Internet of Things (IoT) technology for hydrological measurements

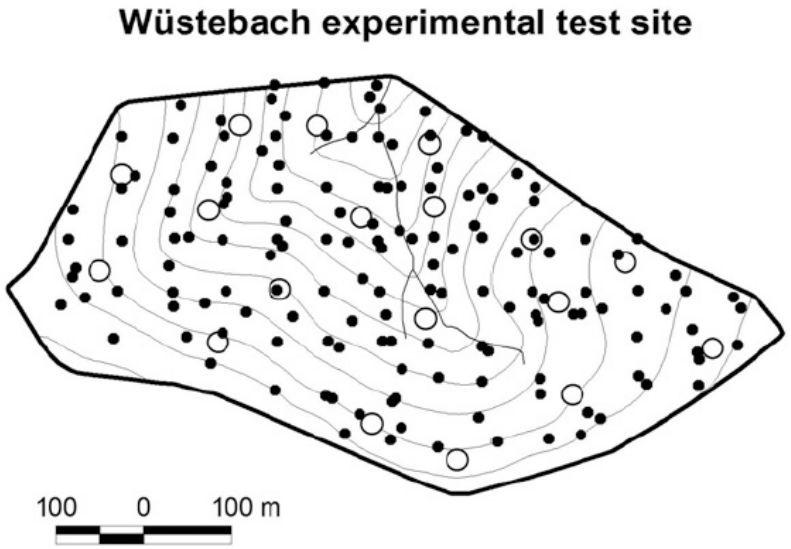
Crash course at NMAIST Arusha, 11-12 March 2019

Resource persons: Douglas Nyolei and Jan Diels

µcontroller with antenna

Sensors

Inexpensive wireless sensor networks allow to measure at spatial scales that matter for hydrologists



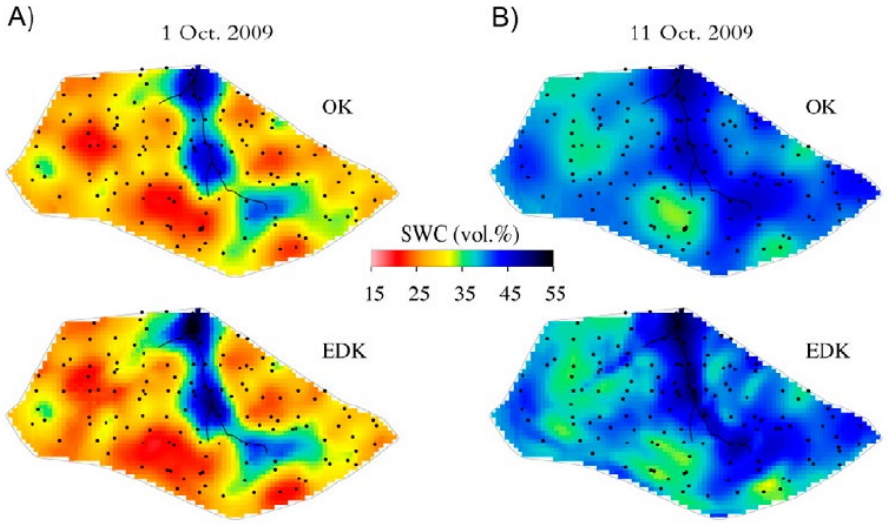


Fig. 8. Interpolated maps of soil water content (SWC) for the (A) drier and (B) wetter measurement dates using ordinary kriging (OK) and external drift kriging (EDK).

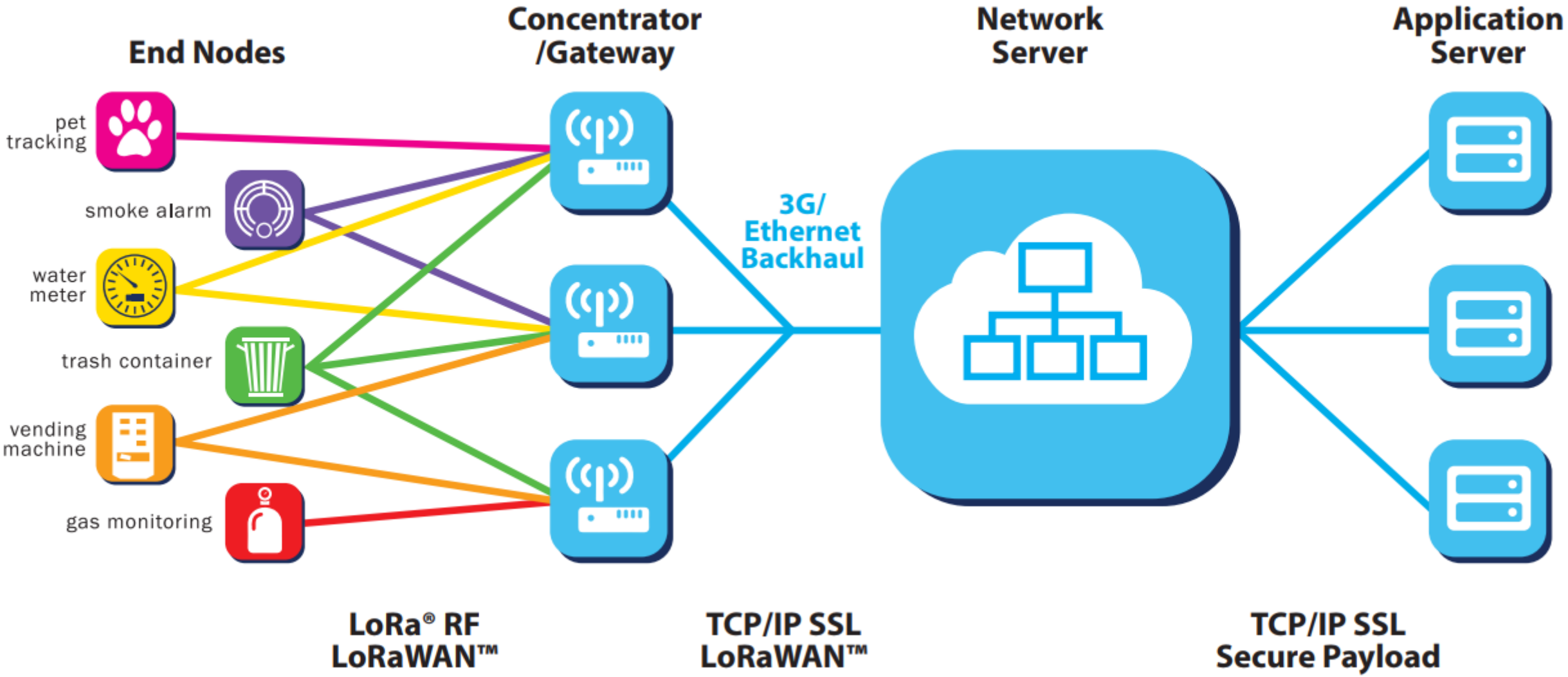
Source: Bogena, H.R., Herbst, M., Huisman, J.A., Rosenbaum, U., Weuthen, A., Vereecken, H., 2010. Potential of Wireless Sensor Networks for Measuring Soil Water Content Variability. Vadose Zone Journal 9, 1002-1013.

Options for wireless communication

	GSM	Wifi & Bluetooth	LoRaWAN & Sigfox
Range	<30 km	<20-50 m	5-10 km
Energy consumption	High	High	Low
Frequency band	Licensed	License-free	License-free
Data speed	High	High	Low

Protocols for Internet of Things (IoT)

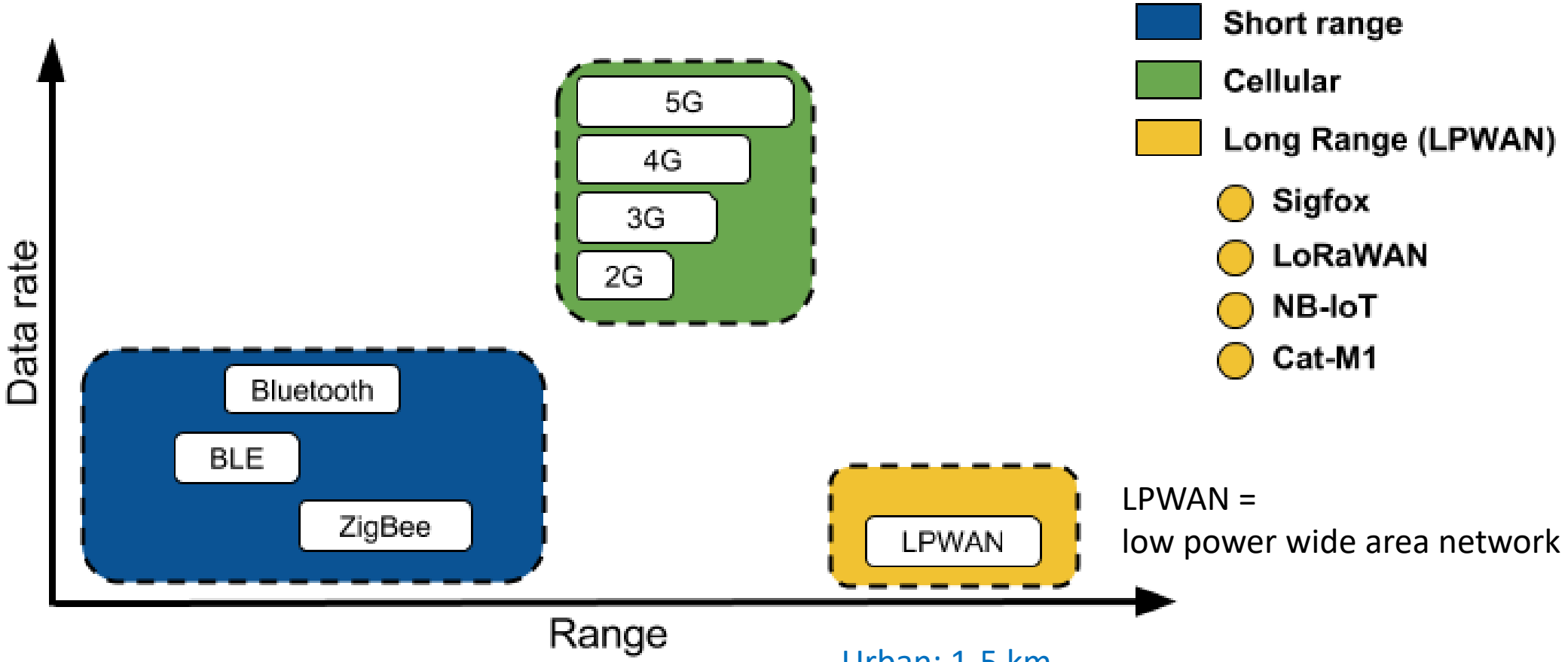
Battery-operated devices with autonomy of months-years that send only few measurements in a day



Source:
LoRa® and LoRaWAN™ Technical Marketing Workgroup 1.0, 2015. A technical overview of LoRa® and LoRaWAN™. https://lora-alliance.org

High power (battery life 1-2 days)

Low power (battery life months to years)



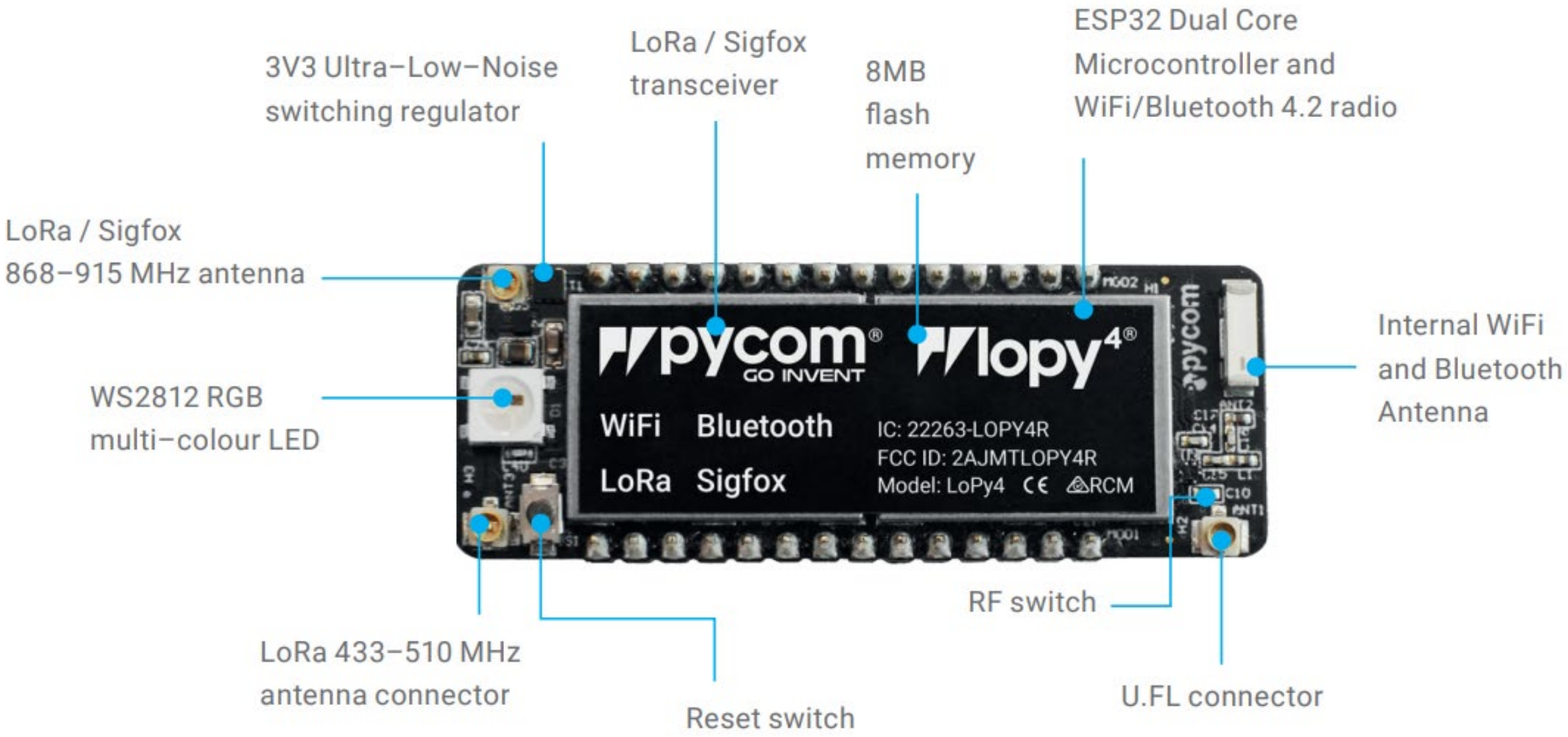
LPWAN =
low power wide area network

Urban: 1-5 km
Rural: 10-40 km

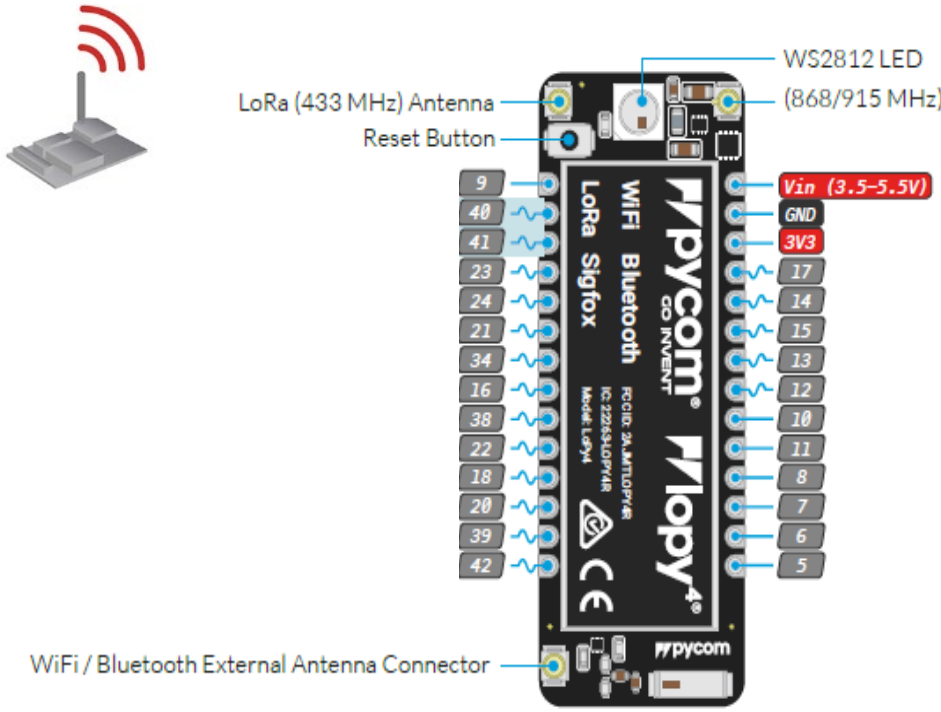
Source: Maria Hernandez (2018). Connectivity Now and Beyond; exploring Cat-M1, NB-IoT, and LPWAN Connections. https://ubidots.com/blog/exploring-cat-m1-nb-iot-lpwan-connections/

LoRaWAN = Long Range Wide Area Network = an **open** protocol for wireless data transmission for IoT

Sigfox = **proprietary** protocol for wireless data transmission for IoT as used by the French company Sigfox



Source: Datasheet lopy4 at Pycom website



LoPy^4 (pycom):

ESP32 dual core processor @ 160 MHz

Programmed in MicroPython

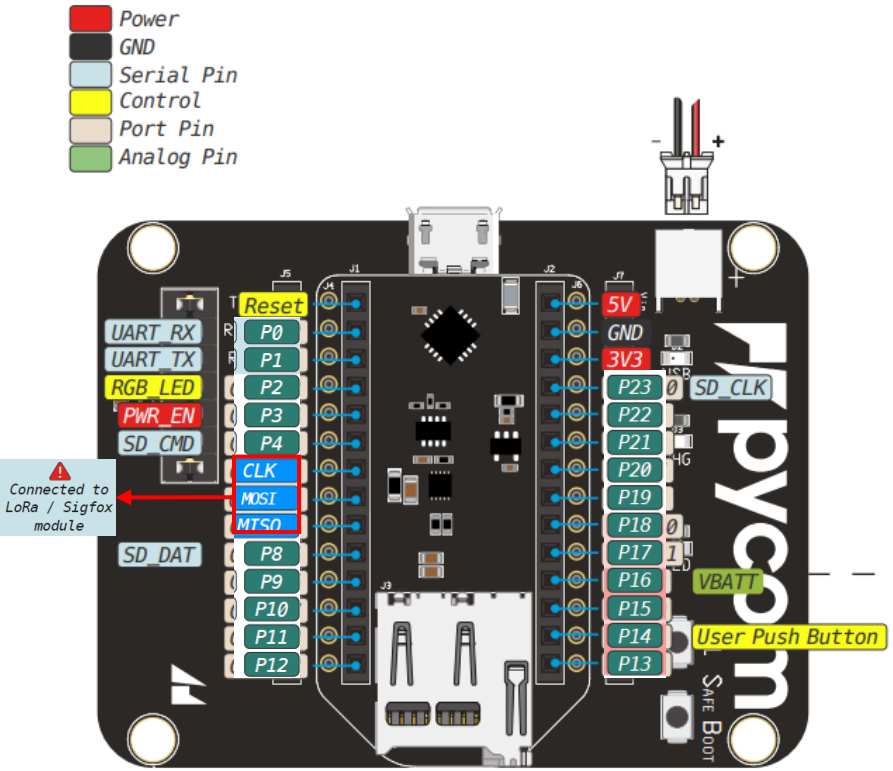
Bluetooth, Wifi, Sigfox, LoRaWAN

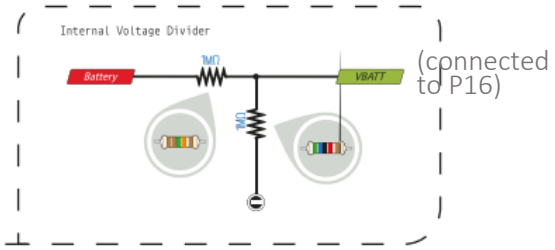
22µA in deepsleep mode (battery of 2000mAh will last for 10 years)

15mA when active and sending data (battery of 2000mAh will last for 2.2 years if device is active 10 minutes per day).

