Prerequisites:

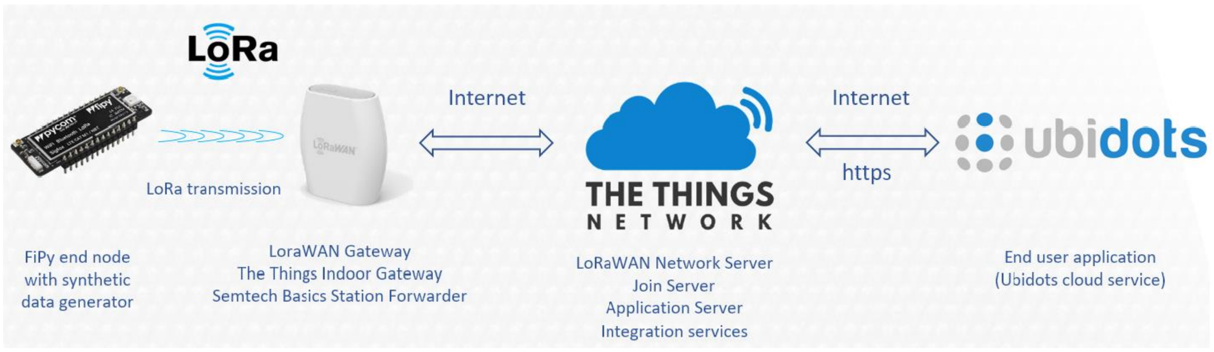
A. Download and install Atom Text Editor (https://atom.io) and add Pymakr plugin.

B. Set up your account on the Ubidots cloud service (http://ubidots.com). Choose free account type for educational or personal use.

C. Sep up your account on The Things Network (http://thethingsnetwork.com)

Part 1: Sending synthetic data to the Ubidots cloud through LoRaWAN The Things Network

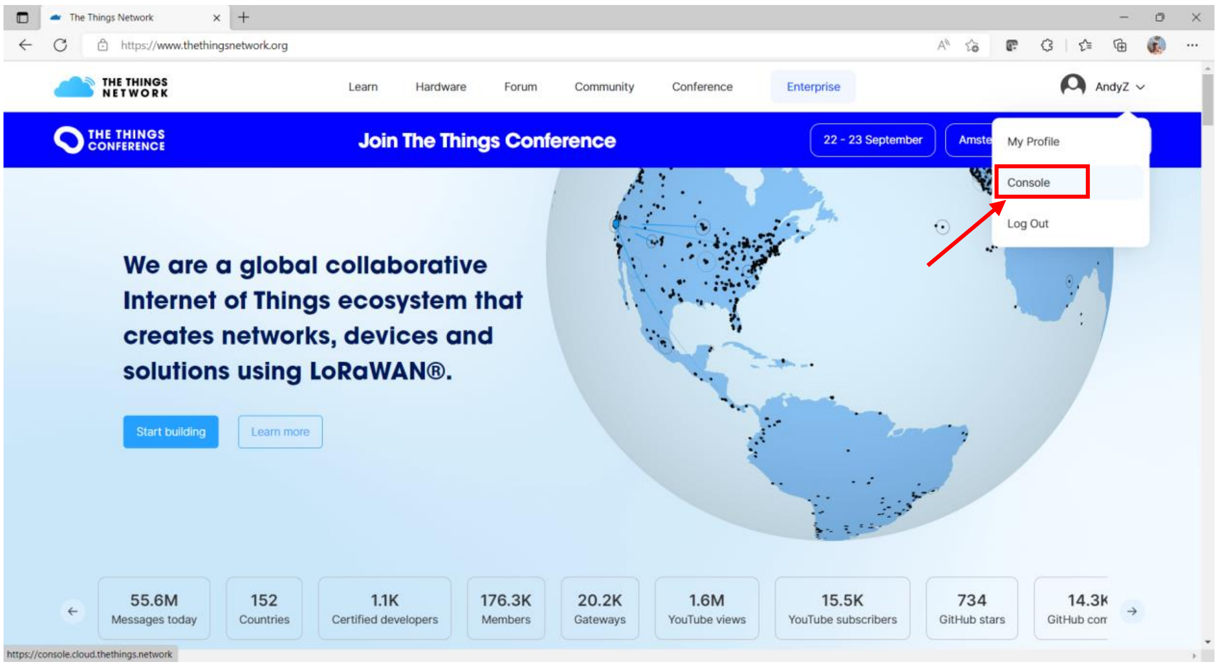
In this point you will create IoT application that generates synthetic data (in the form of the sinewave) and sends it to the Ubidots clouds service using LoRaWAN The Things Network. The figure below shows the block structure of this application.



