Lab1 Arduino 33 IoT processor and PyQt5

This lab will learn you working with the Arduino nano 33 IoT processor boards.
There are six tasks to complete where you will explore:

- The Arduino programming environment.
- Using digitial IO pins, serial communication and real time clock.
- Connecting the nano to a Python program.
- Creating a PyQt5 graphical user interface.
- Plotting measured data from Arduino sensor using PyQt5 and Matplotlib

Introduction Arduino Nano33 IOT
The nano33 is a small powerful microcontroller designed to act as a smart node in an Internet of Things network. It carries a 32 bit ARM microprocessor, a Bluetooth Low energy (BLE) module and a Wifi imodule. The Nano features also an IMU inertial measurement module and a RTC (Real Time Clock) module).
An in-depth description can be found at the www.arduino.cc website in the section: products/arduino –nano-33-iot.

Software development
Creating applications for the Arduino Nano 33 IOT are developed with the Arduino IDE. In this course we use the Arduino IDE V2 which can be downloaded from the www.arduino.cc website and follow the instructions to install the Arduino IDE V2.

Task 1 – Basic IO operation and serial communication.

By doing this task you will learn to program a single input/output pin and communicate with the Arduino IDE.

- Digital IO (see Language reference section: Functions/Digital IO)
- Serial communication (see Language reference: section Functions/ Communication)
- Strings (see Language reference section: Variables/Data Types)
  Create a program which switches the on-board LED (color yellow) on/off by commands send via the serial/USB connection between your BYOD laptop and the Arduino nano. The commands are entered in the serial monitor console part of the Arduinp IDE. The nano will respond with a response message after receiving a command.
  Operate the LED with the following commands:

Description	Command	response
Switch led on	**On**	**Led on**
Switch led off	**Off**	**Led off**
Return led status	**status**	Return status
Unknown command		Unknown command

Prerequisites:
Do not hardcode the command strings, but use the **String** class to define the command strings.. Learning this class will make handling strings easier when needed in the next tasks and project.
Use the template **lab1_template.ino** as a starting point.
Hint:
At the Arduino website in the section language reference you can find information how to operate the digital IO, serial communication and strings.
Save your Arduino sketch file as **lab1_task1.ino** .

Task 2 - Blinking LED and the Real time clock module.

In this task we will use the **RTC (Real Time Clock)** module to blink the on-board LED.
Extend your program from Task 1 to blink the LED every one second. (on period and off period both 1 second)
You have to use the internal RTC and you are not allowed to use the build-in time routines from the Arduino library.
Add the blink command as shown in the table below:

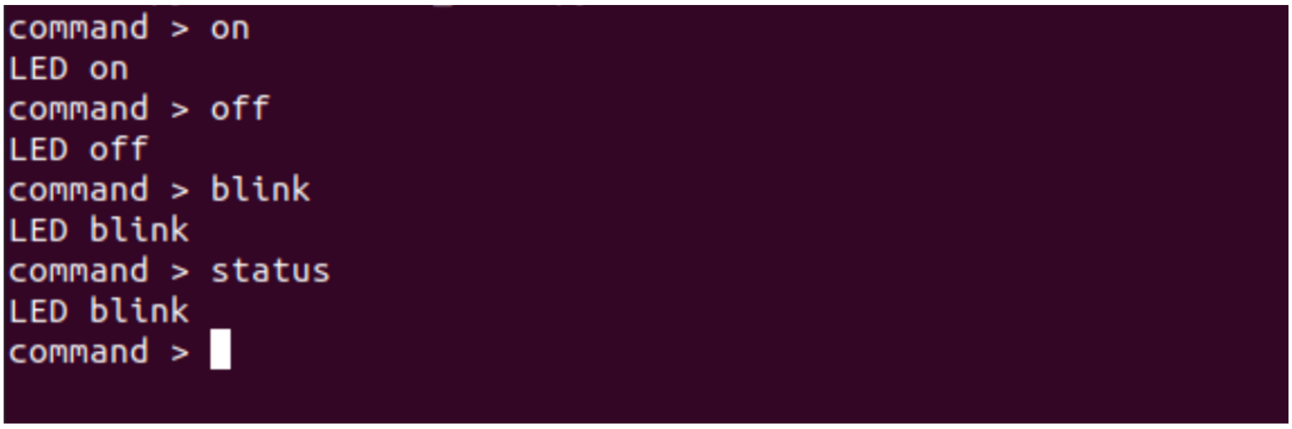
Description	Command	response
Switch led on	**On**	**Led on**
Switch led off	**Off**	**Led off**
Return led status	**status**	Return status
Blink the LED	**Blink**	**Led blink**
Unknown command		Unknown command