Pycom Expansion board V3.1

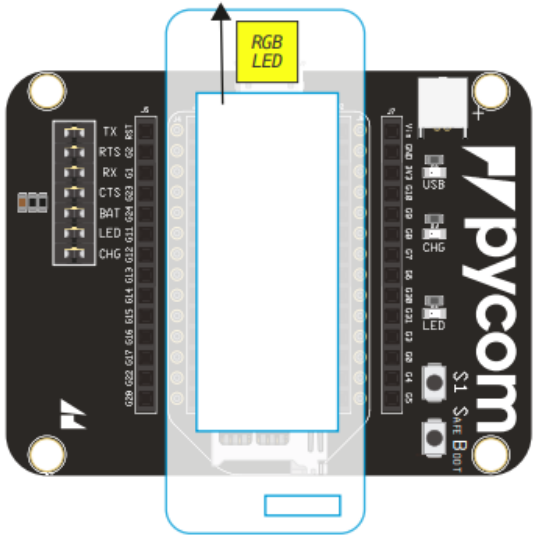
Pin numbers ‘P0’ to ‘P23’ written on lopy4 and on expansion board V3.1 are also used in python code. Do not confuse them with the G1- G17 numbers that are different!

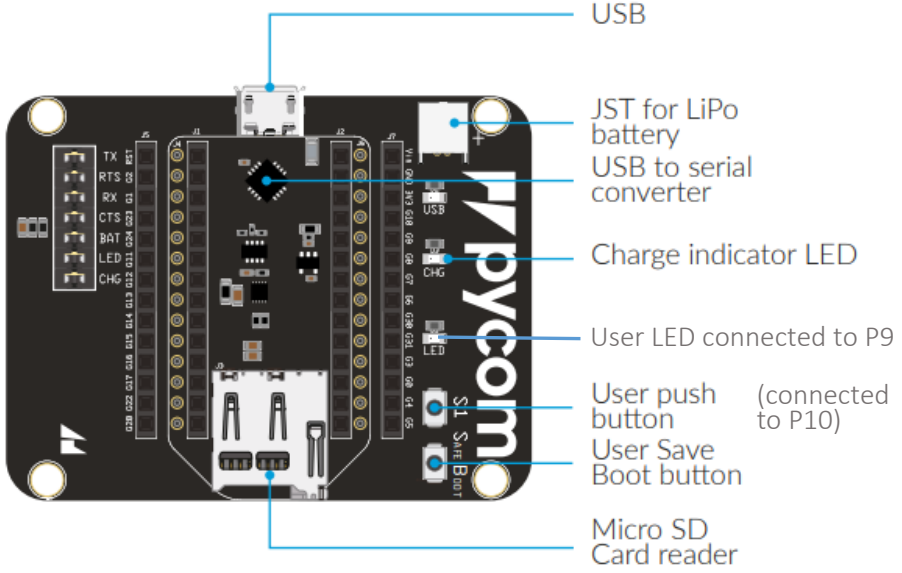




Correct Board Mounting

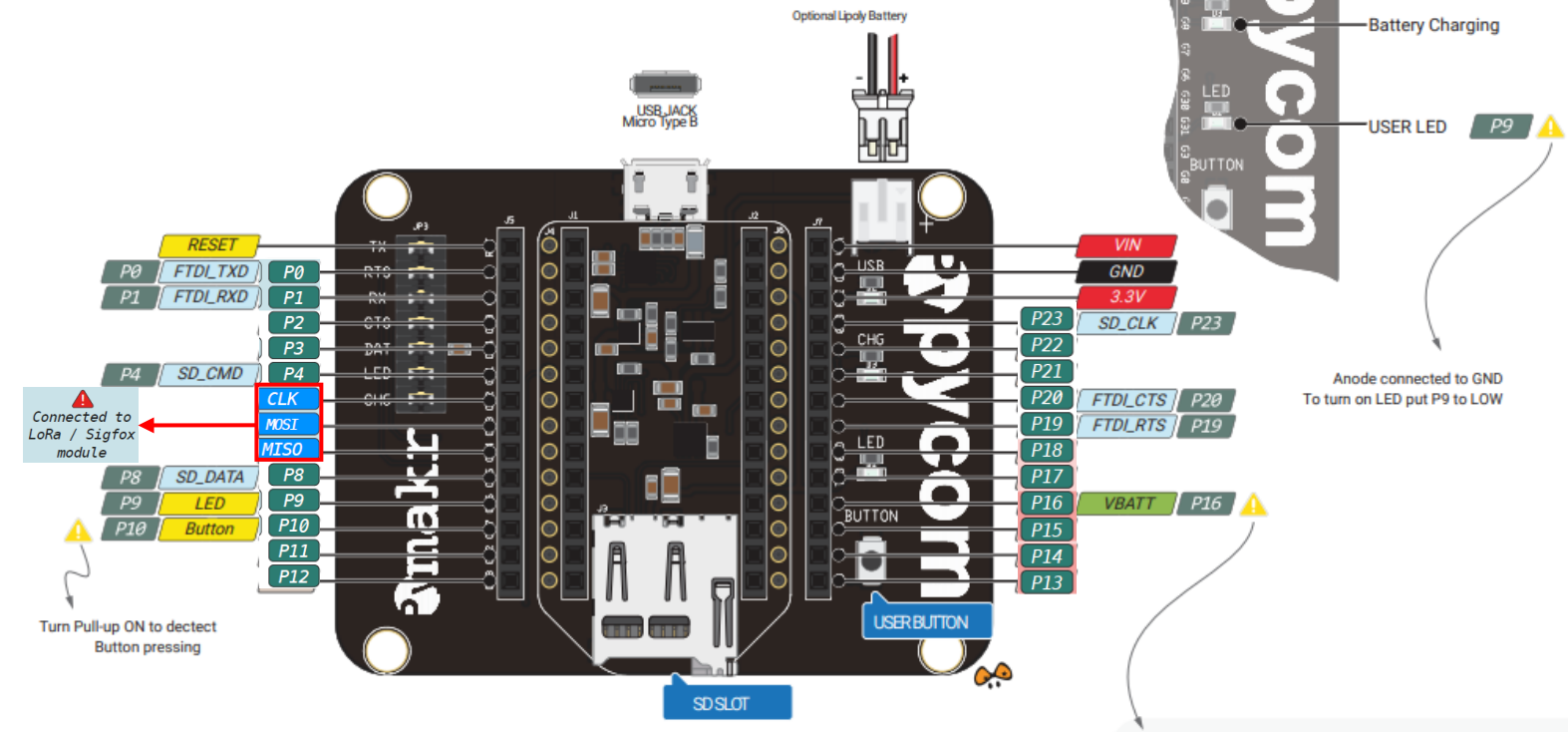
RGB LED must be at the top of the board, above the USB connector

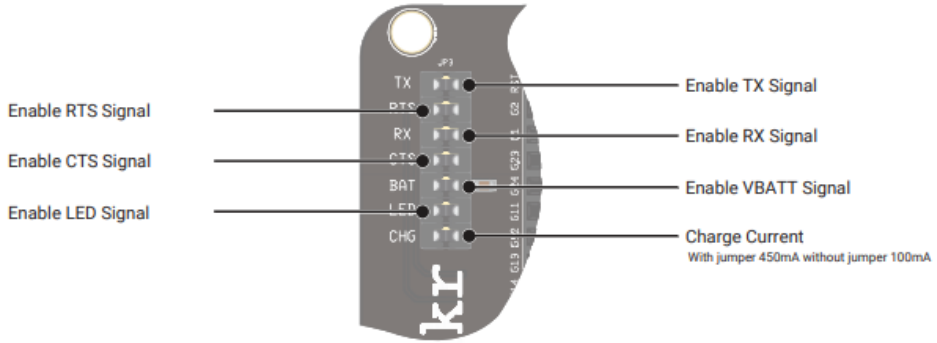


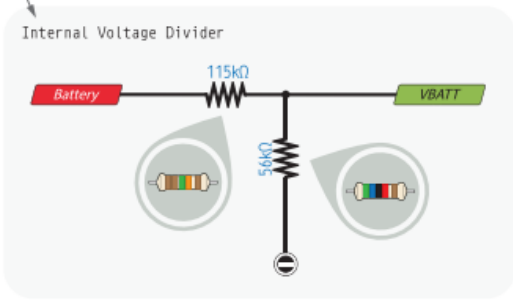


P5 (CLK), P6 (MOSI), and P7 (MISO) normally should not be used as they are connected to the LoRa and Sigfox modules.

Expansion Board PINOUT







Correspondence between G-numbers and P numbers						
G >>> P			**P >>>> G**		**on expansion board connected to:**	
G0	P15		P0	G2	UART RXD	
G1	P1		P1	G1	UART TXD	
G2	P0		P2	G23		
G3	P16		P3	G24		
G4	P14		P4	G11	SD card CMD	
G5	P13		P5	G12	Sigfox/LoRa	
G6	P19		P6	G13	Sigfox/LoRa	
G7	P20		P7	G14	Sigfox/LoRa	
G8	P21		P8	G15	SD card DAT0	
G9	P22		P9	G16	user LED	
G10	P23		P10	G17	user button	
G11	P4		P11	G22		
G12	P5		P12	G28		
G13	P6		P13	G5		
G14	P7		P14	G4		
G15	P8		P15	G0		
G16	P9		P16	G3	Vbatt circuit	remove BAT jumper to avoid interference
G17	P10		P17	G31		
			P18	G30		
G22	P11		P19	G6	UART CTS	remove CTS jumper to avoid interference
G23	P2		P20	G7	UART RTS	remove RTS jumper to avoid interference
G24	P3		P21	G8		
			P22	G9		
G28	P12		P23	G10	SD card SLCK	
G30	P18					
G31	P17					
Information was gotten as follows:						
>>> from machine import Pin						
>>> Pin.exp_board.G22						
Pin('P11', mode=Pin.IN, pull=Pin.PULL_DOWN, alt=-1)						

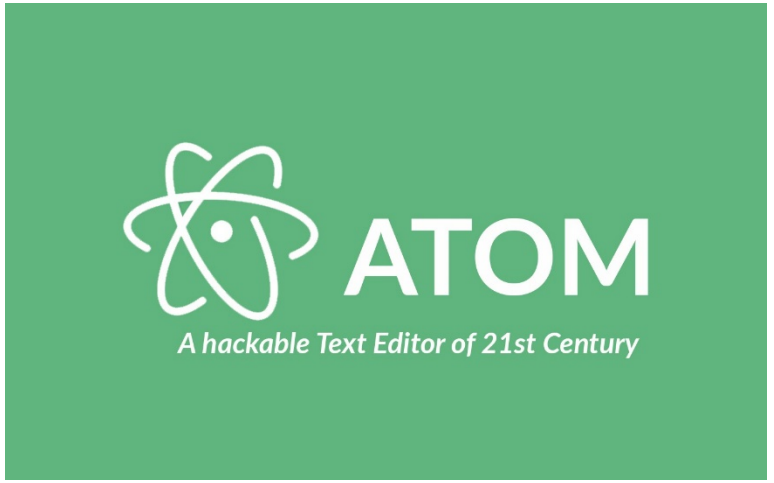
MicroPython

MicroPython = a programming language derived from Python 3.5 that is optimised to run on micro controllers:

- Upward compatible with Python 3.5 (but minor differences)
- Has specific libraries for accessing hardware (pins, led, ADC, …)

Language reference: https://docs.micropython.org
Documentation of the MicroPyhon that is implemented by Pycom: https://docs.pycom.io

Atom: source code editor that we will use for programming the Pycom board: https://atom.io/



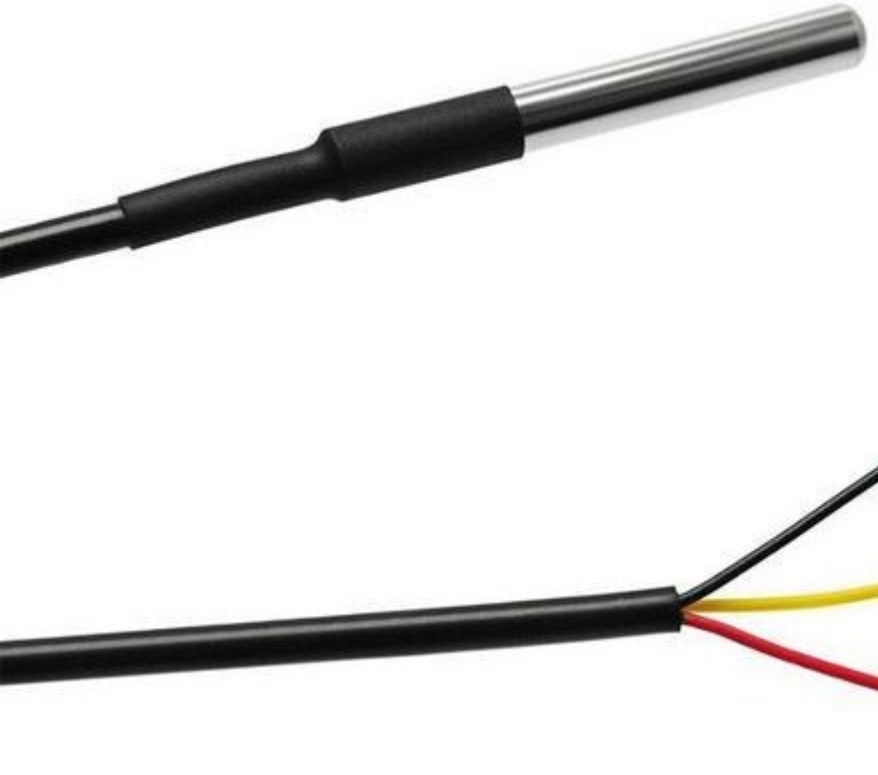
We install the Pymakr Extension (=plugin) in Atom

The Pymakr extension allows to:

- Execute python commands on the Pycom boards
- Edit MicroPython code
- Upload code to the Pycom boards (via USB cable or via Wifi)

Instructions on how to install Pymakr:
https://docs.pycom.io/pymakr/installation/atom.html

Reading the DS18B20 sensor with the on-wire protocol



How to connect to Lopy4

GND (=ground)

P10 or other pin (will be used for communicating with sensor)

+3.3V (=power to sensor)

Since each sensor can be addressed by a unique address, several sensors can be connected to the same data pin.

Reading the DS18B20 sensor with the on-wire protocol

Atom >> File >> Open folder >> \IoT course\DS18B20TemperatureSensor

Import libraries that come with firmware on lopy4

Import library from \lib\ folder in project

main.py:

```
from machine import Pin
import time, ubinascii
from onewire import OneWire, DS18X20

ow = OneWire(Pin('P10'))  # create a OneWire bus on P10

# Define function to measure the temperature with the DS18X20 sensor
# on a one-wire bus (owBUS).  If several sensors are connected to this bus
# you need to specify the ROM address of the sensor as well
# If only one DS18x20 device is attached to the bus you may omit the rom parameter.
def measureTemperature(owBus,rom=None):
    while True: # we loop until we get a valid measurement
        temp = DS18X20(owBus)
        temp.start_conversion(rom)
        time.sleep(1) # wait for one second
        TempCelsius=temp.read_temp_async(rom)
        if TempCelsius is not None:
            return TempCelsius # TempCelsius exit loop and return result

# single temperature measurement omitting the ROM parameter
SensorTempCelsius=measureTemperature(ow)
print("Temperature (degrees C) = %7.1f" % SensorTempCelsius)
```

Library provided by Pycom at:
https://github.com/pycom/pycom-libraries/tree/master/lib/onewire

\lib\onwire.py:

```
"""
OneWire library for MicroPython
"""

import time
import machine

class OneWire:
    CMD_SEARCHROM = const(0xf0)
    CMD_READROM = const(0x33)
    CMD_MATCHROM = const(0x55)
    CMD_SKIPROM = const(0xcc)

    def __init__(self, pin):
        self.pin = pin
        self.pin.init(pin.OPEN_DRAIN, pin.PULL_UP)

    def reset(self):
        """
        Perform the onewire reset function.
        Returns True if a device asserted a presence pulse, False otherwise.
        """
        sleep_us = time.sleep_us
```

Reading the DS18B20 sensor with the on-wire protocol

Open folder \IoT course\DS18B20TemperatureSensor in Atom

demo.py in folder \IoT course\DS18B20TemperatureSensor:

```
from machine import Pin
import time, ubinascii
from onewire import OneWire, DS18X20

ow = OneWire(Pin('P10'))  # create a OneWire bus on P10

temp = DS18X20(ow) # create a DS18X20 sensor object on the OneWire bus

temp.start_conversion() # send a command to the sensor to let it start the
# temperature measurement

time.sleep(1) # wait for one second (needed to ensure conversion is completed)

TempCelsius=temp.read_temp_async() # read the temperature from the sensor

print("Temperature (degrees C) = %7.1f" % TempCelsius) # print result
```

Open demo.py in atom and click on ‘run’ in pymakr window to run the demo file
Or copy and issue commands in command window

See https://docs.pycom.io/tutorials/all/owd.html

Reading the DS18B20 sensor with the on-wire protocol

Task 4: Now warm up the sensor with you hand and observe how quickly the sensor reacts to this (repeatedly run demo.py)

Open folder \IoT course\DS18B20TemperatureSensor in Atom, and next click ‘upload’ from the pymakr console. This will assure that the onwire.py library is available on the lopy4.

Next open demo.py in atom and click on ‘run’ in pymakr window to run the demo file.
Or copy and issue commands in command window

Reading the DS18B20 sensor with the on-wire protocol

main_new.py in folder \IoT course\DS18B20TemperatureSensor

```
from machine import Pin
import time, ubinascii
from onewire import OneWire, DS18X20

ow = OneWire(Pin('P10'))  # create a OneWire bus on P10

# Define function to measure the temperature with the DS18X20 sensor
# on a one-wire bus (owBUS).  If several sensors are connected to this bus
# you need to specify the ROM address of the sensor as well
# If only one DS18x20 device is attached to the bus you may omit the rom parameter.
def measureTemperature(owBus,rom=None):
    while True: # we loop until we get a valid measurement
        temp = DS18X20(owBus)
        temp.start_conversion(rom)
        time.sleep(1) # wait for one second
        TempCelsius=temp.read_temp_async(rom)
        if TempCelsius is not None:
            return TempCelsius # TempCelsius exit loop and return result

# single temperature measurement omitting the ROM parameter
SensorTempCelsius=measureTemperature(ow)
print("Temperature (degrees C) = %7.1f" % SensorTempCelsius)

# Each DS18X20 has a unique 64-bit (=8 bytes) address in its ROM memory
# (ROM = read only memory)
# When one or several DS18XB20 are connected to the same onewire bus, we can
# get their ROM addresses in the following way:
roms=ow.scan() # returns a list of bytearrays
for rom in roms: # we loop over the elements of the list
    print('ROM address of DS18XB20 = ',ubinascii.hexlify(rom))# hexlify to show
    # bytearray in hex format

# The following loop measures temperature continuously, looping over all
# sensors as well
while True: # loop forever (stop with ctrl+C)
    for rom in roms: # we loop over the elements of the list, i.e. we loop over
        # the detected DS18XB20 sensors
        print("For DS18XB20 with ROM address =", ubinascii.hexlify(rom), "the temperature (degrees C) = %7.1f"
% measureTemperature(ow,rom))
```

Indentation determines the grouping of statements in python

Reading the DS18B20 sensor with the on-wire protocol

main_new.py in folder \IoT course\DS18B20TemperatureSensor

```
from machine import Pin
import time, ubinascii
from onewire import OneWire, DS18X20

ow = OneWire(Pin('P10'))  # create a OneWire bus on P10

# Define function to measure the temperature with the DS18X20 sensor
# on a one-wire bus (owBUS).  If several sensors are connected to this bus
# you need to specify the ROM address of the sensor as well
# If only one DS18x20 device is attached to the bus you may omit the rom parameter.
def measureTemperature(owBus,rom=None):
    while True: # we loop until we get a valid measurement
        temp = DS18X20(owBus)
        temp.start_conversion(rom)
        time.sleep(1) # wait for one second
        TempCelsius=temp.read_temp_async(rom)
        if TempCelsius is not None:
            return TempCelsius # TempCelsius exit loop and return result

# single temperature measurement omitting the ROM parameter
SensorTempCelsius=measureTemperature(ow)
print("Temperature (degrees C) = %7.1f" % SensorTempCelsius)

# Each DS18X20 has a unique 64-bit (=8 bytes) address in its ROM memory
# (ROM = read only memory)
# When one or several DS18XB20 are connected to the same onewire bus, we can
# get their ROM addresses in the following way:
roms=ow.scan() # returns a list of bytearrays
for rom in roms: # we loop over the elements of the list
    print('ROM address of DS18XB20 = ',ubinascii.hexlify(rom))# hexlify to show
    # bytearray in hex format

# The following loop measures temperature continuously, looping over all
# sensors as well
while True: # loop forever (stop with ctrl+C)
     for rom in roms: # we loop over the elements of the list, i.e. we loop over
        # the detected DS18XB20 sensors
        print("For DS18XB20 with ROM address =", ubinascii.hexlify(rom), "the temperature (degrees C) = %7.1f"
% measureTemperature(ow,rom))
```

