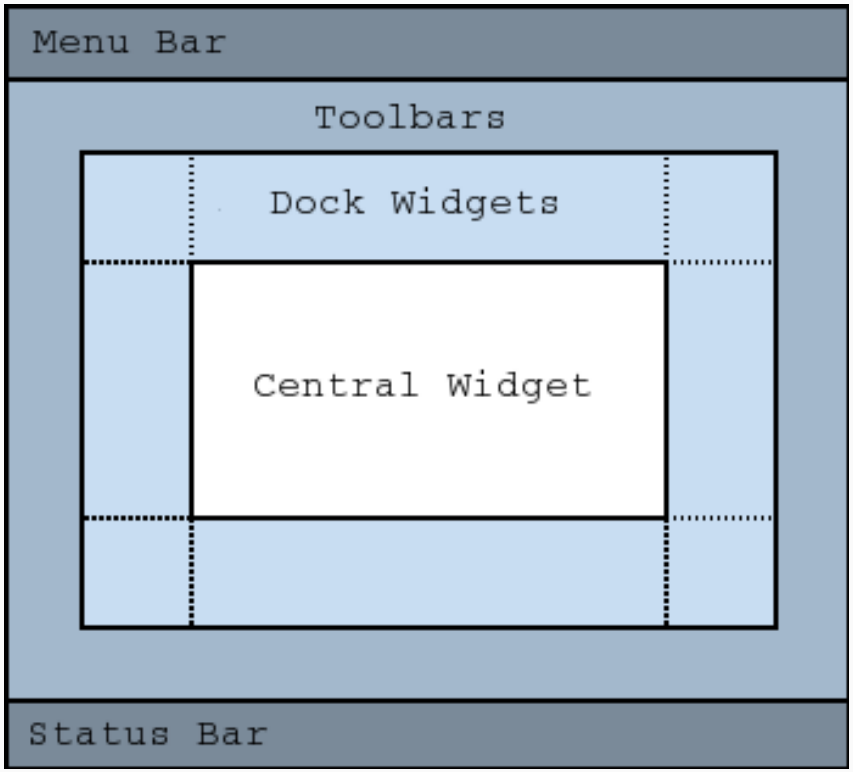

Figure 1: QMainWindow

QDialog

Class for all dialog windows. Dialog window block all other windows until is active.

- QFileDialog
- QInputDialog
- QMessageBox
- ...

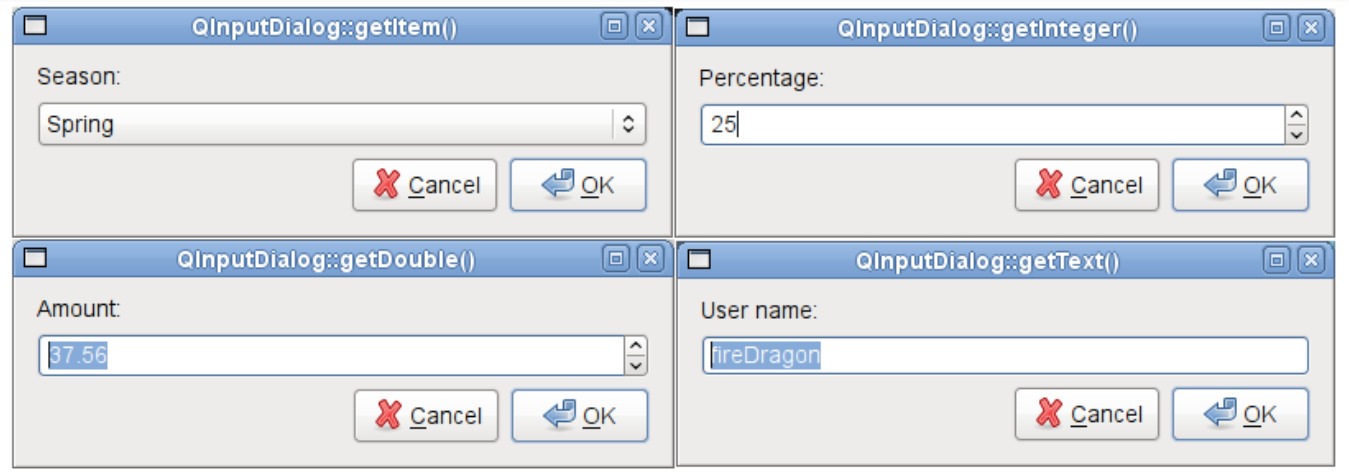

Figure 2: QInputDialog

Other

- QWidget – Base for all other
- QPushButton
- QCheckBox
- QTableWidget
- QProgressBar
- QLineEdit
- QSpinBox
- QLabel – text can be formatted with simple HTML

Layout

Of course we can set position of each element manually using `ob.move(const QPoint&)`. But in this case we need update it manually after each change. Easier would be if framework will do it for us.

- QGridLayout
- QHBoxLayout
- QVBoxLayout

```
main_layout = QVBoxLayout()
btn_lay = QHBoxLayout()
btn_lay.addWidget(self.new_matrix_btn)
btn_lay.addWidget(self.save_btn)
btn_lay.addWidget(self.load_btn)
main_layout.addLayout(btn_lay)
```

Matplotlib integration

Import:

```
from matplotlib.backends.backend_qt5agg import
    FigureCanvasQTAgg as FigureCanvas
```

Usage:

```
fig = pyplot.figure(figsize=(6, 6), dpi=100, frameon=False,
    facecolor='1.0', edgecolor='w', tight_layout=True)
self.figure_canvas = FigureCanvas(fig)
self.fig_num = fig.number
```

Reuse:

```
fig = pyplot.figure(self.fig_num)
pyplot.imshow(self.matrix, cmap=self.cmap)
fig.canvas.draw()
```

Tasks

Tasks

1. Add colormap choose with `QComboBox`
2. Add checkbox which allow break symmetry