Save your program as **lab1_task2.ino .**

Task 3 - Serial connection to Python

In this task you will create a Python program to communicate with the Nano. To access the serial/usb port, use the PySerial library. Install this library by executing the following command: **pip3 install pyserial** .

Create a Python program to operate the on-board LED as described in task Below an example off the output:



```
command > on
LED on
command > off
LED off
command > blink
LED blink
command > status
LED blink
command > █
```

Hint:
Use Linux command **dmesg** to find the serial port number connected to the nano.
Submit your program as **serial_nano.py.**

Task 4 - Building a GUI with PyQt5

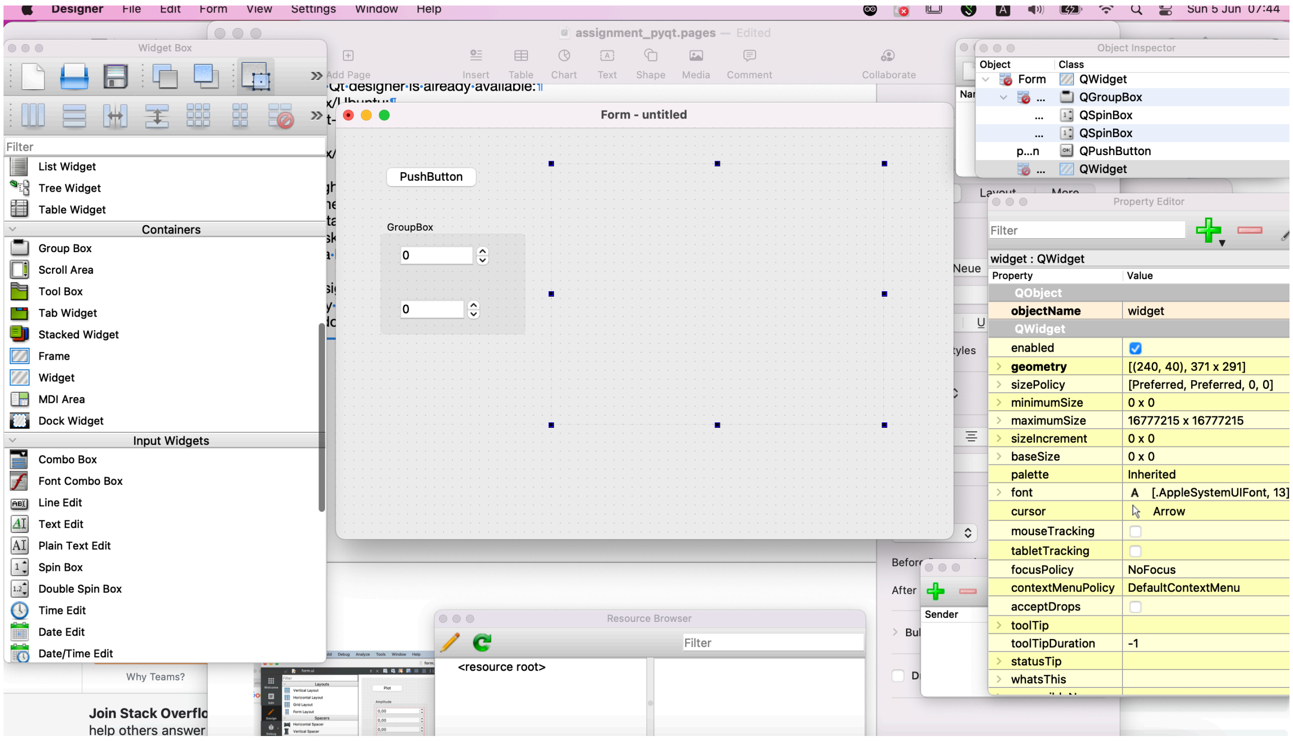
In this task you will extend the Python application from lab1_task3 to interact with the nano via a GUI (Graphical User Interface). The GUI is created with the PyQt5 package.