Function arguments

Function definition

Return to the programme that called the function, and pass result (TempCelsius)

Function call

Function arguments passed from main programme to function

Another function call

Reading the DS18B20 sensor with the on-wire protocol

main.py in folder \IoT course\DS18B20TemperatureSensor

```
from machine import Pin
import time, ubinascii
from onewire import OneWire, DS18X20

ow = OneWire(Pin('P10'))  # create a OneWire bus on P10

# Define function to measure the temperature with the DS18X20 sensor
# on a one-wire bus (owBUS).  If several sensors are connected to this bus
# you need to specify the ROM address of the sensor as well
# If only one DS18x20 device is attached to the bus you may omit the rom parameter.
def measureTemperature(owBus,rom=None):
    while True: # we loop until we get a valid measurement
        temp = DS18X20(owBus)
        temp.start_conversion(rom)
        time.sleep(1) # wait for one second
        TempCelsius=temp.read_temp_async(rom)
        if TempCelsius is not None:
            return TempCelsius # TempCelsius exit loop and return result

# single temperature measurement omitting the ROM parameter
SensorTempCelsius=measureTemperature(ow)
print("Temperature (degrees C) = %7.1f" % SensorTempCelsius)

# Each DS18X20 has a unique 64-bit (=8 bytes) address in its ROM memory
# (ROM = read only memory)
# When one or several DS18XB20 are connected to the same onewire bus, we can
# get their ROM addresses in the following way:
roms=ow.scan() # returns a list of bytearrays
for rom in roms: # we loop over the elements of the list
    print('ROM address of DS18XB20 = ',ubinascii.hexlify(rom))# hexlify to show
    # bytearray in hex format

# The following loop measures temperature continuously, looping over all
# sensors as well
while True: # loop forever (stop with ctrl+C)
     for rom in roms: # we loop over the elements of the list, i.e. we loop over
        # the detected DS18XB20 sensors
        print("For DS18XB20 with ROM address =", ubinascii.hexlify(rom), "the temperature (degrees C) = %7.1f"
% measureTemperature(ow,rom))
```

8-bit address in hexadecimal notation

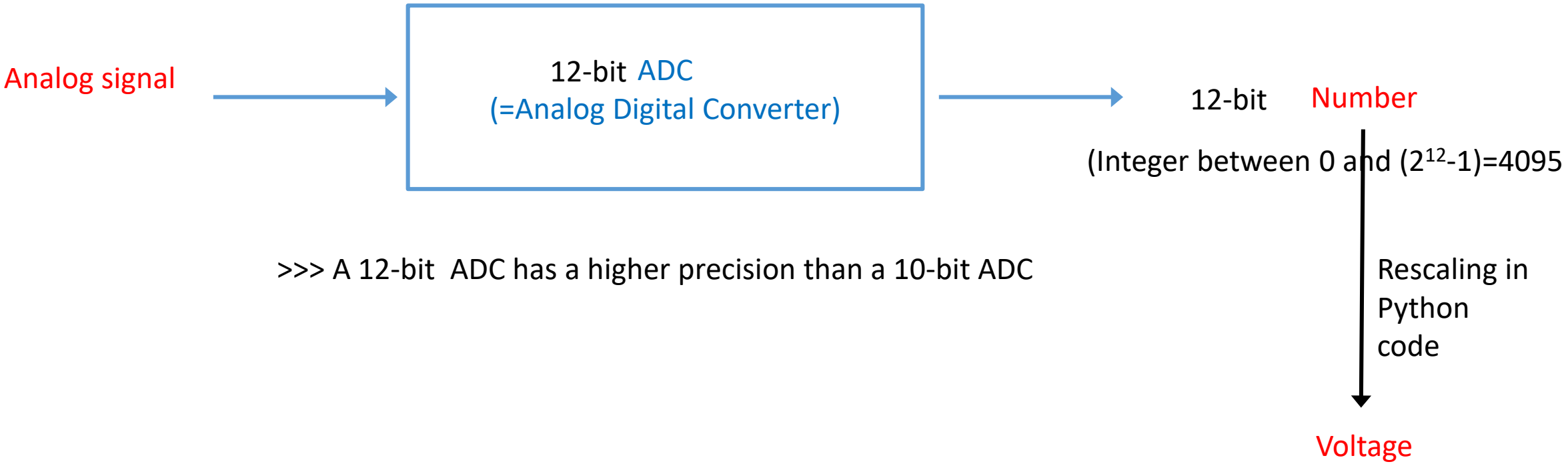
```
ROM address of DS18XB20 =  b'28aab10d1b13021c'
ROM address of DS18XB20 =  b'28aac4b81a130218'
```

Reading the DS18B20 sensor with the on-wire protocol

Task 5: For this task you need two DS18X20 sensors: work with neighbour

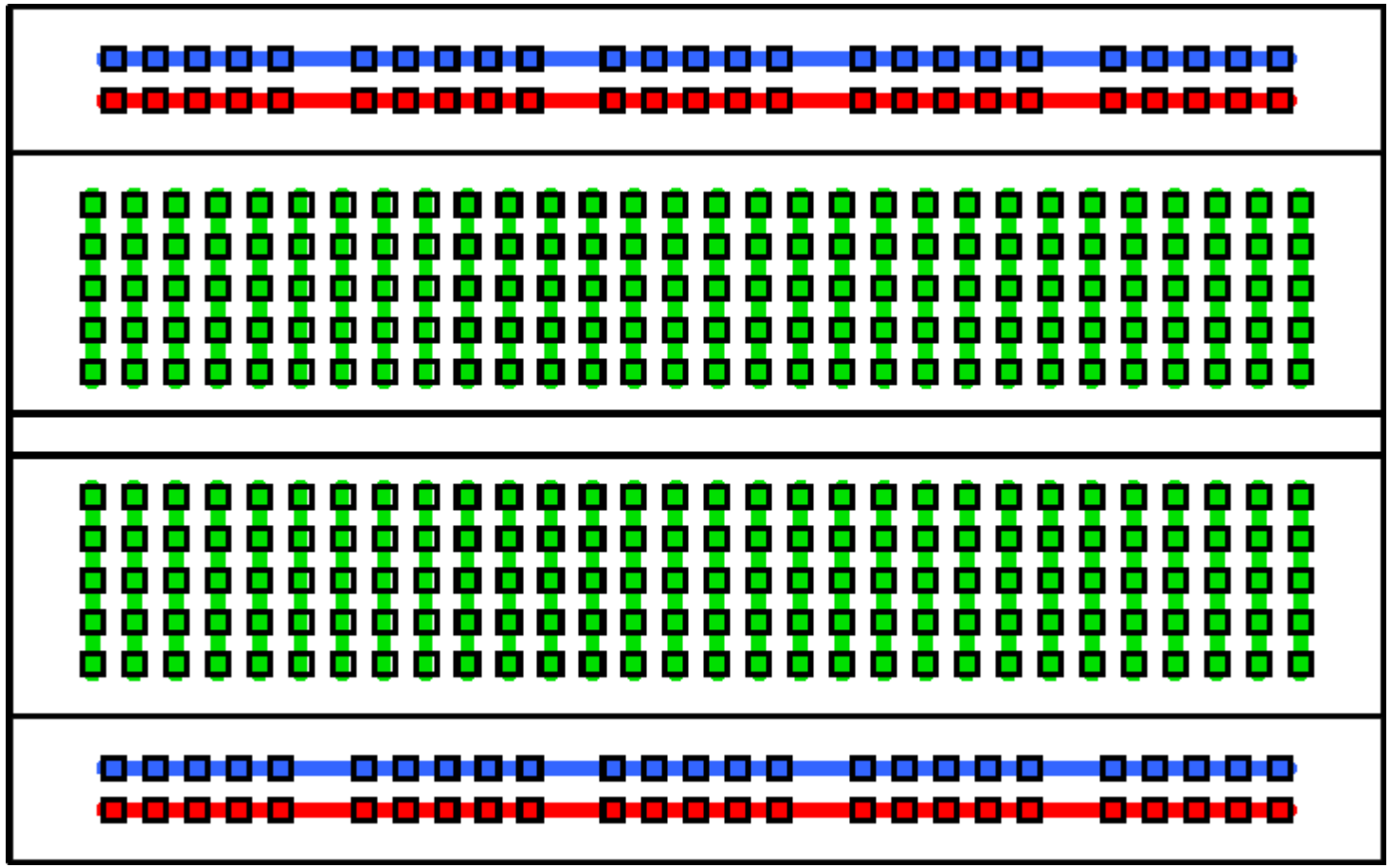
- Connect the two sensors to exactly the same pins on the lopy4: red to +3.3V, black to GND, and yellow to P10.
- Use the white breadboard and jumper cables to do this.
- Next run main.py and observe the temperature reading.
- Are the two sensors giving the same temperature? Check if the readings react to warming them with your hands (one by one).

Use of the analog-digital converter (ADC) on the LoPy4 to measure voltage

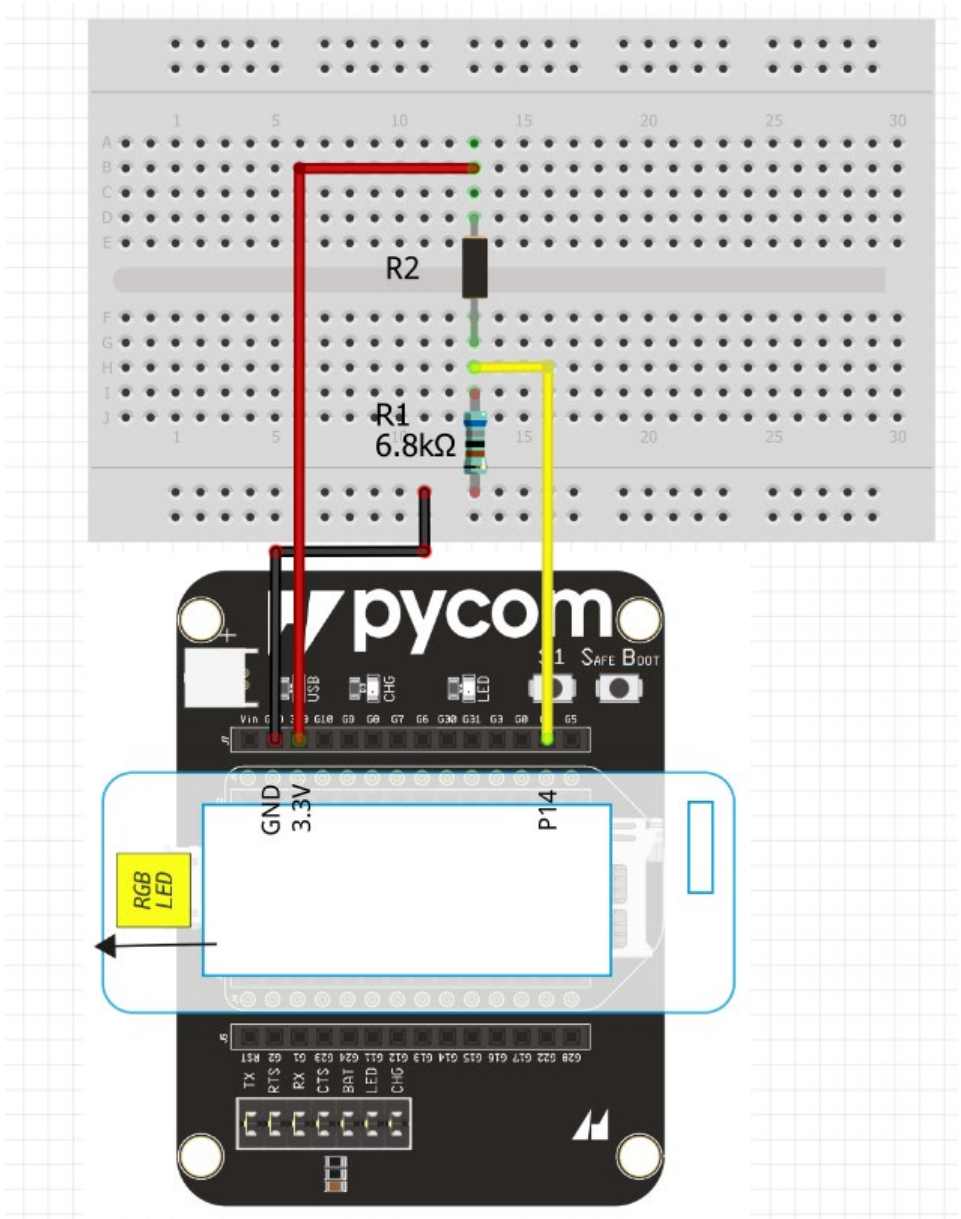


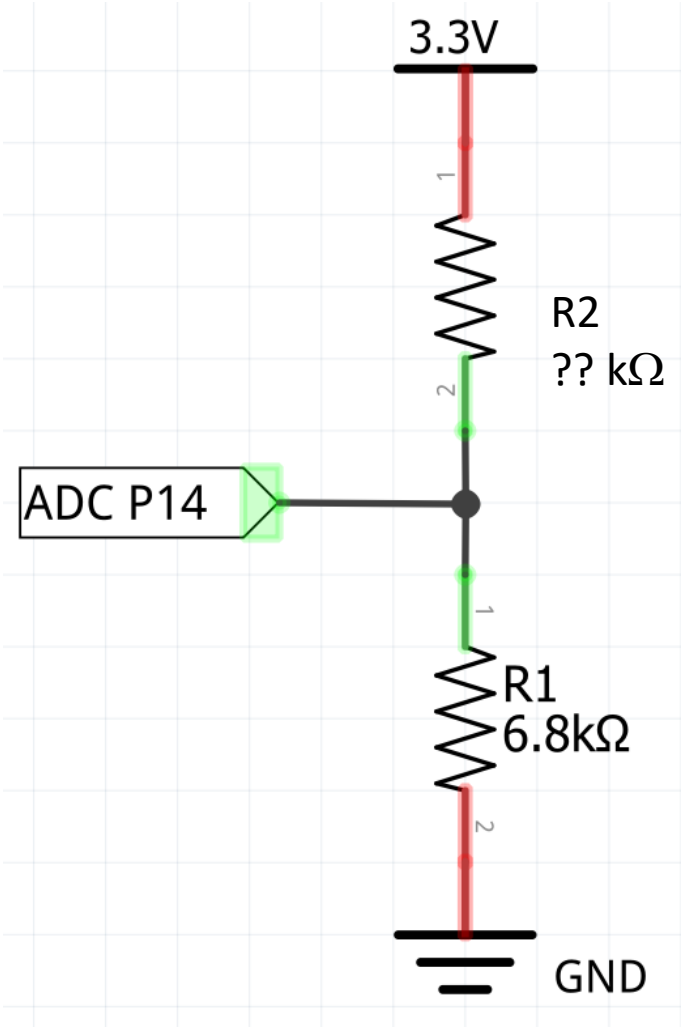
>>> A 12-bit ADC has a higher precision than a 10-bit ADC

Breadboard wiring:



Use of the analog-digital converter (ADC) on the LoPy4 to measure voltage





Use of the analog-digital converter (ADC) on the LoPy4 to measure voltage

main.py in folder \IoT course\ReadVoltage

```
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='P19',attn=attn)   # Create an analog pin on P1?, 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)
```

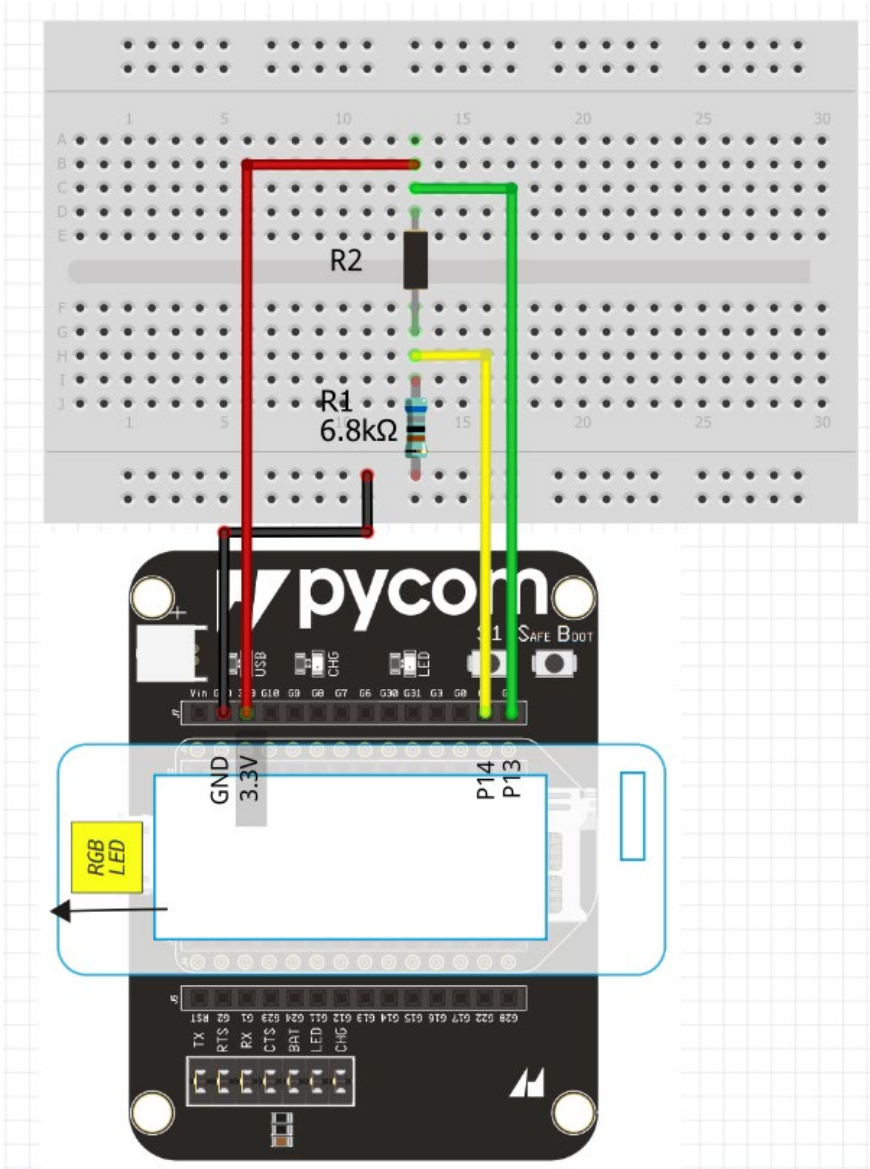
```
# adc.py    adc library with conversion to voltage
def adcRead(apin,nSamples=100):
    """
    Take ADC reading (average of nSamples) and convert to voltage.

    Keyword arguments:
    apin -- pin object as created with adc.channel() call
    nsamples -- (optional) number of times the ADC reading is repeated to calculate an average (de-
fault=100)
    """
    meanADC=0.0
    for i in range(nSamples):
        meanADC+=apin()    # read ADC and add to sum
    meanADC /= nSamples    # calculate mean by dividing by number of observations
    voltage=apin.value_to_voltage(int(meanADC+0.5))/1000 # convert using new built-in calibration
    return(meanADC,voltage)
```