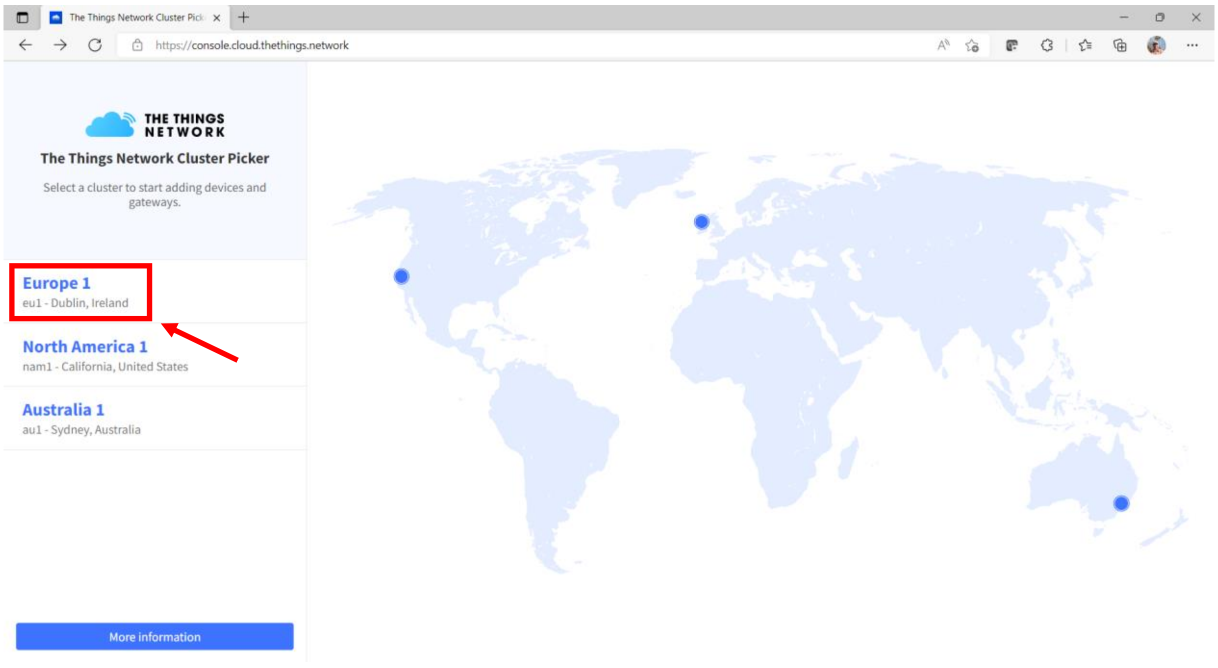
1.1 Plug FiPy module to PC using USB port. Open Atom IDE and using the following commands read LoRa EUI of the module. Write it down for future use.

```
from network import LoRa
import binascii
print(binascii.hexlify(LoRa().mac()).upper())
```

1.2 Log into your TTN account and go to the Console



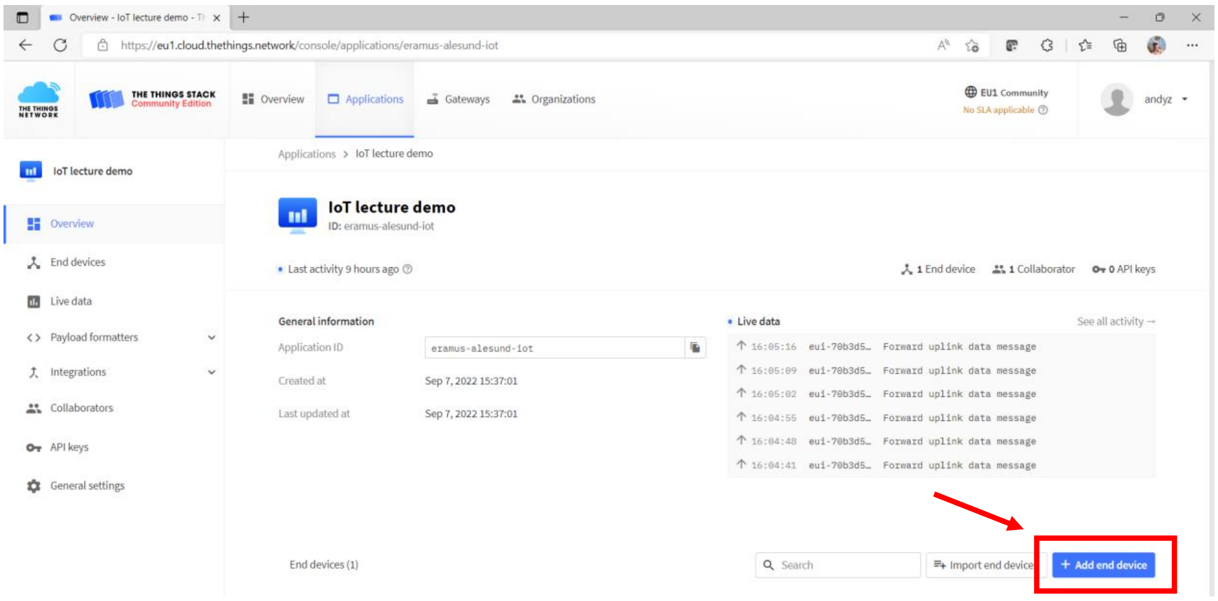
1.3 Choose Europe 1 server and then go to “Applications” option.


