# Internet of Things (IoT) technology for hydrological measurements

Crash course at NMAIST Arusha, 11-12 March 2019

Resource persons: Douglas Nyolei and Jan Diels

## 1. Basics of MicroPython

**Useful links:**

Pycom documentation (Lopy4, extension board, MicroPython syntax): https://docs.pycom.io/

Pycom forum: https://forum.pycom.io/

MycroPyhon language reference: https://docs.micropython.org (but some differences with the MicroPython used by Pycom)

**Preparations:**

Install the Pymakr plugin in Atom: https://docs.pycom.io/pymakr/installation/atom.html

**Connecting to the Lopy4 from the Pymakr console in Atom**

Method 1: Through a USB cable between your laptop and the extension board

Method 2: Through a wifi connection (to be used if you do not have an extension board):

Step 1: Make sure the Lopy4 is powered.

Step 2: Look at the list of wireless connections that are available on your computer. The Lopy4 should be visible as a network with the name "lopy4-wlan-78a0" (the last 4 characters will differ between devices).

Step 3: Connect to this "lopy4-wlan-xxxx" network (which means your current wifi connection to the internet is lost!), and provide the following wifi password: www.pycom.io. This will establish the wifi connection (this takes a few seconds!).

Step 4: Chose the command "Project settings " from the "Settings" menu at the top right corner of the Pymakr window. This opens the file 'pymakr.conf' file. Now make sure that the address is as follows: "address": "192.168.4.1" (this is the IP address of the Lopy4). Next save this file (e.g. by closing it and saving it when you get the question if the file needs to be saved).

Step 5: Click the command '"Connect" from the menu at the at the top right corner of the Pymakr window. This should create the connection to the Lopy4. If successful, you will get a command prompt: ">>> ""

A wifi connection is much less stable then a usb cable connection. If the connection gets lost ("Failed to connect ( Connection was reset). Click here to try again"), you may be able to restore the connection by disconnecting from the "lopy4-wlan-xxxx" network, and next connecting to that wifi network again (this takes a few seconds!), followed by the command '"Pymakr > Connect " from the "All commands" menu at the bottom of the VS Code window.

## 2. Submitting MicroPython commands on the REPL console

At the bottom of the Atom window, there is a terminal window (also called Pycom console or REPL console). 'REPL' stands for 'Read Evaluate Print Loop', and is the name given to the interactive MicroPython prompt that is accessible on the Pycom devices. Through this console, you can submit (=execute) MicroPython commands on the Pycom board, and the Pycom board returns results. This is very useful for trying new code syntax before adding it to your scrips.

Task 1: Please try the examples yourself (experiment with variable assignment and use of operators)

**Variable assignment**

In Python, one can assign values to variables and use them in operations as follows:

```
>>> a=3
>>> b=4
>>> a/b
0.75
```

The variable type is assigned automatically depending on the assignment. With the function 'type(object)' we can get the type of the object:

```
>>> type(a)
<class 'int'>
>>> type(a/b)
<class 'float'>
>>> c='hello'
>>> type(c)
<class 'str'
```

So the variable 'a' is an integer, the result of the division 'a/b' is a float (a real number), while c is a string variable. This just follows from the values assigned to the variables.

**Use of operators**

Arithmetic operations are performed with the operators +, -, *, /, **:

```
>>> side=4
>>> CubeVolume=side**3          Raise to the power of 3
>>> print(CubeVolume)
64
>>> type(CubeVolume)
<class 'int'>
>>> test=CubeVolume**(1/3)
>>> print(test)
4.0
>>> type(side)
<class 'int'>
>>> type(test)
<class 'float'>
```

## Use of functions

Basic mathematical functions are available through the math module:

```
>>> import math                    The math module first needs to be imported
>>> slopeDegree=30
>>> slopePercent=math.tan(slopeDegree/180*math.pi)*100
>>> print(slopePercent)
57.73503
```

In (Micro)Python, Python code in one module (or *.py file) gains access to the code in another module by the process of importing it. A module is a file containing (Micro)Python definitions and statements. The math module is a build-in module. That means it is part of the interpreter, and this module ('math.py' file) does not have to be uploaded to Pycom board.

The math module provides standard goniometric, exponential, logarithmic and truncation functions (see https://docs.pycom.io/firmwareapi/micropython/math.html ).