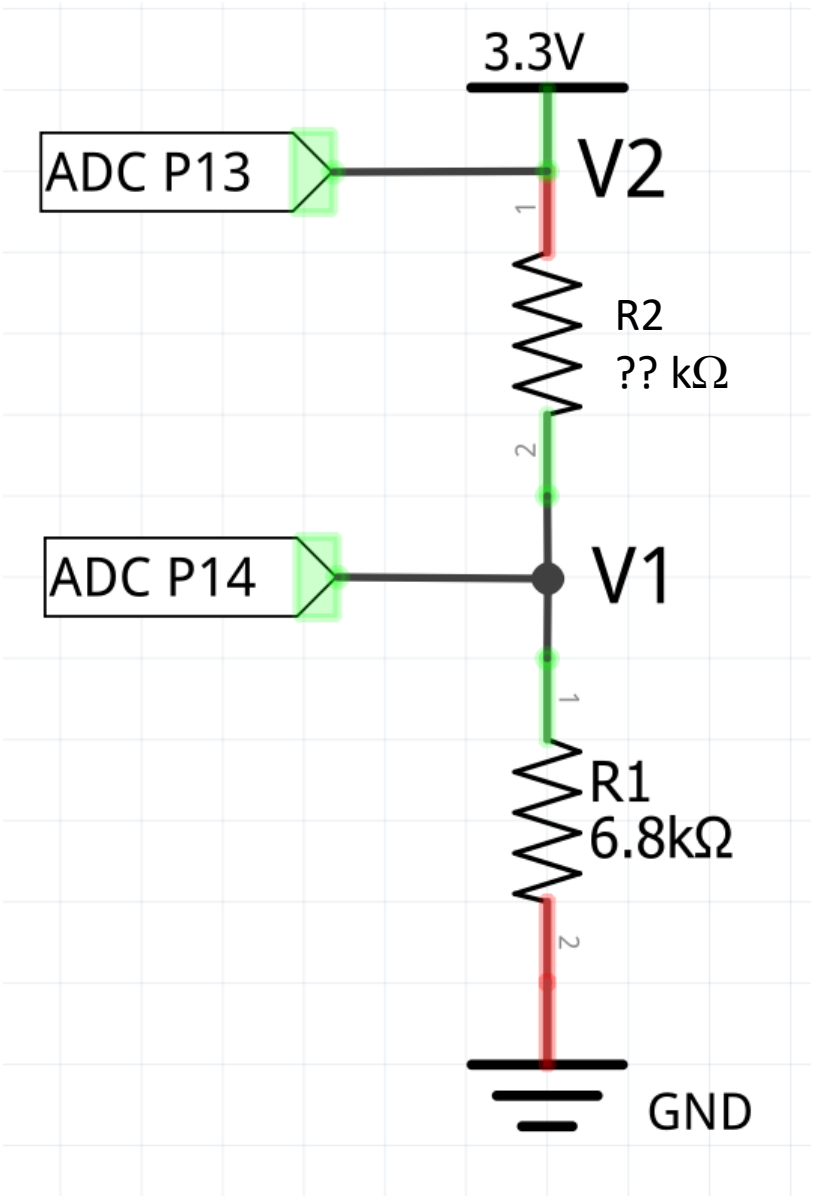
Use of the analog-digital converter (ADC) on the LoPy4 to measure voltage

Task 6:

- Create a voltage divider with the resistor of 6.8kΩ and the resistor of unknown value put in series on the breadboard. Wire everything with jumper cables to the expansion board with the lopy4 (see 2 slides back)
- Next run main.py (in folder \IoT course\ReadVoltage) and observe the voltage reading, and compare it with a voltage measurement with a multimeter.
- Check what happens if you set nSamples to 10 instead of 10000

Use of the analog-digital converter (ADC) on the LoPy4 to measure resistance





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

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

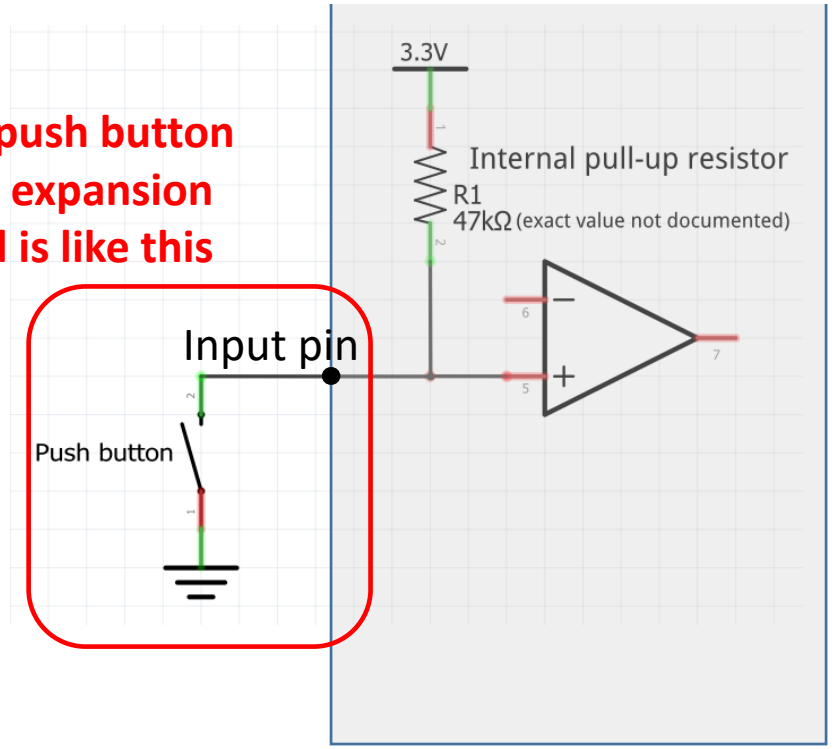
Use of the analog-digital converter (ADC) on the LoPy4 to measure resistance

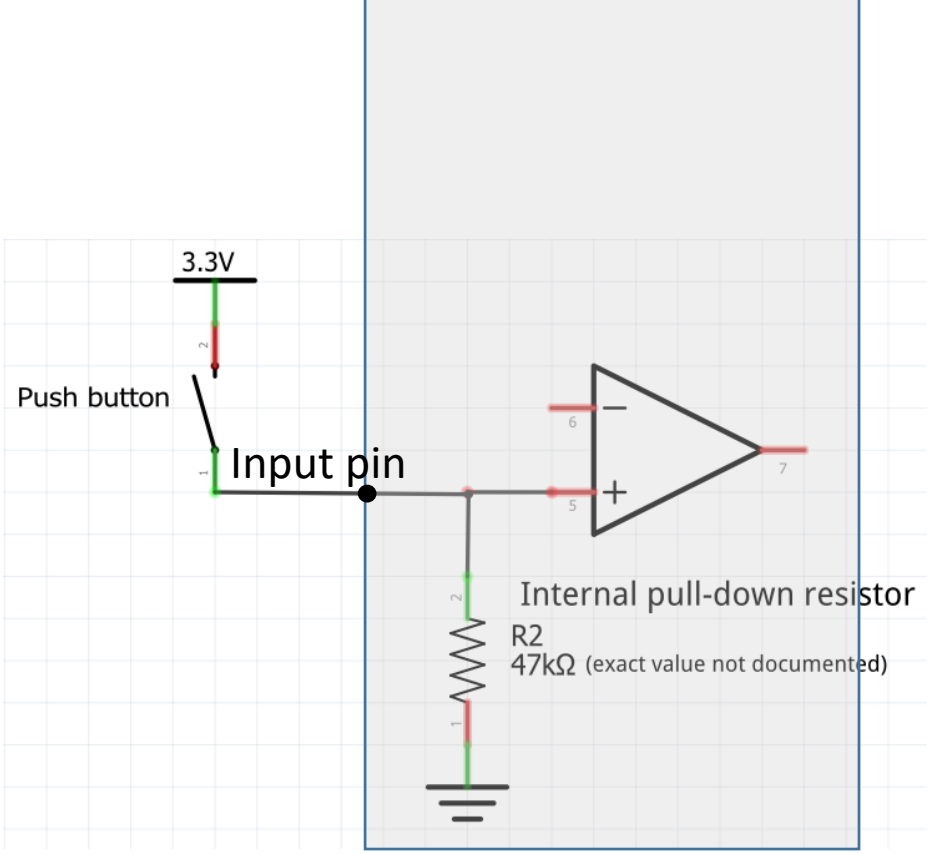
Task 7:

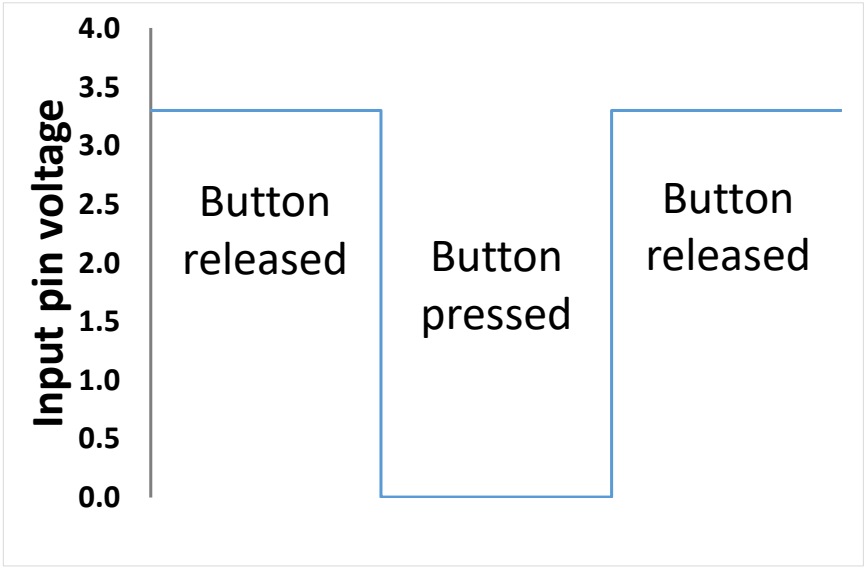
- Now also add the green jumper cable (to P13) as shown in the previous slide)
- Modify the script main.py (in folder \IoT course\ReadVoltage) to measure the two voltages (V1 and V2), and to calculate and print the value of the unknown resistance

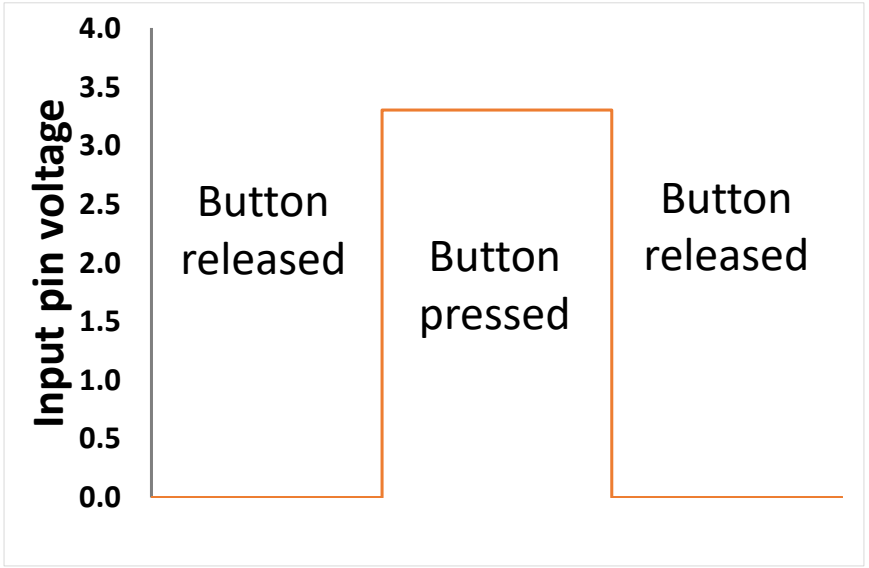
Action when button is pressed and interrupt calls

User push button S1 on expansion board is like this

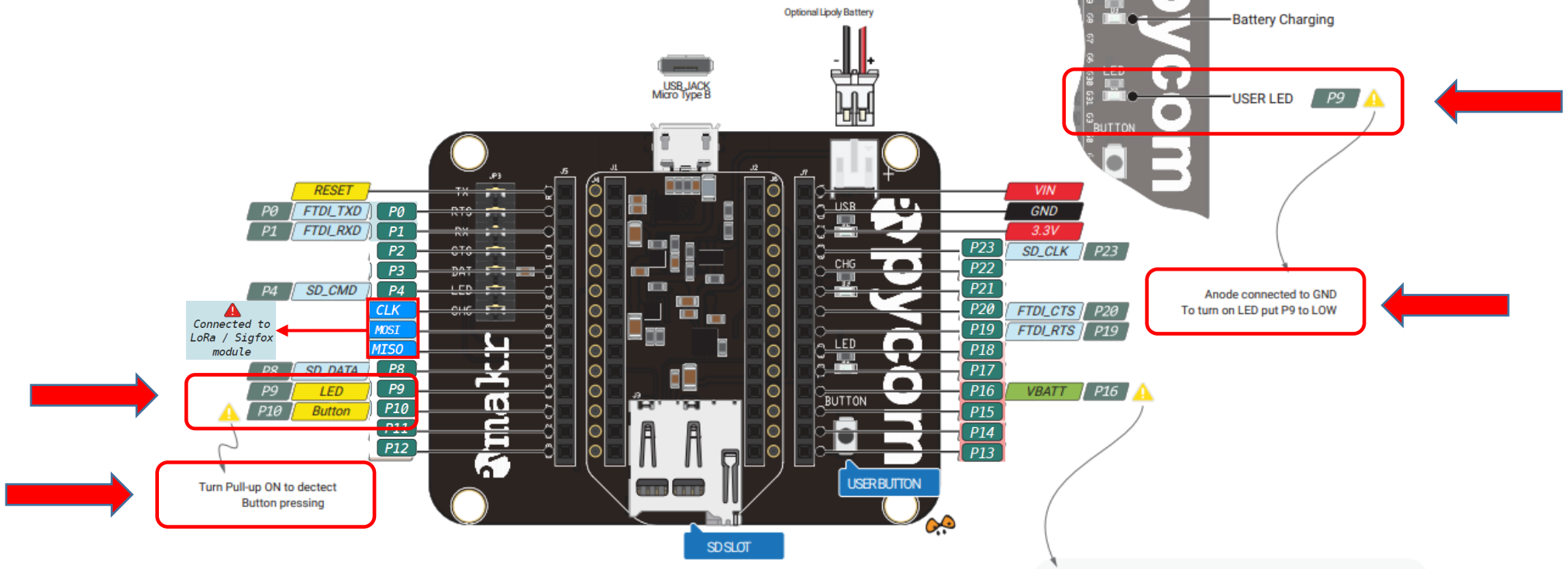


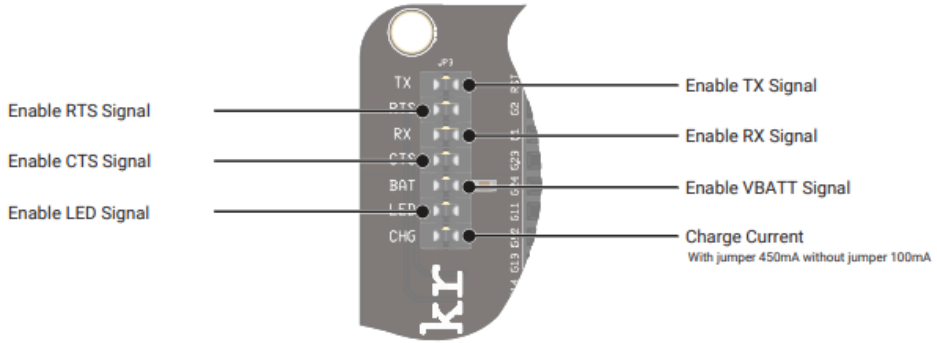


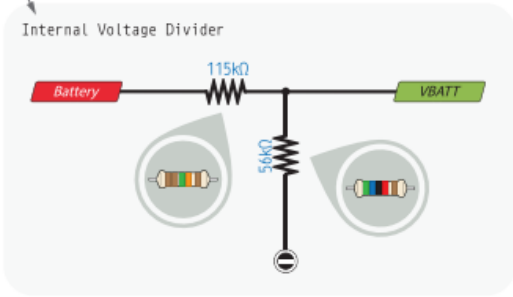




Expansion Board PINOUT







Action when button is pressed and interrupt calls

main.py in folder \IoT course\PushButton&LedExpansionBoard:

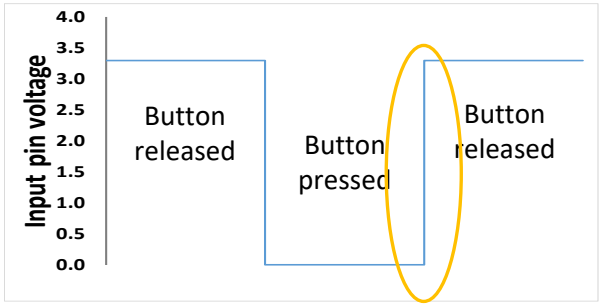
```
from machine import Pin

# initialize `P9` in gpio mode and make it an output
led=Pin('P9', mode=Pin.OUT) # user LED on expansion board
# is connected to P9

# initialize 'P10' in gpio mode as input with the pull-up enabled
button=Pin('P10',mode=Pin.IN, pull=Pin.PULL_UP)# user button on
# expansion board is connected to P10

def pin_handler(arg): # define callback function
    print("got an interrupt in pin %s" % (arg.id()))
    led.toggle()

button.callback(Pin.IRQ_RISING, pin_handler) # trigger callback function when
# button is pushed. When button is pressed, voltage on P10 falls.
# When released, it rises again and triggers a callback
```

P9 defined as output pin (>>> LED)

P10 defined as input pin (<<< button)

Define what action is needed when interrupt is called

Trigger interrupt on pin 'button'



Action when button is pressed and interrupt calls

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 3 slides back)? No need to also implement the changes.

PicoScope 6

File Edit Views Measurements Tools Help

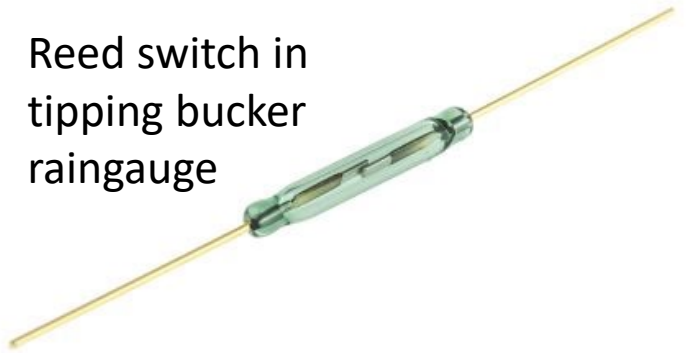
20 µs/div | 100 kS | 1 of 1 | x 1

A ±5 V DC | B Off DC

pico Technology

100 µs debounce time

Reed switch in tipping bucker raingauge



V
5.0
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0

-42.11 -22.11 -2.111 17.89 37.89 57.89 77.89 97.89 117.9 137.9 157.9

x1.0 µs

Stopped | Trigger Single | A | 1.544 V | 21.055 % | 0 s | Measurements | Rulers | Notes

Keeping time and deepsleep

Lopy4 (and all microprocessors and computers) have a Real Time Clock (RTC) that keeps time

The Real Time Clock can be accessed through the machine.TRC() function as follows:

```
>>> from machine import RTC
>>> rtc = RTC()
>>> print(rtc.now()) # This will print date and time if it was set before going
(1970, 1, 1, 2, 56, 0, 266802, None)
>>> rtc.init((2019, 3, 11, 15, 39)) # now manually set year, month, hour, minutes, seconds
>>> print(rtc.now()) # This will print date and time if it was set before going
(2019, 3, 11, 15, 39, 17, 806991, None)
>>>
```

Lopy4 does not have backup battery for RTC. When powered up, the RTC is set to beginning of 1 January 1970 ('Unix time')

Keeping time and deepsleep

main.py in folder \IoT course\Deepsleep:

```
# main.py -- put your code here!

from machine import RTC
from machine import Pin
import pycom
import time
import machine

pycom.heartbeat(False) # stop the heartbeat

# Set up the Real Time Clock (RTC)
rtc = RTC()
print(rtc.now()) # This will print date and time if it was set before going
# to deepsleep.  The RTC keeps running in deepsleep.

# rtc.init((2019, 3, 11, 15, 39)) # manually set the time

print("wake reason (wake_reason, gpio_list):",machine.wake_reason())
'''    PWRON_WAKE -- 0
       PIN_WAKE -- 1
       RTC_WAKE -- 2
       ULP_WAKE -- 3
 '''
# blink the led
for cycles in range(2): # stop after 2 cycles
    pycom.rgbled(0x007f00) # green
    time.sleep(1)
    pycom.rgbled(0x7f7f00) # yellow
    time.sleep(1)
    pycom.rgbled(0x7f0000) # red
    time.sleep(1)
```