The Things Network Cluster Pick
https://console.cloud.thethings.network
THE THINGS NETWORK
The Things Network Cluster Picker
Select a cluster to start adding devices and gateways.
Europe 1
eu1 - Dublin, Ireland
North America 1
nam1 - California, United States
Australia 1
au1 - Sydney, Australia
More information

1.4 Click “+ Add application button”

1.5 Set your Application ID (must be unique within the whole TTN). Optionally you can fill “Application name” and “Description” fields.

1.6 In created application select “Add end node”



IoT lecture demo
ID: eramus-alesund-iot

Last activity 9 hours ago — 1 End device — 1 Collaborator — 0 API keys

General information

Application ID	eramus-alesund-iot
Created at	Sep 7, 2022 15:37:01
Last updated at	Sep 7, 2022 15:37:01

Live data

```
↑ 16:05:16  eui-70b3d5…  Forward uplink data message
↑ 16:05:09  eui-70b3d5…  Forward uplink data message
↑ 16:05:02  eui-70b3d5…  Forward uplink data message
↑ 16:04:55  eui-70b3d5…  Forward uplink data message
↑ 16:04:48  eui-70b3d5…  Forward uplink data message
↑ 16:04:41  eui-70b3d5…  Forward uplink data message
```

1.7 On the “**Register end device**” page choose “**Manual**” tab and then select the following options:

Frequency plan: Europe 863-870 MHz (SF9 fo RX2)

LoRaWAN version: 1.0.2

Regional Parameters version: RP001 Regional Parameters revision B

DevEUI: value read from FiPy module

AppEUI: 0000000000000005

AppKey: click “Generate” button

Finally click “**Register end device**” button.

1.8 Copy generated **AppKey** to the clipboard

1.9 In Atom environment open folder with “NTNU_IoT_Guest_Course” application and double click “main.py” file in the list om the left side.

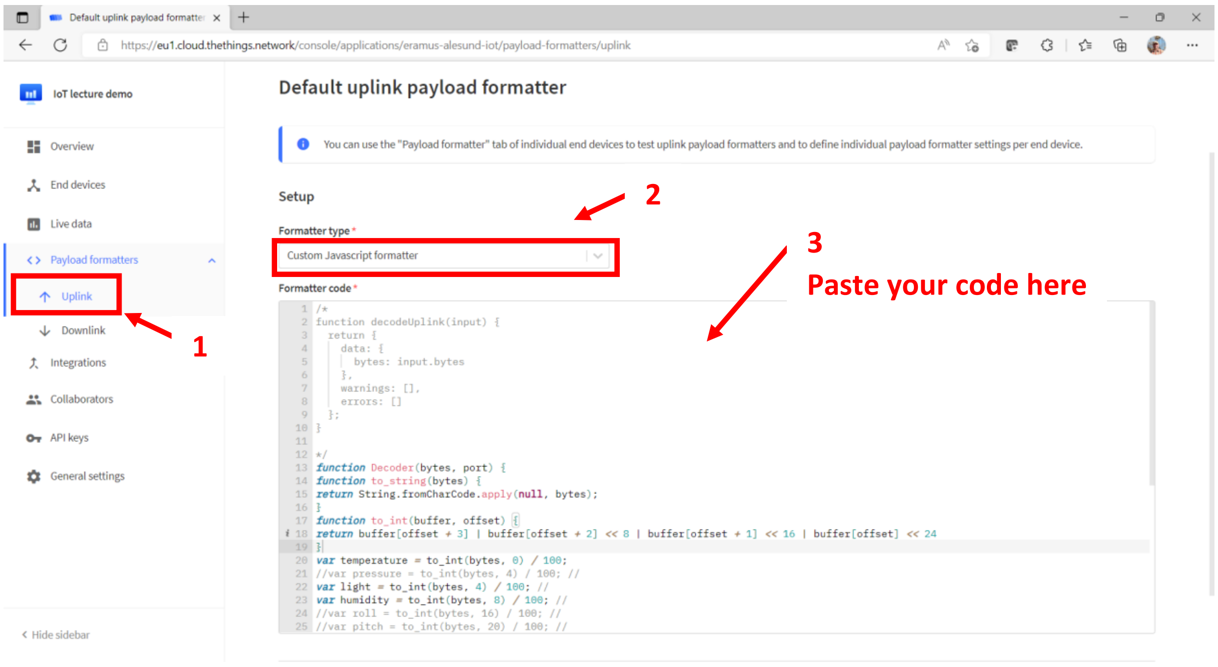
1.10 Look through the opened code and put your values of **AppEUI** and **AppKey** in the corresponding variables in the code.