There are several ways to create a Qt user interface:
1. Build it programmatically
2. Build it with Qt Designer (Qt Creator) and generate from the ".ui" file a Python file using pyuic5
3. Build it with Qt Designer and load/compile the ".ui" file directly inside your Python program.

We will use the second method, although the other methods can be explored too if you like. There are many sources on internet which explain how to do this.
First proceed with checking if all libraries and tools are installed on your system.

installation
In Linux/Mac check if PyQt5 has been installed together with an Anaconda installation (we assume this is the case if you have done the Python assignments of "Programmeertalen".
Type in a console window 'which python' and check if it's coming from an Anaconda installation. If not install Anaconda Python3 from:
 https://www.anaconda.com/products/individual
Check if PyQt5 has been installed too by opening a python shell and type: "import PyQt5". If not, install it using the conda installer (conda install pyqt).

Check if Qt designer is already available:
For Linux/Ubuntu:



Object	Class
Form	QWidget
...	QGroupBox
...	QSpinBox
...	QSpinBox
p...n	QPushButton
...	QWidget

widget : QWidget

Property	Value
QObject	
objectName	widget
QWidget	
enabled	✓
geometry	[(240, 40), 371 x 291]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
maximumSize	16777215 x 16777215
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [.AppleSystemUIFont, 13]
cursor	Arrow
mouseTracking	☐
tabletTracking	☐
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	☐
toolTip	
toolTipDuration	-1
statusTip	
whatsThis	

Sudo apt-get install qttools5-dev-tools (and perhaps qttools5-dev).

For Linux/CentOS and Fedora: sudo yum install qt5-designer

Mac: slightly more work. You will need first an Xcode installation (developers environment for Mac, including iPhones). After this proceed with qt5:
brew install qt5
brew cask install qt-creator
(Or use a Linux virtual machine)

Start designer and create a new Qt for Python project with UI file. This will result in an empty "UI" window where you drag user interface elements from the left into this window. It could look like the following:

Part 4.1: a super simple user interface
Put one pushbutton inside the window and look at the yellow window on the right.
Put below the pushbutton a groupbox which is just a container to show that inputs inside the container have a logical relation. Put in the groupbox two normal SpinBoxes. We can use these spinboxes later.
The object on the right we will add later.
You can change the name of the objects and also other characteristics.
Save your user interface as "lab1.ui"

Now we create a python file which loads this user interface, but first we need to generate a python file which contains references we have made in the .ui file.
We will use the pyuic5 executable which is also in the anaconda/bin directory.
Generate the file as follows:

```
pyuic5 lab1.ui > lab1_ui.py
```

Have a look in this file:it has created a Ui_form class with references to all our user interface elements.
Now we create a separate Python file which we can name for example
pyqt_lab1.py

```
import sys
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import *

from lab1_ui import *

class Lab1(QMainWindow):
     def __init__(self, *args):
         QMainWindow.__init__(self)

         self.ui = Ui_Form()
        self.ui.setupUi(self)
         self.setWindowTitle("arduino_sensors”)

if __name__ == "__main__":
     app = QApplication([])
     form = Lab1()
     form.show()
     sys.exit(app.exec_())
```

This program will not do very much of course. So, now connect a clicked event to the button:
In init: self.ui.pushButton.clicked.connect(self.mybuttonfunction)
And define the function self.mybuttonfunction inside the Lab1 class. Check if it works (just print some text). We assume that you have become a bit familiar with object oriented programming during the Programmeertalen course. Have you seen the QMainWindow after the class Lab1 definition and the QMainWindow.__init__ call? Why is that? If you don’t know look it up on internet.