A number of functions are build-in and do not require an import: The type(object) and print(variable) functions are build-in functions. See https://docs.pycom.io/firmwareapi/micropython/builtin.html .

```
>>> round(5.14)
5                    The functions 'round' and 'abs' are build-in functions
>>> abs(-5.5)
5.5
```

Apart from the math module, there are another 20 modules available. See https://docs.pycom.io/firmwareapi/micropython/

**Task 2:** Please write and execute the statements that are needed to convert temperature from Farenheit to Celsius, and to convert (soil water) pressure head from kPa to pF units. Please use self-explaining variable names ('airTemperatureFarenheit' and not 'test1'). Round the results to 1 decimal.

The sensors have measured the following:
Air temperature = 72.68 °F; pressure head = -36.1 kPa

The conversion formulas are as follows:
DegreesCelsius = (DegreesFarenheit − 32) × 5/9
PressureHead_cmH2O = PressureHead_kPa*100/g   with  g=acceleration of gravity = 9.806 (m/s)
pF = log10[abs(PressureHead_cmH2O)]

Tip1: with the cursor key (upward, downward arrows) you can scroll through previous commands issued at the REPL console to re-use them or to modify them.

Tip2: if the log10 function is not available, you can use the following property: log10(x)=ln(x)/ln(10)

## 3. Making a Pymakr project

So far MicroPython commands were executed in the REPL console. For more advanced applications, we will write MicroPython scripts that are uploaded onto the Pycom board. Such scripts (main.py and possibly some libraries and other files) are organized in a project.

**Task 3:** Read 'Your first Pymakr project' at https://docs.pycom.io/gettingstarted/programming/first-project.html#your-first-pymakr-project and next upload and run a script that makes the on-board RGB LED flash various colours (red-green-blue in that order).

## 4. Reading the DS18B20 sensor with the on-wire protocol

Some sensors have their own integrated circuits that are programmed to measure or observe something. This is also the case with the inexpensive DS18B20 temperature sensor (price around 6-10€ for a waterproof version with a cable attached). It measures temperature with an accuracy of +/- 0.5 °C. Microprocessors like the lop4 need to use the so called 'one-wire protocol' to communicate with the sensor. So far MicroPython commands were executed in the REPL console. For more advanced applications, we will write MicroPython scripts that are uploaded onto the Pycom board. Such scripts (main.py and possibly some libraries and other files) are organized in a project.

**Task 4:** Connect the DS18X20 sensor (see slides), and next open folder \IoT course\DS18B20TemperatureSensor in Atom. This will open a new Atom session. If you have another Atom session open then the new Atom session will not connect to your lopy4. So first (save and) close your previous Atom sessions. After that your new Atom session should connect automatically to the lopy4 via usb. To read the temperature, now open demo.py, and run this script by clicking the 'Run' command from the pymakr console. You should see the temperature in the pymakr console. Now warm up the sensor with you hand and observe how quickly the sensor reacts to this (repeatedly run demo.py).

Since each temperature measurement requires a sequence of commands (start_conversion, read_temp_async, …) it is more efficient to create a function for that. That is what is done in the script main_new.py in folder \IoT course\DS18B20TemperatureSensor (the main.py file did so far not contain any python commands). This main_new.py script can moreover handle several DS18B20 temperature sensors connected to the same onwire bus (on P10).

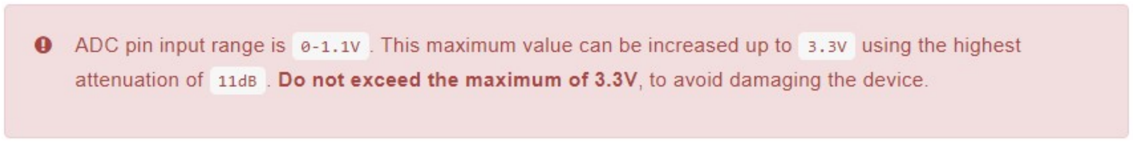
**Task 5:** For this task you need two DS18X20 sensors, so you need to perform this task with a colleague so you have two sensors available. Connect the two sensors to exactly the same pins on the lopy4: red to +3.3V, black to GND, and yellow to P10. Since you can only connect one pin to each slot in the expansion board, you need to use the white breadboard and jumper cables to do this. Next run main_new.py and observe the temperature reading. Are the two sensors giving the same temperature? Check if the readings react to warming them with your hands (one by one).

