Outline

- Controlling a single device
- Saving the data
- Controlling devices via a Graphical User Interface

Controlling a single device

- VISA compatible device (scope, generator, ...)
  - Low level access : PyVISA, python-vxi11, python-usbtmc, pySerial
  - High level : python-ivi, pyivi, DIY !

- Proprietary interface
  - Call DLL functions using `ctypes`

Exercise 1

- Connect to the Lecroy scope and retrieve its serial number (< 5 lines of code)
  - Hints :
    - Use `vxi11` `'localhost'`
    - The IP address is ~~134.157.91.252~~
    - Read the Lecroy manual p.156

Exercise 2

- Fetch a waveform from the scope (< 5 lines of code)
  - Hints :
    - Read the Lecroy manual p.54
    - Use the `ask_raw` method

Exercise 3

- Write a function to decode the returned string and create two numpy arrays time & voltage (< 20 lines)
  - Hints :
    - Read the Lecroy manual p.282
    - Use the numpy `fromstring` function
    - Look for the `WAVEDESC` string that defines the beginning of the file
    - Data start at position 346

Exercise 4

- Write a `LecroyScope` class with a `fetchwaveform` method that returns two numpy arrays (< 30 lines)

- Hints :
  - Inherit from `vxi11.Instrument`

Data Saving

- File based solutions (easy)
  - Easy to copy (and be messy). Organize your files !
  - Install a good backup from the beginning
  - No concurrent access
- Database server (complex)
  - Difficult to copy the data
  - Fast search
  - Concurrent access allowed
  - Instant automatic backup (if well configured)

Data Saving

- File based solutions (easy)
  - Python cPickle + Zip file
  - HDF5 format with pyTables

- Database server (complex)
  - SQL approach (old)
  - Try MongoDB instead (pyMongo)

Save everything you can !

Exercise 5

- Write a script to acquire one trace and save it as a pickle file using cPickle (<15 lines).
  - Hints :
    - Pack the data in a dictionary and pickle the dictionary
    - Use the `with` statement

Exercise 6

- Write a script to acquire one waveform and save it as a pickle file using cPickle (<15 lines).
  - Hints :
    - Use `cPickle.dumps()`
    - Use `zipfile.writestr()`

Exercise 7

- Write a script to acquire 10 waveforms and save them in a HDF5 table
  - Hints :
    - Read the PyTables tutorial ...

GUI with PyQt

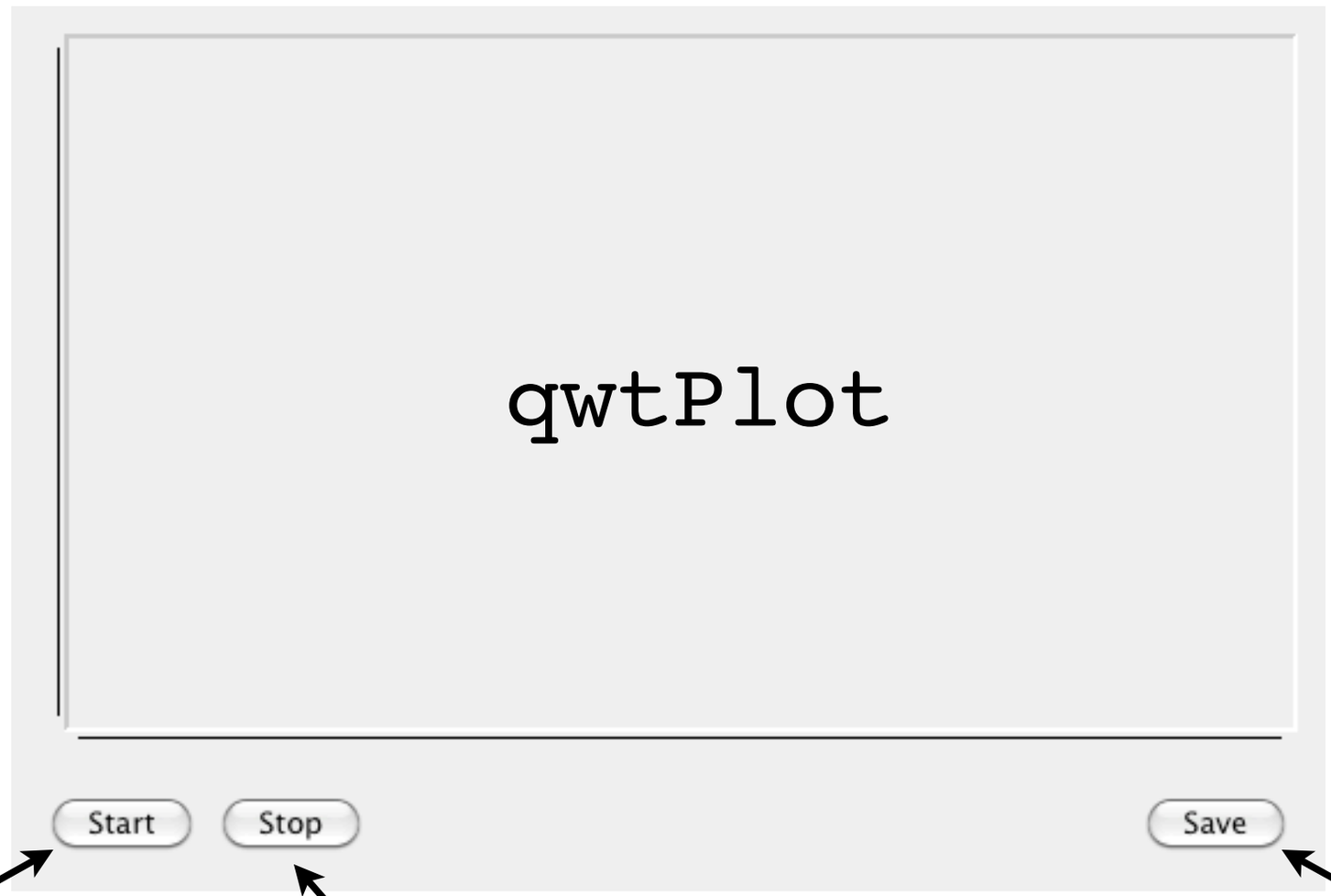
- Specify your application !
- Think about the user...
- Create a GUI with Qt-designer and converts it to python code with `pyuic`
- Draw all possible event chains that could happen