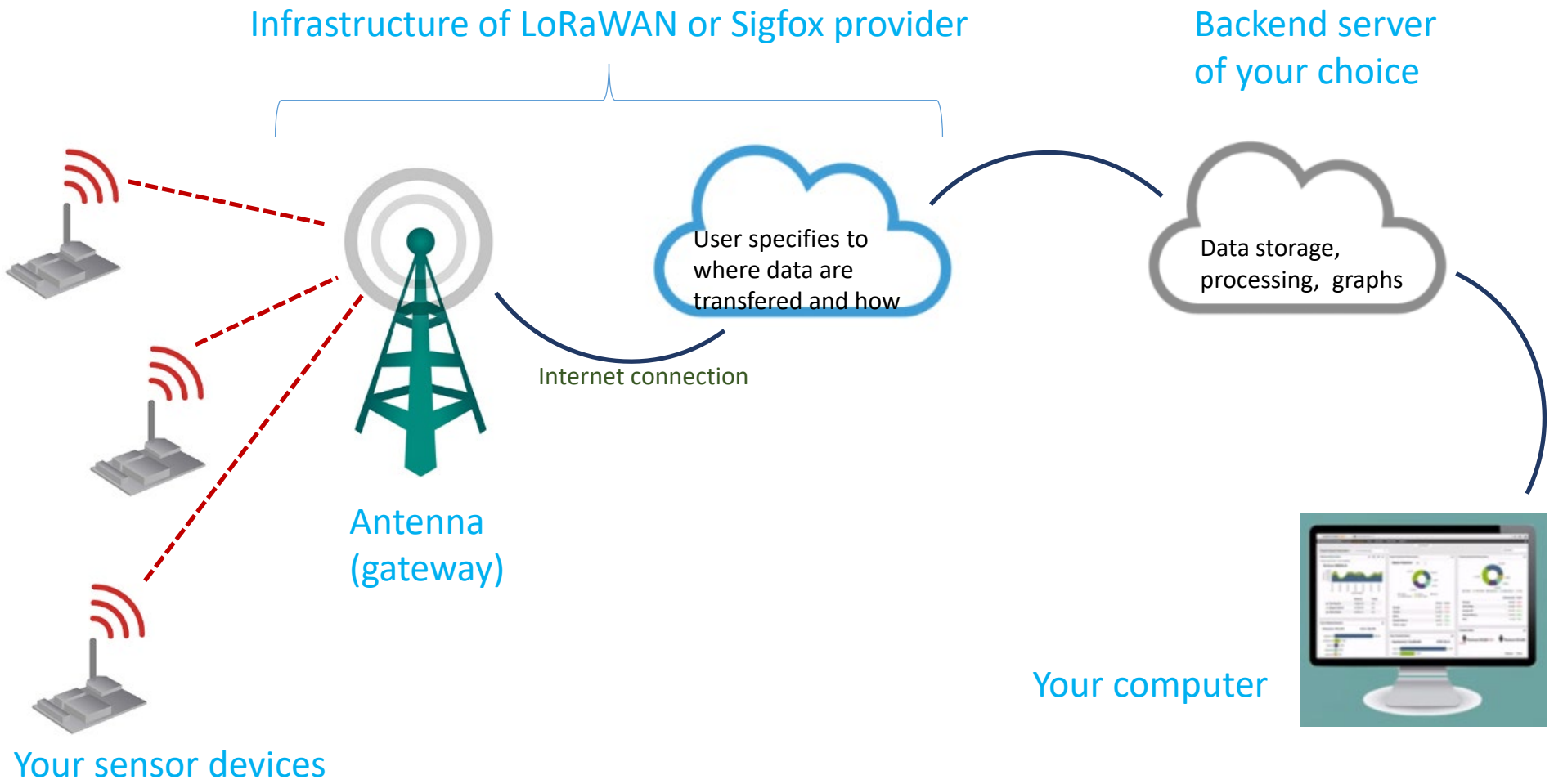
```
# We now want to set the user button on the Expansion Board as a button
# to wake up the lopy4 from deepsleep.  The user button on the
# expansion board is connected to P10
InterrupPin=Pin('P10',mode=Pin.IN, pull=Pin.PULL_UP) # define pin as input pin
# with pull-up resistor enabled (keeps pin high as long as button is not
pressed).

# Now configure pin P10 as deepsleep wakeup pin.
machine.pin_deepsleep_wakeup(pins=['P10'], mode=machine.WAKEUP_ALL_LOW, ena-
ble_pull = True)
# With WAKEUP_ALL_LOW we ask for wakeup when the pin goes low (when button is
pressed
# the pin gets connected to ground, hence goes low)
# With  'enable_pull = True' we keep the pull up resistor enabled during deep
sleep.

print("Time to go to sleep ....")
sleepSeconds=20  # set deepsleep time in seconds

machine.deepsleep(sleepSeconds*1000) # time in ms
# Note that when it wakes from deepsleep it reboots, but the RTC keeps time
during deepsleep
# You can always interrupt the deep sleep with the reset button on the LoPy4
```

Keeping time and deepsleep

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.

Sending data: How does it work?



Can we do it without provider? Yes with LoRaWAN

Backend server
of your choice

Antenna
(gateway)

Data storage,
processing, graphs

THE THINGS
NETWORK

Antenna network is
built by you - the
people

Nano gateway
with lopy4

Your computer

Your sensor devices

THE THINGS NETWORK

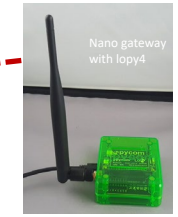
A gateway (=antenna) can reach sensors up to 10km away

At this moment, there are 2932 gateways up and running

Data communication we will use:

Cloud services

LoRaWAN



Your lopy4

HTTP via internet

THE THINGS NETWORK

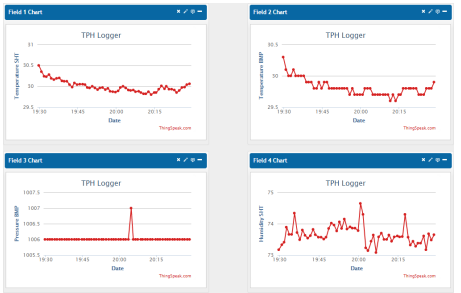
Node-RED

To reformat the data for ThingSpeak

HTTP via internet

ThingSpeak

MATLAB



Visualization, quality control, storage, analysis

Registering your lopy4 as a device on The Things Network (TTN)

Step 1: get the (unique) DevEUI from your lopy4

Open folder \IoT course\SendLora2TTN_abp in Atom (and close other folders):

findLoRaDeviceEUI.py in folder \IoT course\SendLora2TTN_abp:

```
# Run this code on your Lopy4 device to get the LoRa device EUI that you need
# to enter when registering the device on TTN
from network import LoRa
import ubinascii

lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
print("DevEUI: %s" % (ubinascii.hexlify(lora.mac()).decode('ascii')))
```

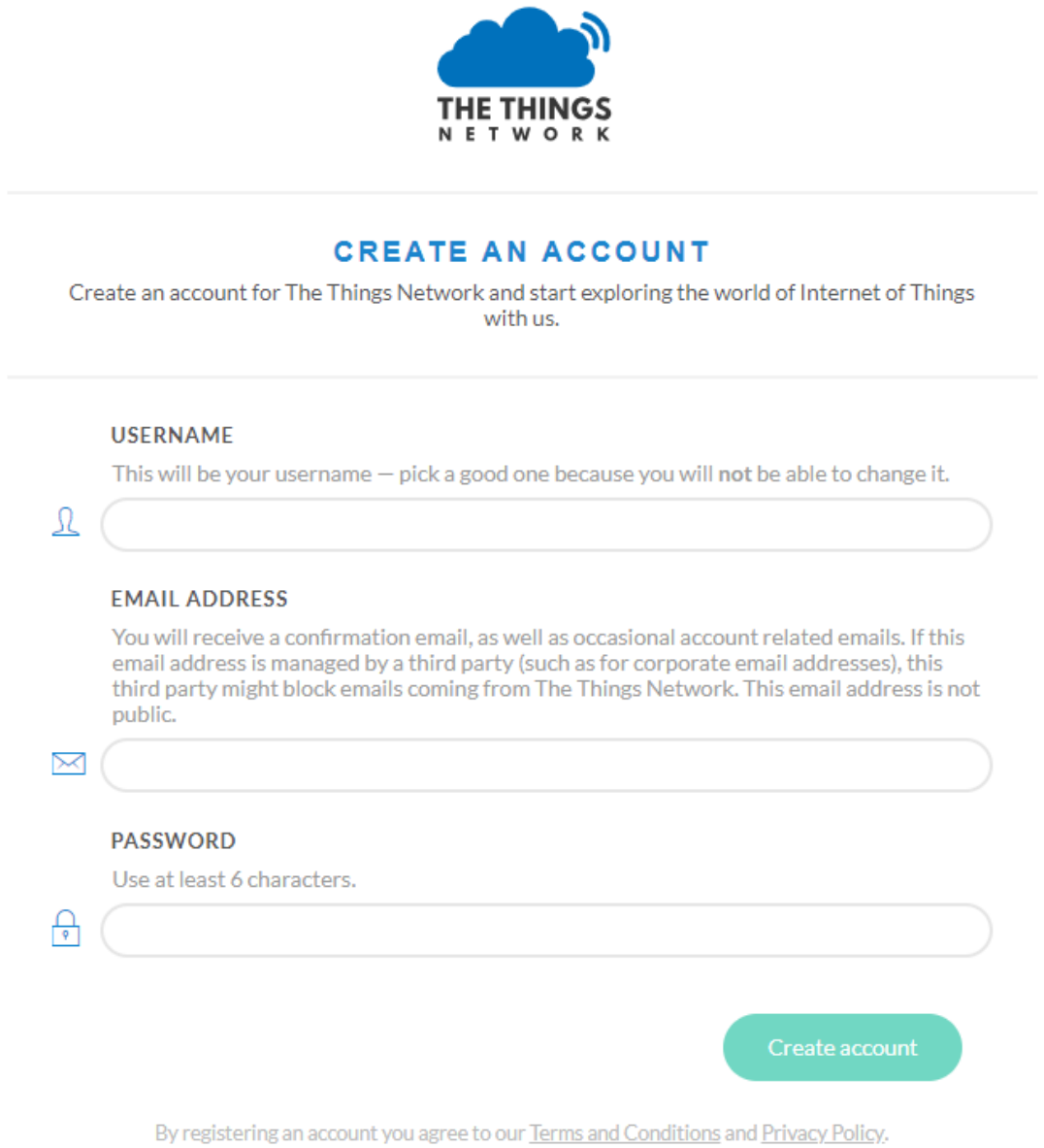
Make 'findLoRaDeviceEUI.py' the active window in Atom, next click on 'Run' in the Pymakr menu:

```
>>> Running C:\KULeuven\Lab\PyCom\Scripts\IoT course\SendLora2TTN_abp\findLoRaDeviceEUI.py
>>>
>>>
DevEUI: 70b3d54998e71974
>
Pycom MicroPython 1.18.2.r1 [v1.8.6-849-e0fb68e] on 2018-11-26; LoPy4 with ESP32
Type "help()" for more information.
>>>
>>>
```

Registering your lopy4 as a device on The Things Network (TTN)

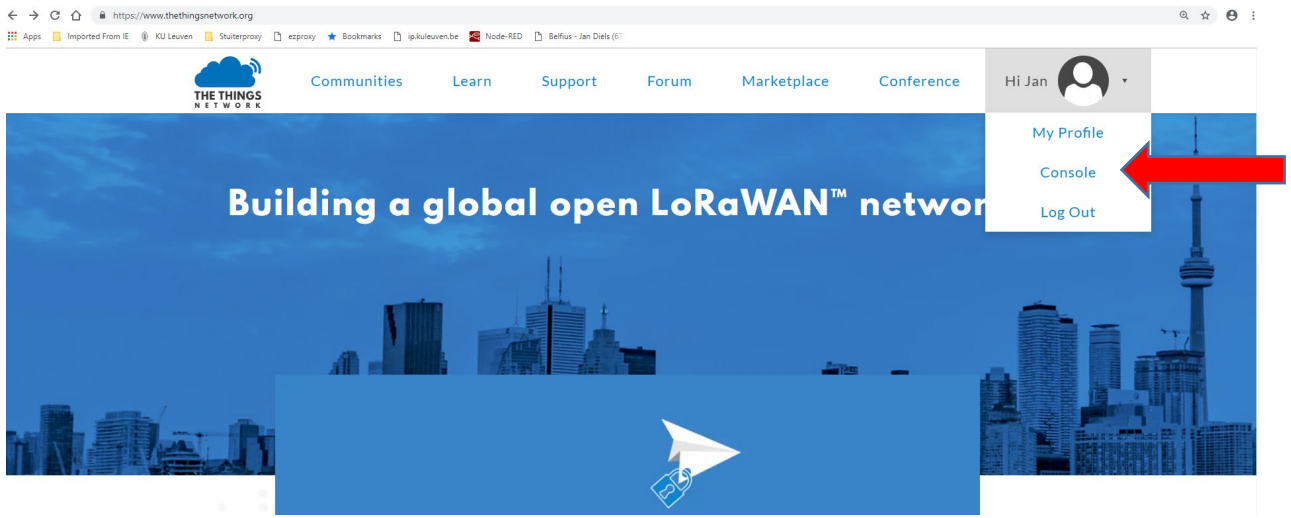
Step 2: Register yourself as user on The Things Network

https://account.thethingsnetwork.org/register



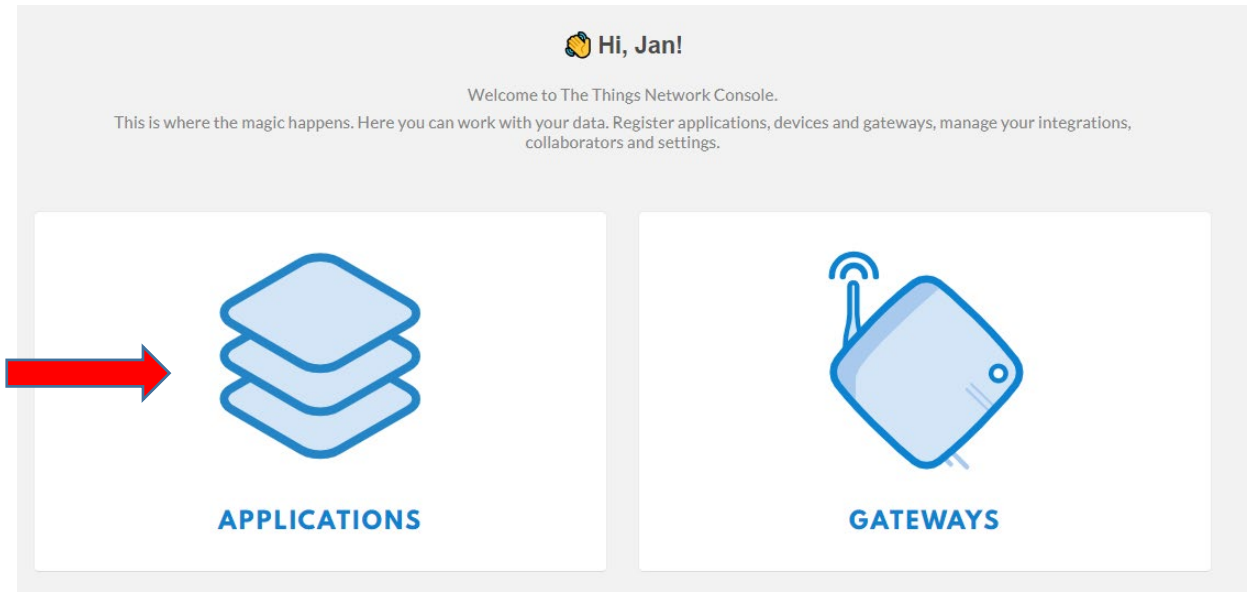
Registering your lopy4 as a device on The Things Network (TTN)

Step 3: Register an application on The Things Network



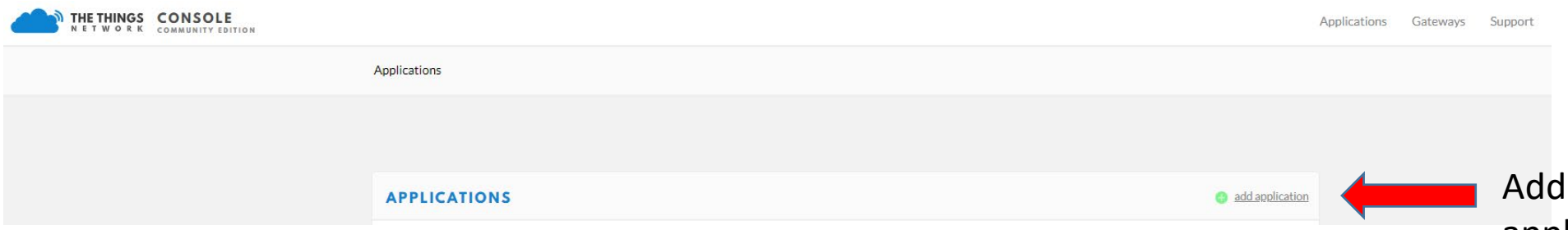
Open console

Next chose ‘Applications’



Registering your lopy4 as a device on The Things Network (TTN)

Step 3: Register an application on The Things Network



Add application

Registering your lopy4 as a device on The Things Network (TTN)

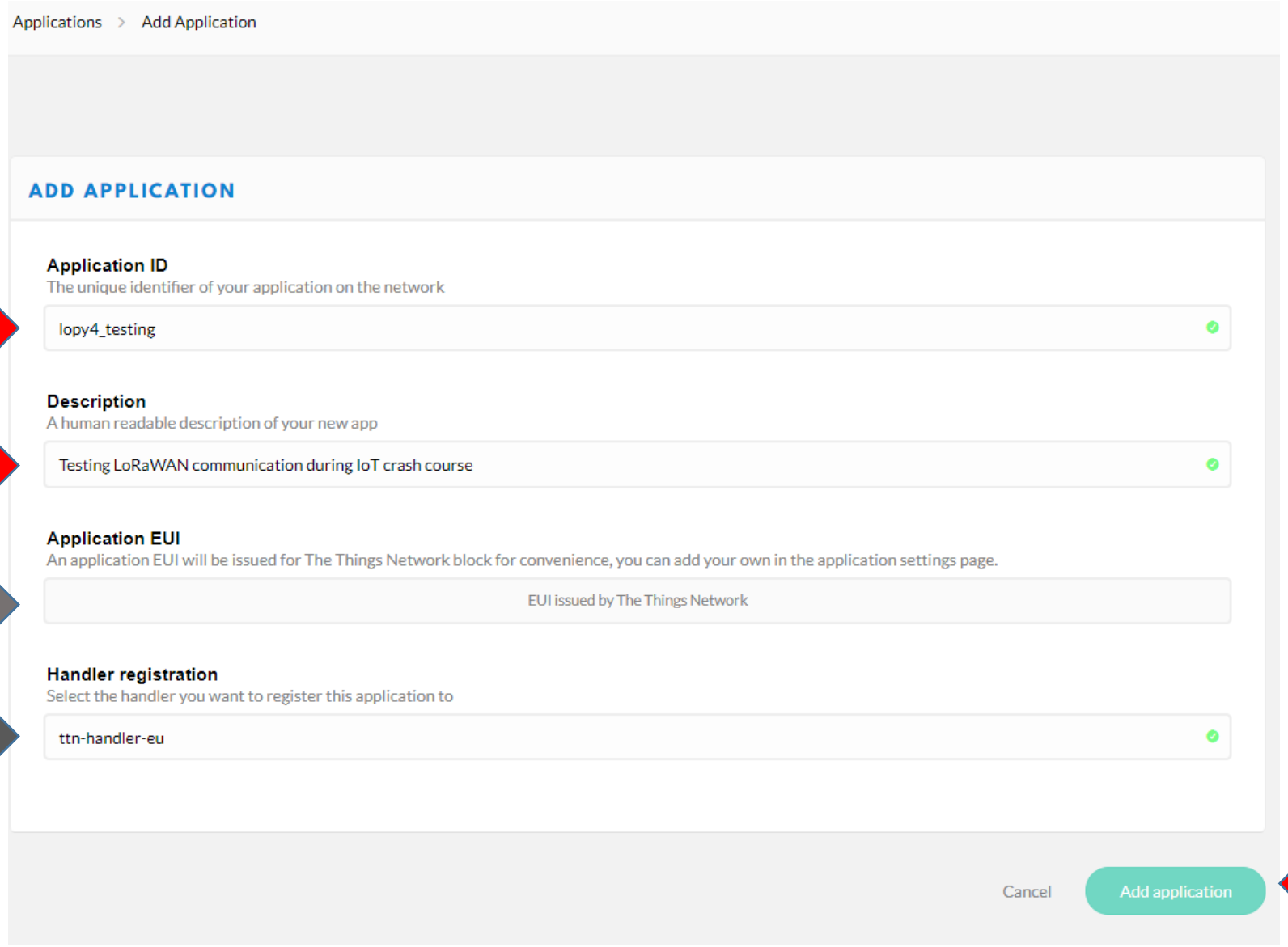
Step 3: Register an application on The Things Network