## 5. Use of the analog-digital converter (ADC) on the LoPy4 to measure voltage (exercise: measure battery voltage)

Pins P13 to P20 can be used to measure a voltage. The lopy4 has a built-in 12-bit analog-digital converter (ADC). It means that an analog signal (an electrical signal having a certain voltage) is converted into an integer number. For a 12-bit ADC that means a number between 0 and 4095 (=$2^{12}$ -1). That number can then be rescaled in the micropython code to a voltage. The following code snippet shows that there are 3 commands needed (circled in red) to measure a voltage.

```
import machine
import pycom
import adcR # module for reading ADC and converting it to voltage

attn=3 # chose attenuation of ADC
# attn=0 (default) is 0 dB (0-1.1V) with internal reference voltage of ADC being (about) 1.1V
# attn=1 is 2.5 dB         (0-1.5V)
# attn=2 is 6 dB           (0-2.2V)
# attn=3 is 11 dB          (0-3.9V)

bits=12 #'Bits' can take integer values between 9 and 12 and selects the number
# of bits of resolution of the ADC. More bits means higher precision

adc = machine.ADC(bits=bits) # create an ADC object and specify the resolution.

apin = adc.channel(pin='P14',attn=attn)   # Create an analog pin on P14, set attenuation. Valid pins
are P13 to P20.

nSamples=10000 # set number of repeated measurements
while True:
    ADCreading,Voltage=adcR.adcRead(apin,nSamples)  # read ADC and convert reading to voltage
    print("ADCreading = %6.4f" % ADCreading, "    Voltage = %8.3f" % Voltage)
```

Please pay attention to the attenuation: you need to select the right attenuation for the voltage that you expect. There is the following important warning in the Pycom documentation:


ADC pin input range is `0-1.1V`. This maximum value can be increased up to `3.3V` using the highest attenuation of `11dB`. **Do not exceed the maximum of 3.3V**, to avoid damaging the device.

For the third command (the actual ADC reading and conversion to voltage), we use a function that resides in the file adcR.py in the library folder.  Please also examine that file so you understand how the reading is done.

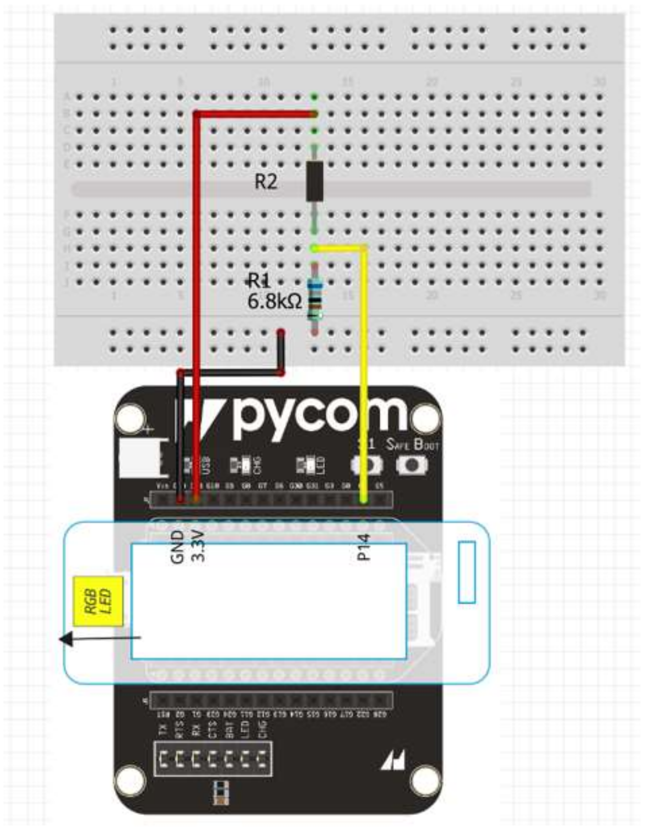
Please note that in this way measurements can be initiated on several pins at the same time (and with different attenuation), as the following code illustrates:

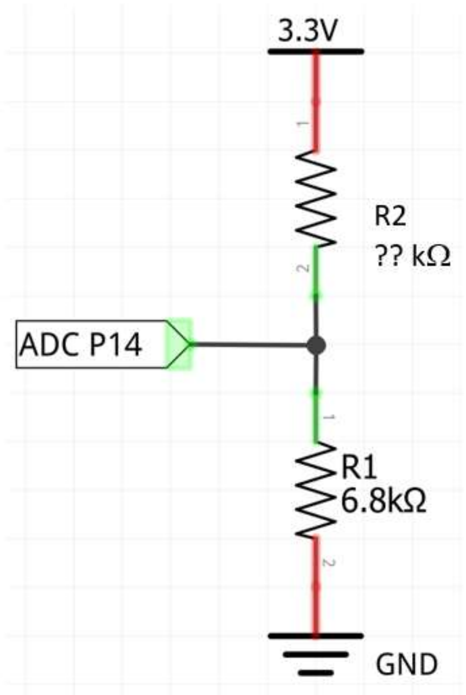
```
import machine
import pycom
import adcR # module for reading ADC and converting it to voltage
adc = machine.ADC(bits=12)  # create an ADC object and specify the resolution.

apin13 = adc.channel(pin='P13',attn=3)   # Create an analog pin on P13, set attenuation to 3.
apin14 = adc.channel(pin='P14',attn=3)   # Create an analog pin on P14, set attenuation to 3.

nSamples=10000 # set number of repeated measurements
while True:
    ADCreading,Voltage=adcR.adcRead(apin13,nSamples)   # read ADC and convert reading to voltage
    print("ADCreading on P13 = %6.4f" % ADCreading, "    Voltage = %8.3f" % Voltage)
    ADCreading,Voltage=adcR.adcRead(apin14,nSamples)   # read ADC and convert reading to voltage
    print("ADCreading on P14 = %6.4f" % ADCreading, "    Voltage = %8.3f" % Voltage)
```

The built-in ADC actually measures voltage by comparing it with an internal voltage reference of about 1.1V. With 'about' we mean that there is some variation between different lopy4's. If you want to increase the accuracy of your voltage measurements, you can measure that internal voltage reference of your lopy4 with a digital multimeter, store that value in the NVRAM (the non-volatile random access memory) of the lopy that you thus calibrated. Next, in the python script where you are using the ADC, you communicate that value to the ADC function with the command 'adc.vref(1123)' (with 1123 being in the internal reference voltage in mV that you measured). All this is indicated in the code in folder " \IoT course\ReadVoltage\ "

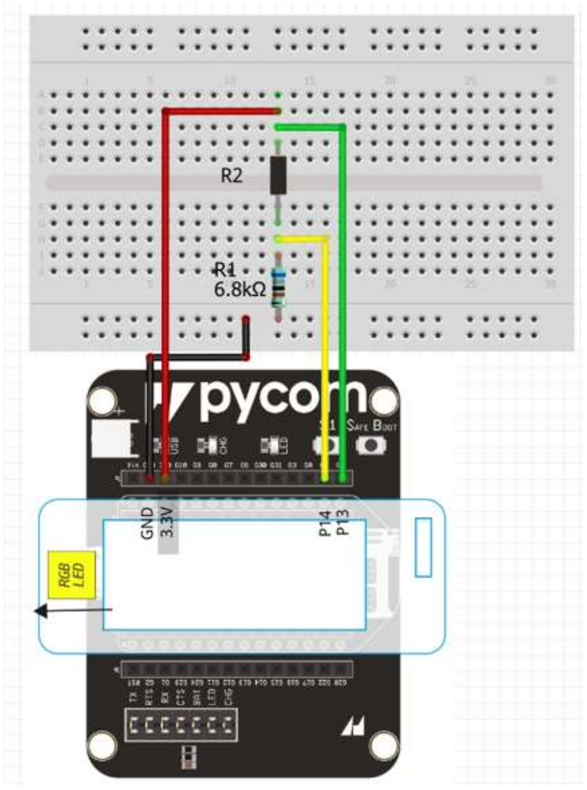
**Task 6:** For this task, you proceed as follows:

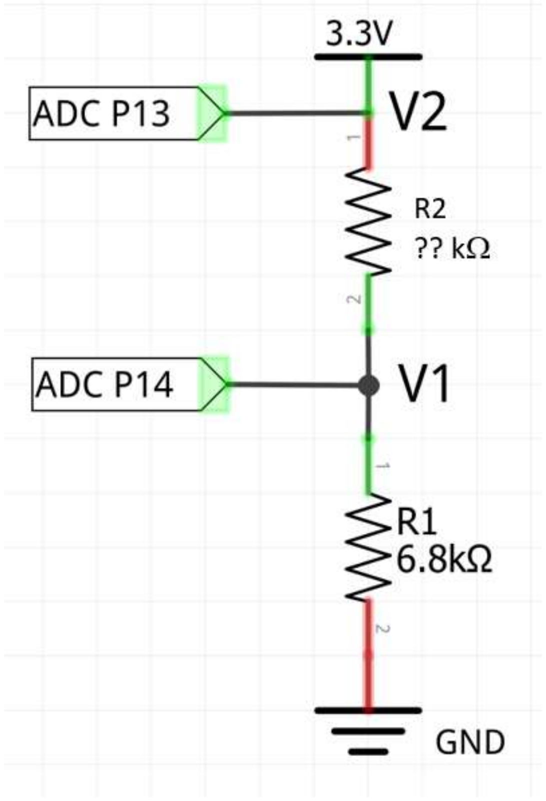
- First open the folder " \IoT course\ReadVoltage\" in atom and examine the code carefully. You will later use the main.py script to do a voltage measurement.
- Disconnect the expansion board from your laptop (disconnect USB) so the lopy4 is no longer powered.
- Set up a voltage divider circuit on the breadboard using two resistors (one of 6.8 kohm (+/- 1% precision) , and one of unknown resistance value (the colour code is hidden to you with a black shrink sleeve). Connect it to the expansion board. Please follow the schematic below meticulously and double-check everything before proceeding to the next step.
- Connect the expansion board to your laptop again (USB cable) to power the lopy4.
- Please make sure that the pin on which you do a voltage measurement in the script (P14) is the same as the physical pin from which you measure a voltage on the breadboard.
- Now upload the script and examine the voltage measurement results. Compare the voltage reading with the value you read with a digital multimeter.





We can also use the ADC to measure a resistance. To read out the soil moisture content with Watermark® sensors, we need to be able to  measure the resistance of the sensor (the resistance that changes with the soil moisture content of the soil in which the sensor is placed).  The following schematic shows how to adapt the circuit to measure the resistance.  In this case we need to measure two voltages (V2 is approximately 3.3V, but for better accuracy it is best to measure that voltage too):





The circuit forms a voltage divider. We know that for two resistors in series the relative voltage drop across a resistor is proportional to its (relative) resistance:

$$\frac{V2 - V1}{V1} = \frac{R2}{R1}$$

That implies that the unknown resistance of R2 can be calculated as:

$$\mathrm{R2} = \frac{V2 - V1}{V1} R1$$

As we know the value of R1 (6.8 kohm +/- 1%) and can measure V1 and V2, we can calculate R2.

**Task 7:** In this task, you will determine the resistance of the unknown resistor. Use the schematic and the formula given above. You will need to modify your code to measure the voltage on two pins and to use that to calculate the resistance in your script. It is best to create a new folder " \IoT course\ReadResistance\" in which you copy all the contents of the folder '\IoT course\ReadVoltage\'. Next you can open the new folder folder " \IoT course\ReadResistance\" and adapt the code

## 6. Action when button is pressed and interrupt calls

Pins can be defined as input pins (to sense something, to receive digital data or to sense when a button is pressed) or as output pins (to switch something on, to lit a LED light or to send digital information). As explained in the slides, one can use a pushbutton to trigger an interrupt call. With that call, some action can be taken, such as toggle a LED on and off.

**Task 8:**

- Run main.py in folder \IoT course\PushButton&LedExpansionBoard and observe what happens when you push user button 'S1'.
- What changes are needed in the code if we want to implement the configuration with the internal pull-down resistor (right hand sides in slide on internal pull-up and pull-down resistor)? No need to also implement the changes.

## 7. Keeping time and deepsleep

To unsure low power consumption, it is essential that IoT devices go into deepsleep mode between two measurements or actions. During deepsleep mode the CPUs and peripherals are powered off. Only the Real Time Clock (RTC) keeps running (to keep time and to initiate a wakeup at a set time). It is of course essential that the lopy4 remains powered during deepsleep. The RTC and deepsleep functions are explained in the slides.

**Task 9:**

- Run main.py in folder \IoT course\Deepsleep and observe what happens
- Observe what happens when you push user button 'S1' during deepsleep.
- In order to properly set the time you need to interrupt the script by repeatedly pressing CTRL+C while the programme is active (blinking led). This does not work during deepsleep (but you can use the reset button on the lopy4 to get it out of deepsleep)
- Once you have stopped the script and you get the command prompt, you need to set the time with the command: rtc.init((2019, 3, 11, 15, 39))  (year, month, hours, minutes, seconds)
- Now restart the programme by clicking on the 'run' menu button (upper right of Pymakr window in Atom). Check if the time is restored after deepsleep.

## 8. Sending data with LoRaWAN

First register yourself to TTN (and get username and password) and next also register your lopy4 as device following the steps explained in the slides.

**Task 10:**

- (After saving file) Run main.py in folder \IoT course\SendLora2TTN_abp and check both the Pymakr console and the TTN website. Is any downlink received?