Part 4.2: inserting a Matplotlib QWidget element
Before inserting the streaming sensor data of the Arduino into a plot it's a good idea to have some fake data and use some widgets you can use later for a different purpose. Now add the following elements to your user interface:

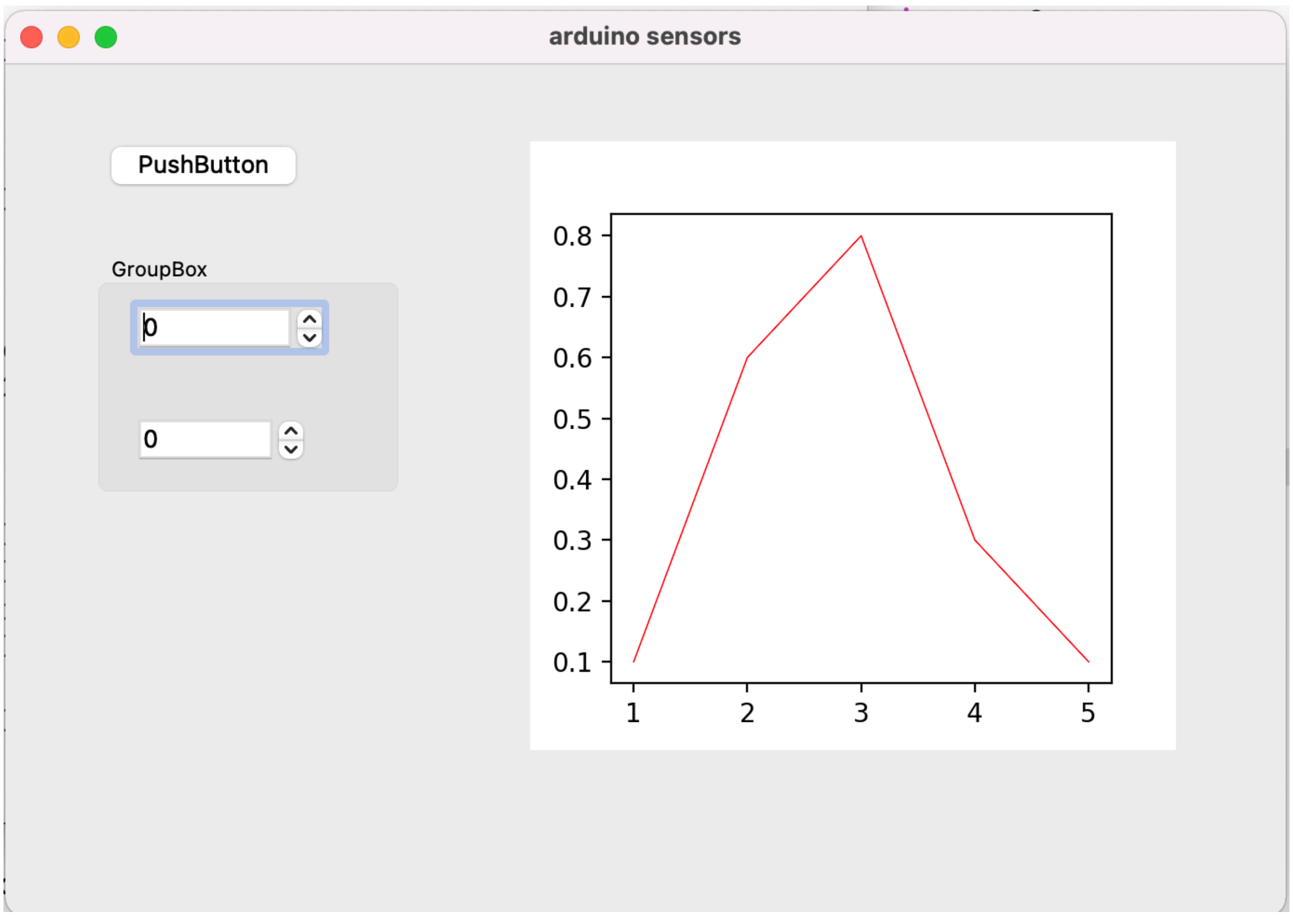
Add in you lab1.ui on the right a widget container: it will look empty (only when selected you will see some dark selection points). Call the object MplWidget.
Right click on the MplWidget and select the promote option (with the "dots"). Enter the name of the object in the first field below QWidget: MplWidget and leave the contents of the header file as it is (mplwidget.h). Click “Add” and “Promote” !!!
Save you user file.
The MplWidget will contain our Matplotlib graph and needs a separate file with the name mplwidget.py. An example is available on canvas. It contains a class function for the matplotlib graph. Put it in the same directory as the other files.