Chose an application ID (1 single word)

Add some description

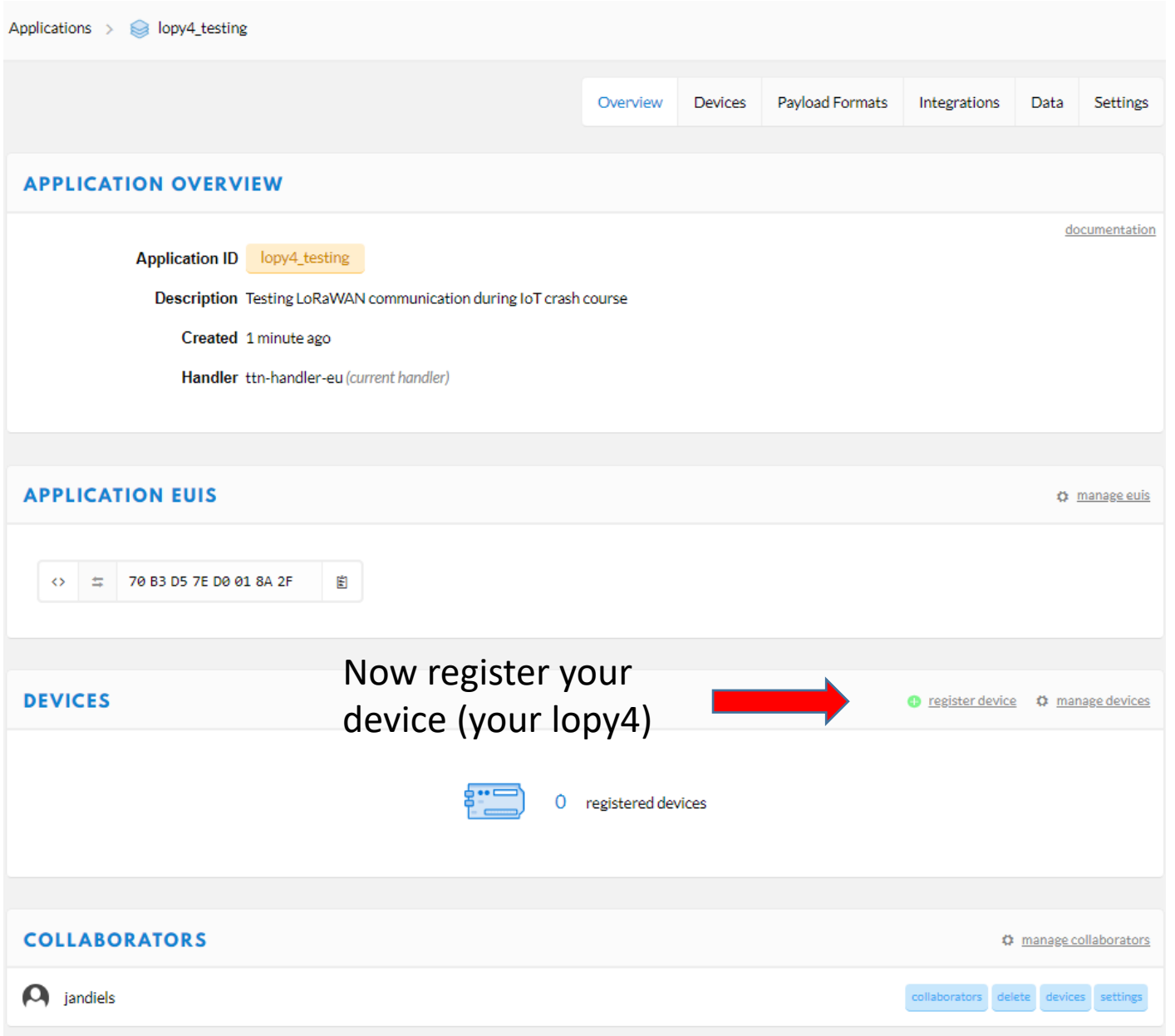
Leave blank

Don’t change



Registering your lopy4 as a device on The Things Network (TTN)

Step 3: Register your lopy4 as device on The Things Network

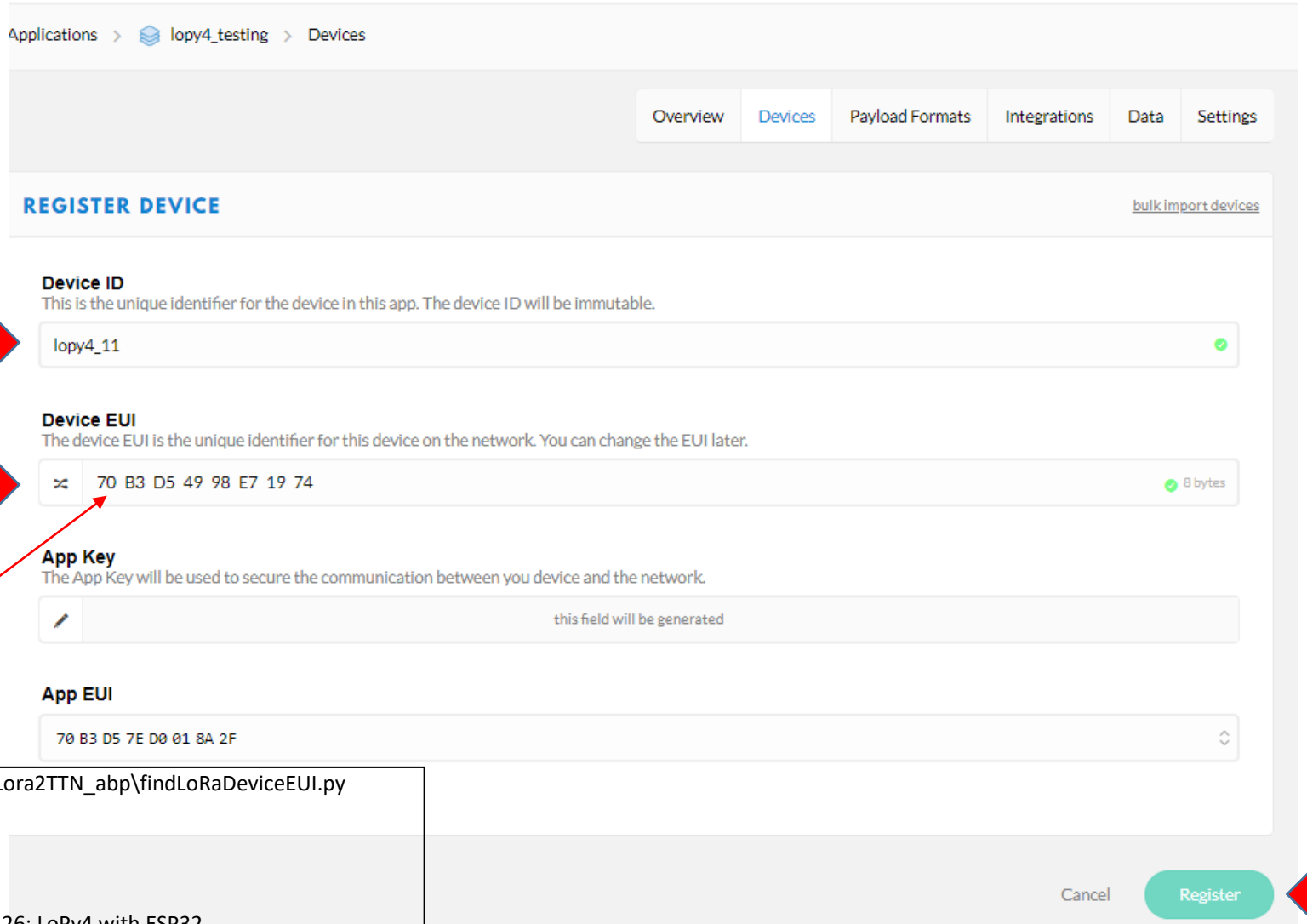


Registering your lopy4 as a device on The Things Network (TTN)

Step 3: Register your lopy4 as device on The Things Network

Chose a unique device ID (1 single word)

Copy DevEUI from Pymakr window



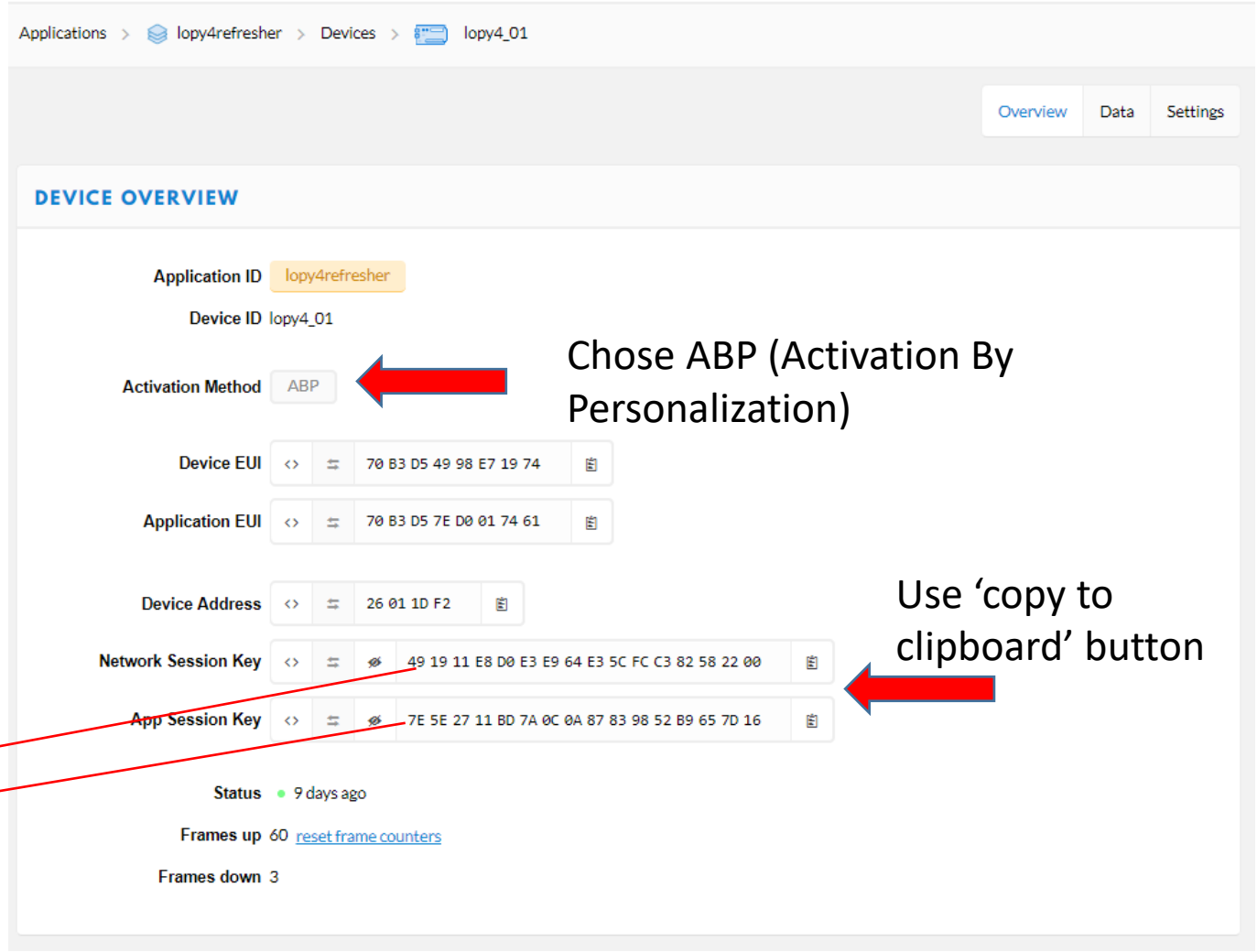

```
>>> Running C:\KULeuven\Lab\PyCom\Scripts\IoT course\SendLora2TTN_abp\findLoRaDeviceEUI.py
>>>
>>>
DevEUI: 70b3d54998e71974
>
Pycom MicroPython 1.18.2.r1 [v1.8.6-849-e0fb68e] on 2018-11-26; LoPy4 with ESP32
Type "help()" for more information.
>>>
>>>
```

Registering your lopy4 as a device on The Things Network (TTN)

Step 4: Now copy Network Session Key and App Session Key from TTN to your micropython code

main.py in folder \IoT course\SendLora2TTN_abp

```
from network import LoRa
from machine import Pin
import socket
import binascii
import struct
import ustruct
import time
import config
import pycom

# initialize LoRa in LORAWAN mode.
# Please pick the region that matches where you are using the device:
# Asia = LoRa.AS923
# Australia = LoRa.AU915
# Europe = LoRa.EU868
# United States = LoRa.US915
lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)

# create an ABP authentication params
dev_addr = struct.unpack(">l", binascii.unhexlify('26011DF2'))[0]
nwk_swkey = binascii.unhexlify('491911E8D0E3E964E35CFCC382582200')
app_swkey = binascii.unhexlify('7E5E2711BD7A0C0A87839852B9657D16')


# remove all the non-default channels
for i in range(3, 16):
    lora.remove_channel(i)

# set the 3 default channels to the same frequency
```



main.py in folder \IoT course\SendLora2TTN_abp
(code was taken from Pycom website and adapted)

```
from network import LoRa
from machine import Pin
import socket
import binascii
import struct
import ustruct
import time
import config
import pycom

# initialize LoRa in LORAWAN mode.
# Please pick the region that matches where you are using the device:
# Asia = LoRa.AS923
# Australia = LoRa.AU915
# Europe = LoRa.EU868
# United States = LoRa.US915
lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)

# create an ABP authentication params
dev_addr = struct.unpack(">l", binascii.unhexlify('26011DF2'))[0]
nwk_swkey = binascii.unhexlify('491911E8D0E3E964E35CFCC382582200')
app_swkey = binascii.unhexlify('7E5E2711BD7A0C0A87839852B9657D16')


# remove all the non-default channels
for i in range(3, 16):
    lora.remove_channel(i)

# set the 3 default channels to the same frequency
lora.add_channel(0, frequency=config.LORA_FREQUENCY, dr_min=0, dr_max=5)
lora.add_channel(1, frequency=config.LORA_FREQUENCY, dr_min=0, dr_max=5)
lora.add_channel(2, frequency=config.LORA_FREQUENCY, dr_min=0, dr_max=5)

# join a network using ABP (Activation By Personalization)
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))

# create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# set the LoRaWAN data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, config.LORA_NODE_DR)

# make the socket non-blocking
s.setblocking(False)
```

```
# create 12-bytes payload;
# '>'=big endian;
# 'b'=signed integer 1 byte = int:8; 'B'= unsigned integer 1 byte = uint:8;
# 'h'=short signed integer 2 bytes = int:16 'H'=short unsigned integer 2 bytes =
uint:16
# 'i'=long signed integer 4 bytes = int:32 'I'=long unsigned integer 4 bytes =
int:32
# 'f'=float (single precision real number) 4 bytes
# 'd'=double (double precision real number) 8 bytes
MessageNumber=0 # we need to create payload for upload but have no data
batmV=-999
hPa1=-999
hPa2=-999
hPa3=-999
soilTempCentigradeCelsius=-999
payload=ustruct.pack(">Hhhhhh",MessageNumber,batmV,hPa1,hPa2,hPa3,soilTempCenti-
gradeCelsius)

def sendpayload(payload):
    print('Sending:', payload)
    s.send(payload)
    time.sleep(4)
    rx, port = s.recvfrom(256)
    if rx:
        print('Received: {}, on port: {}'.format(rx, port))
        stats=lora.stats()
        print(stats)
        # signal successful receipt of downlink with green led for 10 seconds
        pycom.heartbeat(False) # stop the heartbeat
        pycom.rgbled(0x007f00) # green
        time.sleep(10)
        pycom.rgbled(0) # switch led off
        pycom.heartbeat(True) # start the heartbeat
    time.sleep(6)
```

Creating payload

Sending payload

Blink green led if downlink received

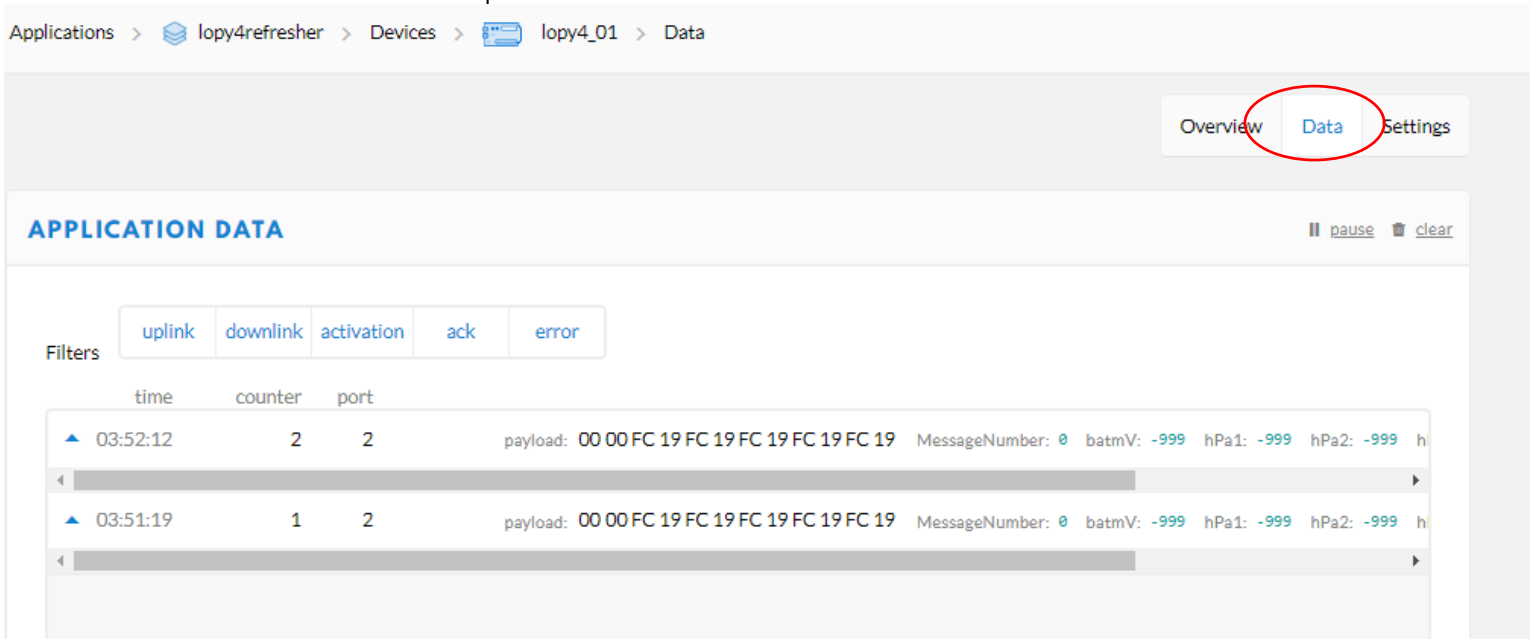
```
# initialize `P9` in gpio mode and make it an output
led=Pin('P9', mode=Pin.OUT) # user LED on expansion board
# is connected to P9

# initialize P10 in gpio mode as input with the pull-up enabled
button=Pin('P10',mode=Pin.IN, pull=Pin.PULL_UP)# user button on
# expansion board connected to P10

def pin_handler(arg):
    print("got an interrupt in pin %s" % (arg.id()))
    led.toggle()
    sendpayload(payload)

button.callback(Pin.IRQ_RISING, pin_handler) # trigger callback function when
# button is pushed. When button is pressed, voltage on P10 falls.
# When released, it rises again and triggers callback
```

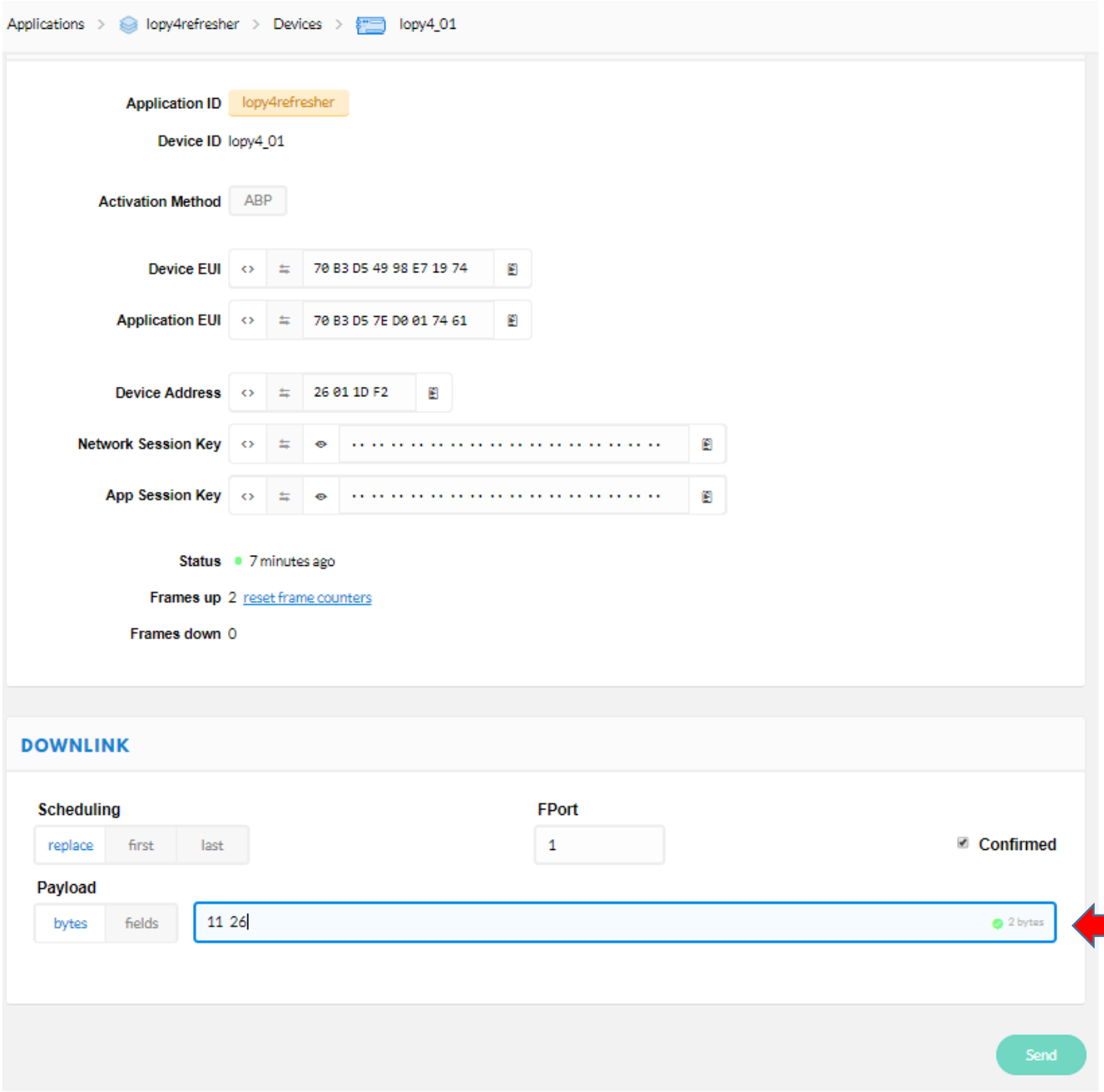
Callback function sends payload

User pushbutton triggers interrupt

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?