The livescope application

- Specifications :
  - Visualize the data from the scope in real time
  - Save the last trace
- Interface :
  - One Start button / One Stop button
  - One plot window
  - One Save button

Exercise 8

- Create the following GUI with designer and convert the `ui` file to the python file `livescope_ui.py`



Minimal PyQt code

See `livescope_00.py`

```
#!/usr/bin/env python
import sys
from PyQt4 import Qwt5, QtCore, QtGui, Qt
from livescope_ui import Ui_MainWindow

class MainWindow(QtGui.QMainWindow):
    def __init__(self):
        # Create the main window
        super(MainWindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

    @QtCore.pyqtSignature("")
    def on_pushButton_Start_clicked(self):
        print 'Clic !'

if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    mainWin = MainWindow()
    mainWin.show()
    sys.exit(app.exec_())
```

Import GUI

Add methods to catch events

- Launch Qt event loop
- Create the main window

Exercise 9

- Fetch one waveform and plot it when the Start button is pressed (< 40 lines)
  - Hints :
    - Create a `Qwt5.QwtPlotCurve`
    - Attach it to the plot
    - Update the curve data

Exercise 10

- Implement the Start/Stop feature (< 70 lines)
  - Hints :
    - Create a `QThread`
    - Connect the finished event of the thread to a callback method of your application

Exercise 10

- Implement the Start/Stop feature (< 70 lines)
  - Hints :
    - Create a `QThread`

```
class AcquisitionThread(QtCore.QThread):
    def __init__(self, fun):
        super(AcquisitionThread, self).__init__()
        self.fun = fun
    def run(self):
        self.fun()
```

Multithreading & Multiprocessing

- Multithreading allows you to keep your application responsive (no hourglass or turning wheel)
- Only one thread at a time should use the CPU :
  - One thread waiting for acquisition to finish
  - One thread analyzing data (CPU consuming)
  - Main thread running the Qt event loop

- Multitasking in Python is done via multiprocessing
  - Beyond the scope of this course...

Exercise 11

- Implement the Save function : save the acquired waveform and the time of acquisition in a dictionary using cPickle (< 15 lines)

- Hints :
  - Use `QtGui.QFileDialog.getSaveFileName()`

Some useful resources

- This course :
  http://nbviewer.ipython.org/gist/jesteve/8851946

- Python in the lab :
  http://python-in-the-lab.blogspot.fr/p/blog-page.html

- One paper :
  http://gael-varoquaux.info/physics/agile_computer_control_of_a_complex_experiment.pdf

- Labscript (cold atom experiment program control) :
  http://labscriptsuite.org/

- PyTango (used for accelerators but can be adapted to control small experiments) :
  https://www.tango-controls.org/static/PyTango/latest/doc/html/