You will need to generate again the lab1_ui.py file using pyuic5 to have the new widget references loaded. You will see in this file that at the bottom a line has been added which imports mplwidget.py and the references to the widget is also added in the Ui_Form class.

We can now refer to this widget inside the Lab1 class with:
self.MplWidget as a Matplotlib object. You will need a bit more Matplotlib imports and some more references for your plot widget:

```
import matplotlib
matplotlib.use("Qt5Agg")
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
NavigationToolbar
from matplotlib.figure import Figure
```

Somewhere in the Lab1 python class (the button function is a logical place):



```
        self.ui.widget.canvas.axes.clear()
        self.ui.widget.canvas.axes.plot(self.x, self.y,'r',
 linewidth=0.5)
        self.ui.widget.canvas.draw()
```

You will need some data for self.x and self.y (for example a random data array for self.y and some counter for self.x).

As you run the application it should look like the figure above.

part 4.3 Adding a timer event
Before reading the streaming data and plotting the data into a the matplotlib widget we add a timer event to our user interface. When the timer is enabled we should execute a timer event function which adds new data to our plot window. When we have reached the last x axis value we should start over and show the new data with modified x axis.
See for timer events: https://pythonpyqt.com/qtimer/
Enable / disable the timer with the pushbutton and write your own timer event function where you plot some random data. Take care that you shift the lists of data.
You can combine the spinboxes to the interval of the timerevent and the maximum value of the x axis of the plot.

Submit this file as pyqt_lab1.py

Task 5: Plotting the IMU data

In this task the x,y,z gravitional forces read from the IMU module are plotted in a 2d plotdiagram.

Arduino side
After receiving a command from the serial port, the gravitational data is read at an interval of 0.5 seconds for a period of 15 seconds and is send via the serial interface.

Documentation: https://www.arduino.cc/reference/en/libraries/arduino_lsm6ds3/

Python side
Create a PyQt program which collects and displays the IMU data received from the nano.
- Plot the data in matplotlib widget
- The time is plotted along the x-axis
- The gravitional forces along the y-axis.
  Calculate for the X,Y and Z axis the following parameters:
- Mean and Standard deviation.
  Display the result in your user interface.

Submit your PyQt5 program as **pyqt_nano.py**

Question
This question is about the accuracy of the measured signals.
- Hold the nano in a fixed position for the duration of the measurement.
- Do this for 5 different fixed positions.

- Write down the calculated parameters.

  - Compare this with your measurements and write down your observation and conclusion in a text document and save as Lab1_QT4.txt .

For our expert programmers:
Combining streaming data with event handling of an user interface is notorious difficult and always asks for a solution with multithreading. The plotting can occur in a separate thread (reading Arduino and storing the data into queue for example) while a user interface timer event can read the queue and plot the data.

For those who want to reach the maximum points for this lab can extend the program and submit it as **pyqt_multi.py**