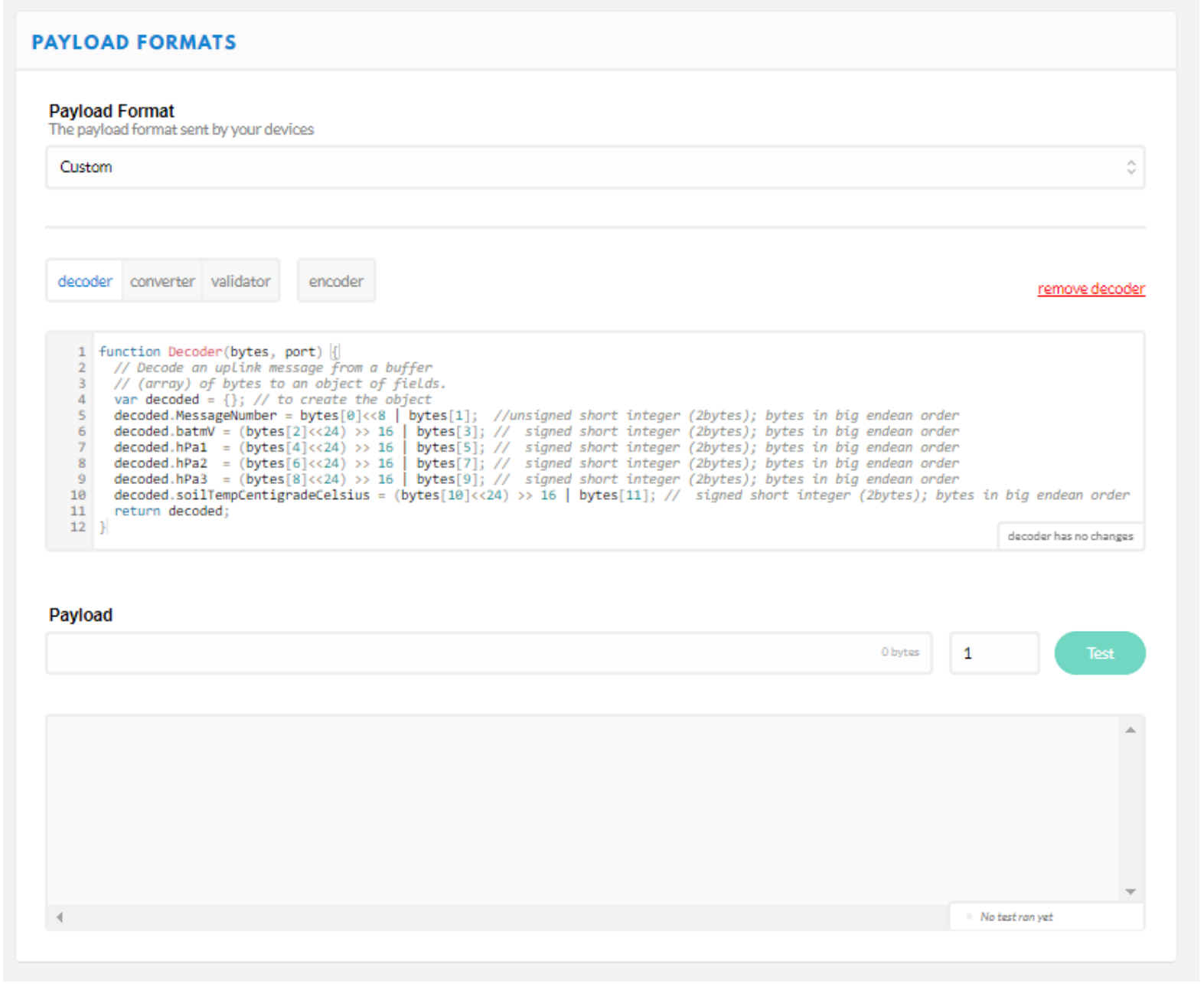


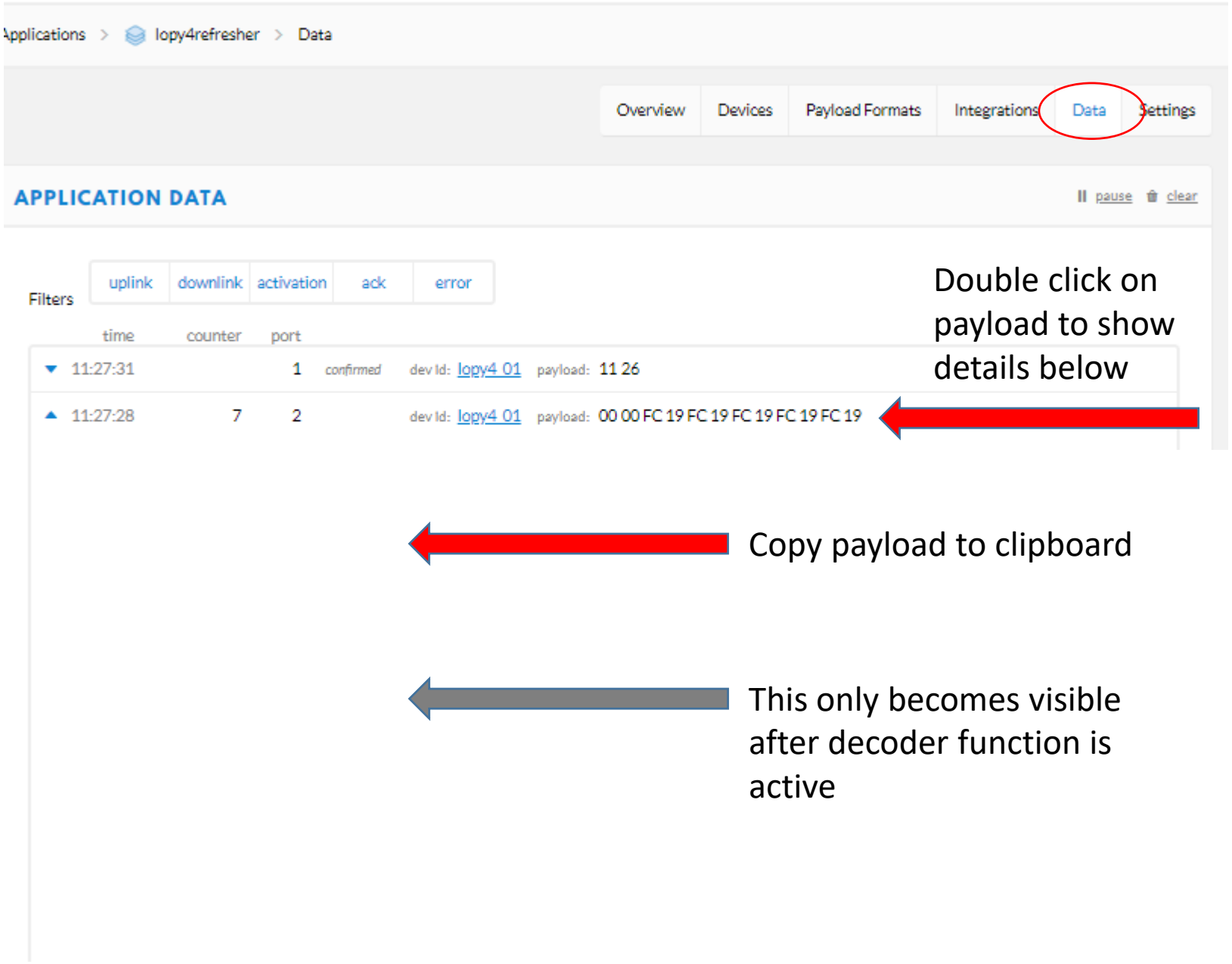
Send some nonsense data with downlink

Registering your lopy4 as a device on The Things Network (TTN)

Step 5: Decoding the payload at TTN

JavaScript code
(\IoT course\SendLora2TTN_abp\decode.js"




Applications > lopy4refresher > Data
Overview
Devices
Payload Formats
Integrations
Data
Settings
APPLICATION DATA
pause
clear
Filters
uplink
downlink
activation
ack
error
time
counter
port
11:27:31
1
confirmed
dev Id: lopy4 01
payload: 11 26
11:27:28
7
2
dev Id: lopy4 01
payload: 00 00 FC 19 FC 19 FC 19 FC 19 FC 19
Double click on payload to show details below
Copy payload to clipboard
This only becomes visible after decoder function is active

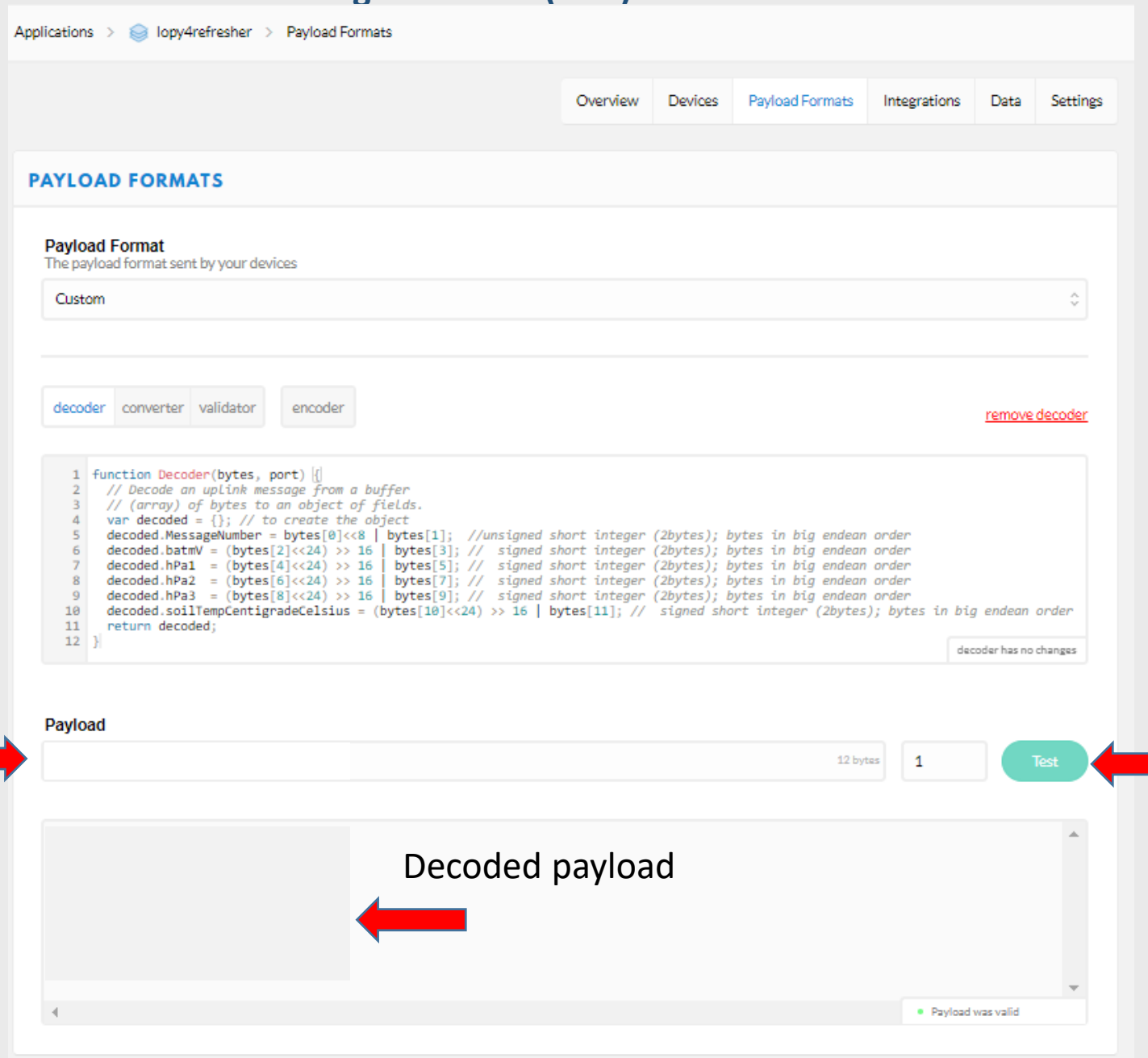
Registering your lopy4 as a device on The Things Network (TTN)

Step 5: Decoding the payload at TTN

JavaScript code

Paste payload there to test decoder

Payload = array of bytes shown in dexadecimal format



Bits and bytes

- Bit (Binary digit) can have value of 0 or 1
- Byte = 8 bits
  - Binary format: 00000000 to 11111111
    (=base 2: values can be 0 or 1)
  - Decimal format: 000 to 255
    (=base 10: values can be 0, 1, 2, …, 9)
  - Hexadecimal format: 00 to FF
    (base 16: values can be 0, 1, 2, …, 9, A, B, … F)

LoRa and Sigfox payloads are send in binary format, but we will show it in hexadecimal format because it is compact

Binary	Decimal	Hexadecimal
00000000	0	00
00000001	1	01
00000010	2	02
00000011	3	03
00000100	4	04
00000101	5	05
00000110	6	06
00000111	7	07
00001000	8	08
00001001	9	09
00001010	10	0A
00001011	11	0B
00001100	12	0C
00001101	13	0D
00001110	14	0E
00001111	15	0F
00010000	16	10
00010001	17	11
00010010	18	12
00010011	19	13
00010100	20	14
00010101	21	15
⋮	⋮	⋮
11111011	251	FB
11111100	252	FC
11111101	253	FD
11111110	254	FE
11111111	255	FF

```
# create 12-bytes payload;
# '>'=big endian;
# 'b'=signed integer 1 byte = int:8; 'B'= unsigned integer 1 byte = uint:8;
# 'h'=short signed integer 2 bytes = int:16 'H'=short unsigned integer 2 bytes = uint:16
# 'i'=long signed integer 4 bytes = int:32 'I'=long unsigned integer 4 bytes = int:32
# 'f'=float (single precision real number) 4 bytes
# 'd'=double (double precision real number) 8 bytes
MessageNumber=0 # we need to create payload for upload but have no data
batmV=-999
hPa1=-999
hPa2=-999
hPa3=-999
soilTempCentigradeCelsius=-999
payload=ustruct.pack(">Hhhhhh",MessageNumber,batmV,hPa1,hPa2,hPa3,soilTempCentigradeCelsius)
```

Encoding in micropython script

Payload

The payloa

Custom

decoder

remove decoder

```
function Decoder(bytes, port) {
  // Decode an uplink message from a buffer
  // (array) of bytes to an object of fields.
  var decoded = {}; // to create the object
  decoded.MessageNumber = bytes[0]<<8 | bytes[1]; //unsigned short integer (2bytes); bytes in big endean order
  decoded.batmV = (bytes[2]<<24) >> 16 | bytes[3]; //  signed short integer (2bytes); bytes in big endean order
  decoded.hPa1  = (bytes[4]<<24) >> 16 | bytes[5]; //  signed short integer (2bytes); bytes in big endean order
  decoded.hPa2  = (bytes[6]<<24) >> 16 | bytes[7]; //  signed short integer (2bytes); bytes in big endean order
  decoded.hPa3  = (bytes[8]<<24) >> 16 | bytes[9]; //  signed short integer (2bytes); bytes in big endean order
  decoded.soilTempCentigradeCelsius = (bytes[10]<<24) >> 16 | bytes[11]; //  signed short integer (2bytes); bytes in big endean order
  return decoded;
}
```

decoder has no changes

Decoding in decoder function at TTN

Payload

00 00 FC 19 FC 19 FC 19 FC 19 FC 19

12 bytes

1

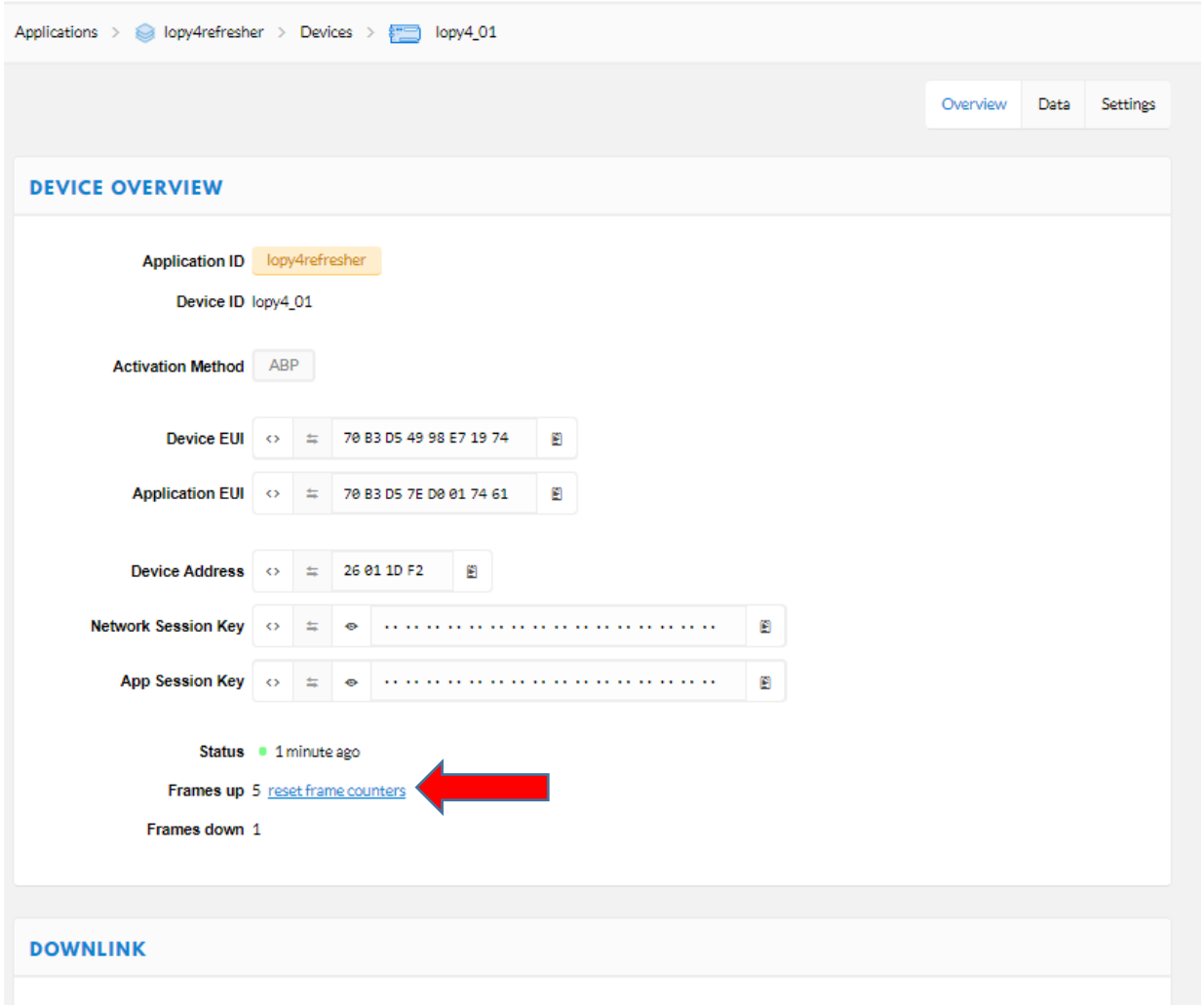
Test

Bytes[0] Bytes[1]

```
{
  "MessageNumber": 0,
  "batmV": -999,
  "hPa1": -999,
  "hPa2": -999,
  "hPa3": -999,
  "soilTempCentigradeCelsius": -999
}
```

Troubleshooting: what to do if payload is not received at The Things Network

Common problem: When your node was restarted, you also need to reset the frame counters:



Data communication we will use:

Cloud services

LoRaWAN

HTTP via internet

THE THINGS NETWORK

Node-RED

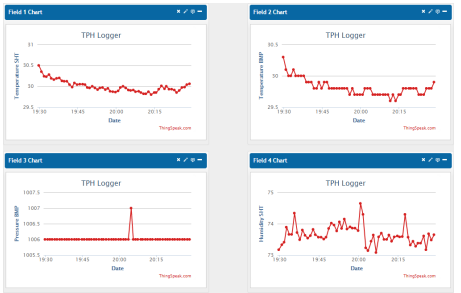
HTTP via internet

ThingSpeak

MATLAB

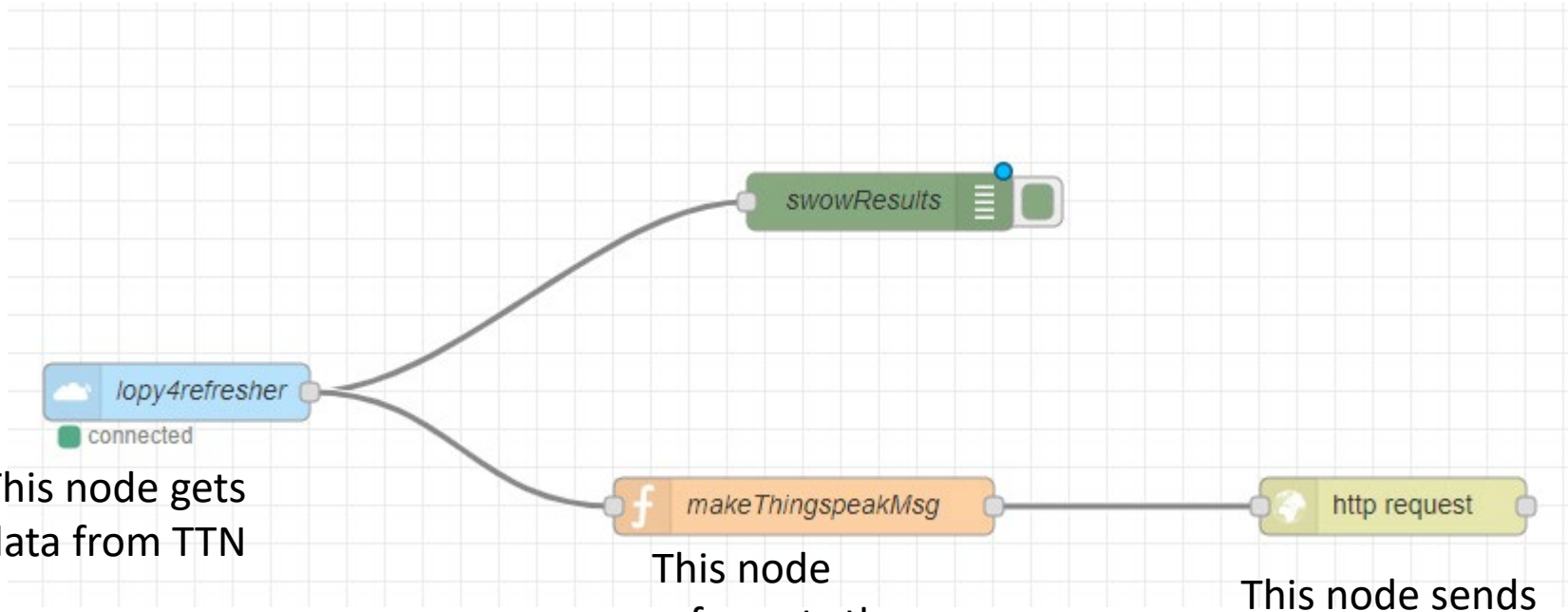
Your lopy4

To reformat the data for ThingSpeak



Visualization, quality control, storage, analysis

Node-RED

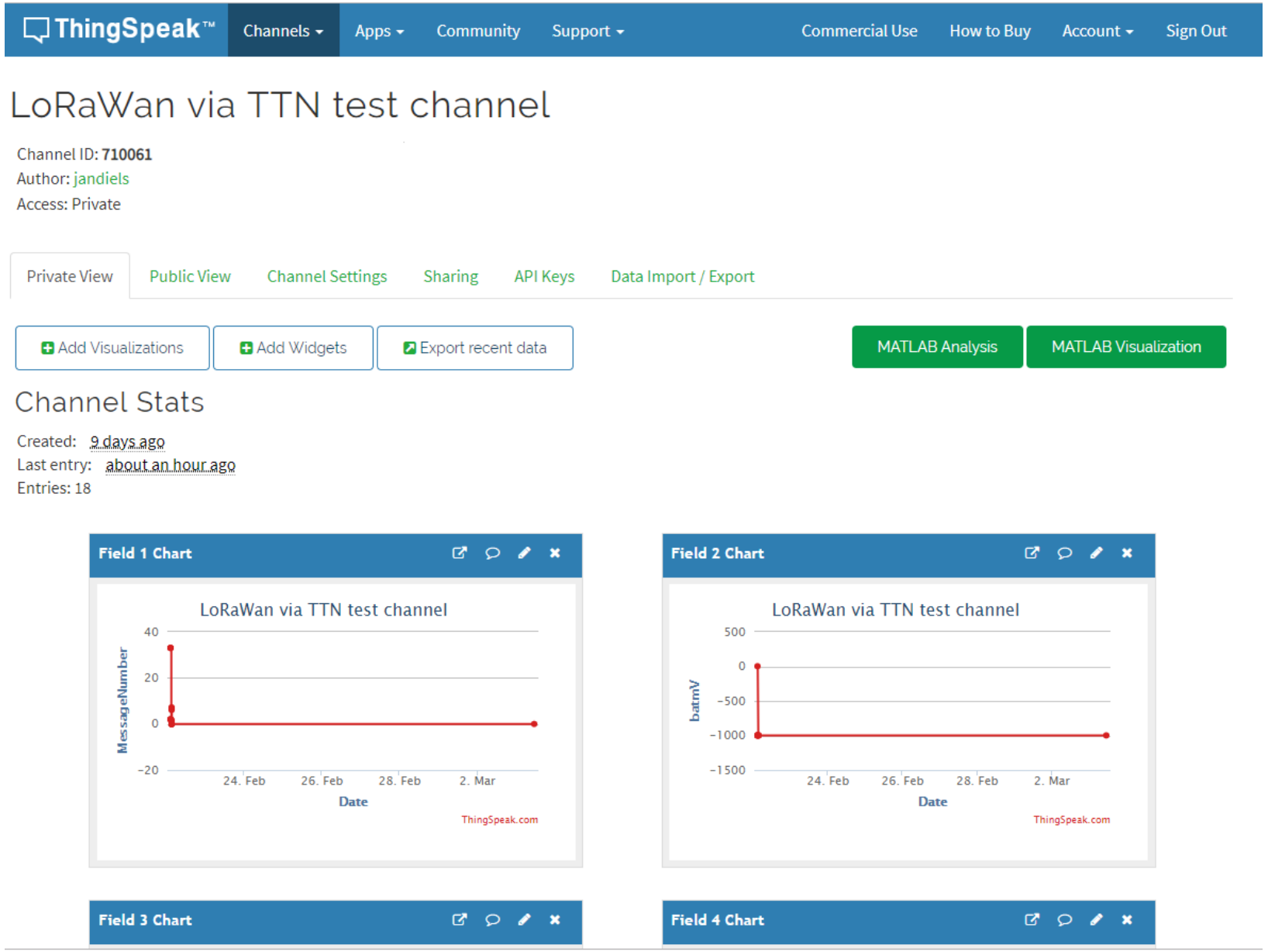


This node gets data from TTN

This node reformats the data for ThingSpeak

This node sends data to ThingSpeak in http format

More information: https://www.npmjs.com/package/node-red-contrib-ttn

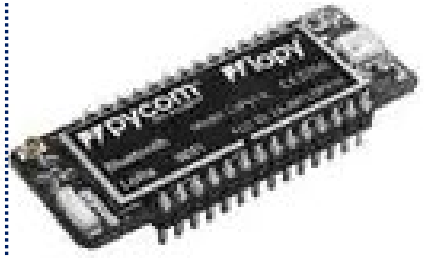

ThingSpeak™
Channels
Apps
Community
Support
Commercial Use
How to Buy
Account
Sign Out
LoRaWan via TTN test channel
Channel ID: 710061
Author: jandiels
Access: Private
Private View
Public View
Channel Settings
Sharing
API Keys
Data Import / Export
Add Visualizations
Add Widgets
Export recent data
MATLAB Analysis
MATLAB Visualization
Channel Stats
Created: 9 days ago
Last entry: about an hour ago
Entries: 18
Field 1 Chart
LoRaWan via TTN test channel
MessageNumber
40
20
0
-20
24. Feb
26. Feb
28. Feb
2. Mar
Date
ThingSpeak.com
Field 2 Chart
LoRaWan via TTN test channel
batmV
500
0
-500
-1000
-1500
24. Feb
26. Feb
28. Feb
2. Mar
Date
ThingSpeak.com
Field 3 Chart
Field 4 Chart

IoT project I am working on:

Inexpensive device for monitoring soil moisture content in irrigated field

3 × Watermark sensor (Irrometer Company)

1 × soil temperature sensor (DS18B20)



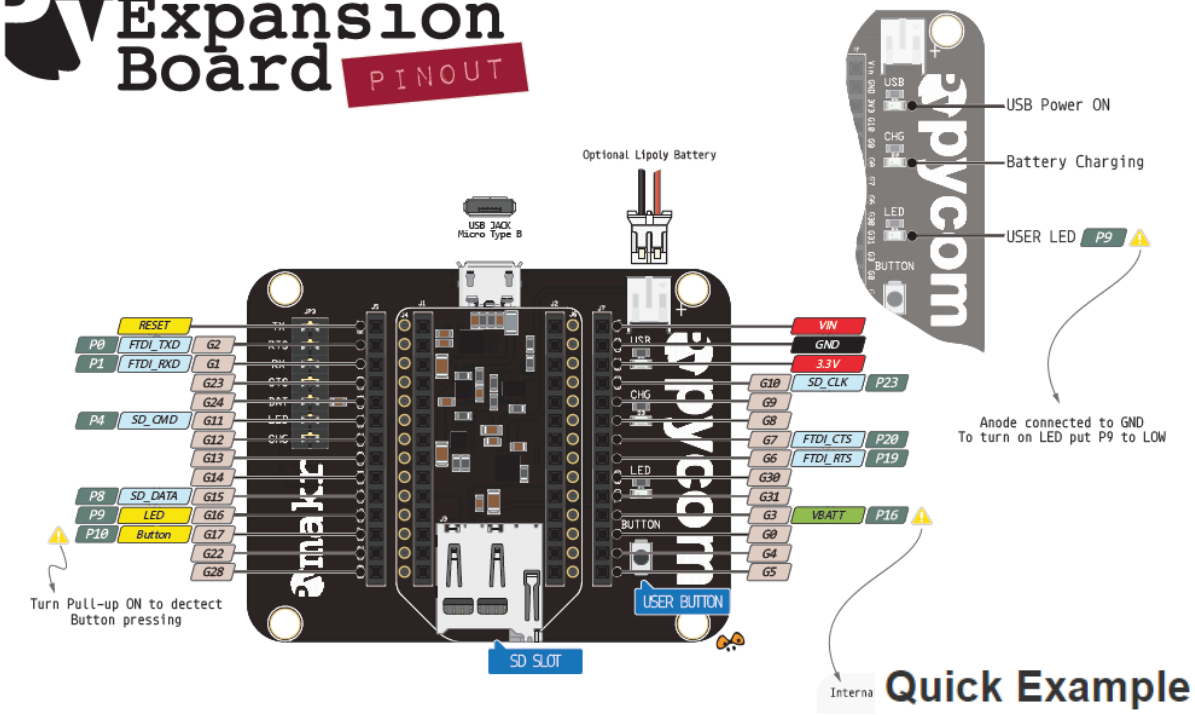
1 × LoPy[4] development board (µcontroller with LoRaWAN and Sigfox capability)

1 × antenna

1 × LiPo battery 3.7V 2000mAh

3 watermark sensors	€ 138.00
1 DS18B20 temperature sensor	€ 9.95
LoPy[4] µcontroller	€ 32.19
Antenna with IP67 cable	€ 12.00
IP67 enclosure	€ 14.95
LiPo battery 2000mAh	€ 12.95
Board with connectors and passive components	€ 30.00
Total (without VAT):	**€ 250.04**

Expansion Board PINOUT



Quick Example Usage:

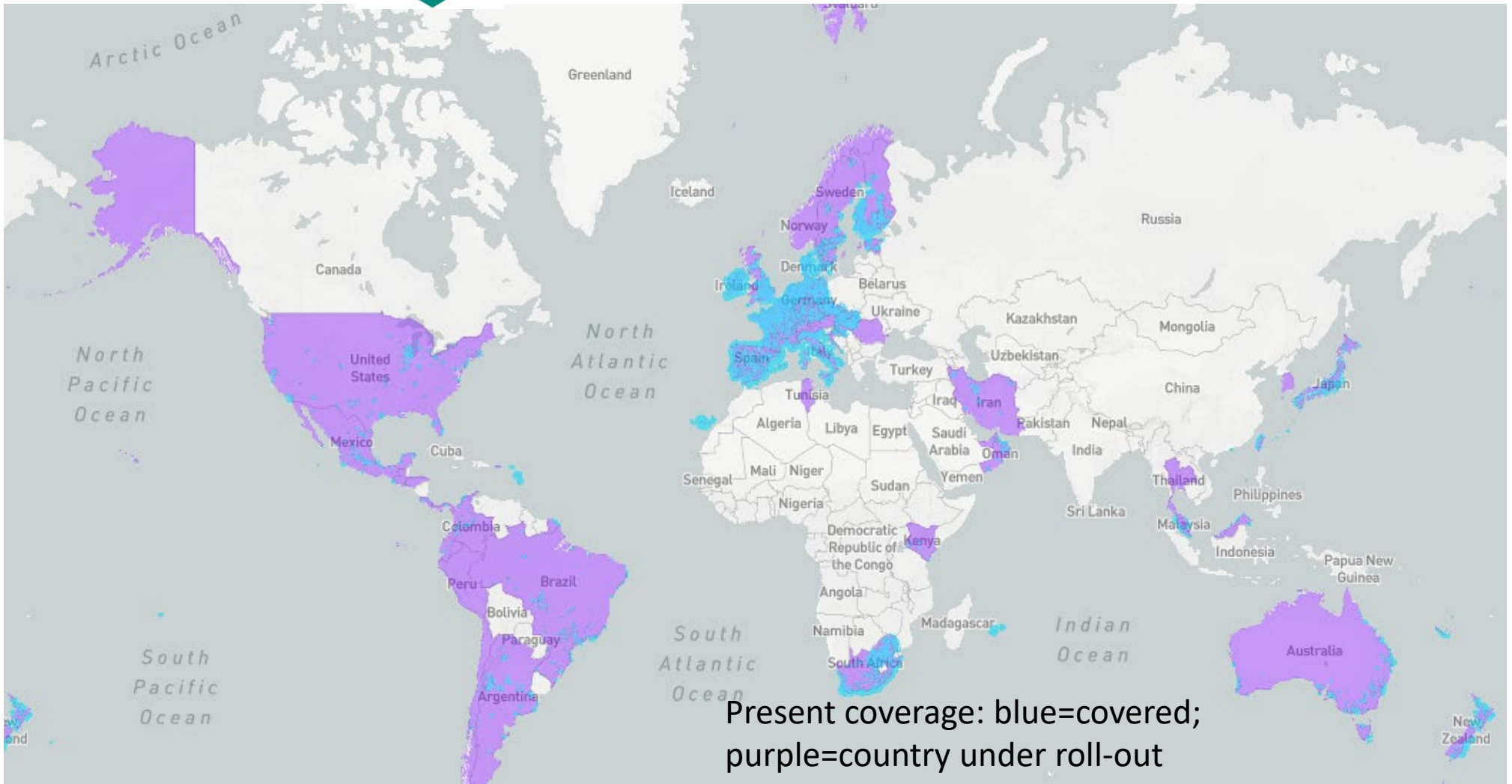
```
from machine import SD
import os

sd = SD()
os.mount(sd, '/sd')

# check the content
os.listdir('/sd')

# try some standard file operations
f = open('/sd/test.txt', 'w')
f.write('Testing SD card write operations')
f.close()
f = open('/sd/test.txt', 'r')
f.readall()
f.close()
```

It is best to also log the measured data on the lopy4 using an SD card (slot on expansion board)

Pycom provides library for accessing the SD card

sigfox

With LoPy4 you get 1 year Sigfox connectivity included in price.



Present coverage: blue=covered; purple=country under roll-out

Useful links:

Two excellent explanation video's about LoRaWAN, TTN, TTN gateway and TTN node: https://www.thethingsnetwork.org/community/berlin/post/lpwan-ttn-2-very-good-explanation-videos

Encoding and Decoding Payloads on The Things Network: https://core-electronics.com.au/tutorials/encoding-and-decoding-payloads-on-the-things-network.html

Explaining LoRaWAN: https://ubidots.com/blog/explaining-lorawan/

Slides lora-aliance: https://www.ietf.org/proceedings/96/slides/slides-96-lpwan-9.pptx

Day 2: Teams of 4-5 participants develop and test an application for hydrological measurements

Three teams will work on the following 3 projects:

- Monitor water depth in a river or channel or reservoir with an ultrasonic water level sensor (MaxBotix)
- Measure air temperature (DS18B20 temperature sensor) and rainfall (tipping-bucket rain gauge)
- Measure soil temperature (DS18B20 temperature sensor) and soil moisture content (Watermark sensors)

At the end of the day, each team presents their application to the entire group. Prototype schematics and MicroPython code for the 3 applications is provided to all participants.

Day 2 **9.00 am to 10.30**	• Teams study the available documentation, make a workplan for completing the project, and agree on who is going to work on which subtask
Day 2 **11.00 am to 12.30**	• Developing and testing code and hardware for project components (sensor reading, LoRa communication, measurement loop, deepsleep, …)
Day 2 **2.00 pm to 3.30**	• Developing and testing code and hardware for project components (continued)
Day 2 **4.00 pm to 5.30**	• Integration of components into (prototype) application and testing • Each team presents their application to the entire group

Costs of sensors in Belgium (VAT included)?

Sonar MB7052 for water depth: 95.15€

Watermark sensor: 56€

DS18B20 temperature sensor: 9.95€

Davis tipping-bucket raingauge 6463M : 131€

Solar radiation shield for temperature sensor (air temperature): 126€