1.11 Save modified code (Ctr+-S) and upload it to the FiPy module.

1.12 Return to TTN console and in your device page choose “Live data” tab.

1.13 Select one of data messages and review its content. Especially identify “frm_payload field with application uplink data written using Base64 encoding.

1.14. In order to automatically decode uplink payload add to your TTN application a decoder script from file “TTN_decoder.txt”.



Default uplink payload formatter

You can use the "Payload formatter" tab of individual end devices to test uplink payload formatters and to define individual payload formatter settings per end device.

Setup

Formatter type *: Custom Javascript formatter

Formatter code *

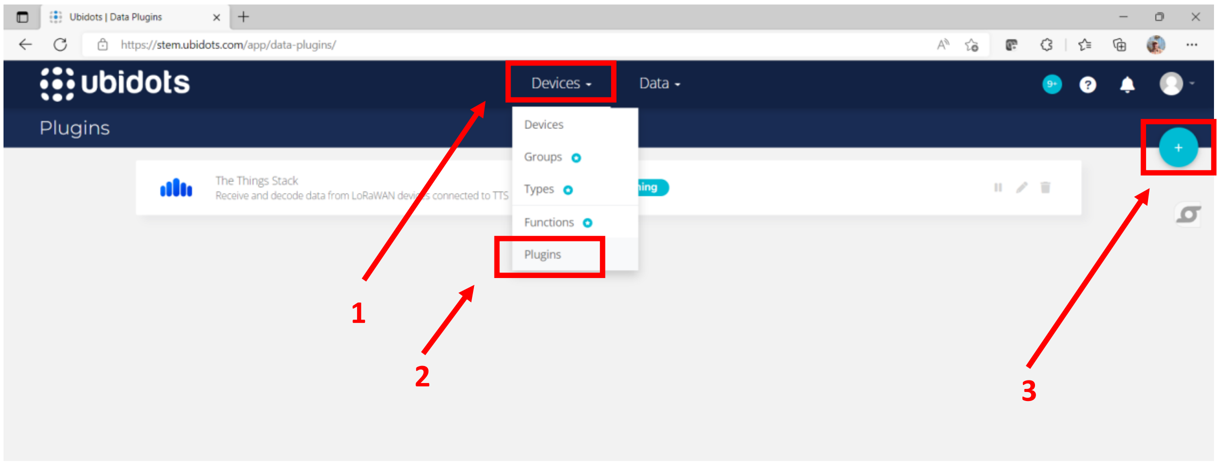
```
/*
function decodeUplink(input) {
  return {
    data: {
      bytes: input.bytes
    },
    warnings: [],
    errors: []
  };
}

*/
function Decoder(bytes, port) {
function to_string(bytes) {
return String.fromCharCode.apply(null, bytes);
}
function to_int(buffer, offset) {
return buffer[offset + 3] | buffer[offset + 2] << 8 | buffer[offset + 1] << 16 | buffer[offset] << 24
}
var temperature = to_int(bytes, 0) / 100;
//var pressure = to_int(bytes, 4) / 100; //
var light = to_int(bytes, 4) / 100; //
var humidity = to_int(bytes, 8) / 100; //
//var roll = to_int(bytes, 16) / 100; //
//var pitch = to_int(bytes, 20) / 100; //
```

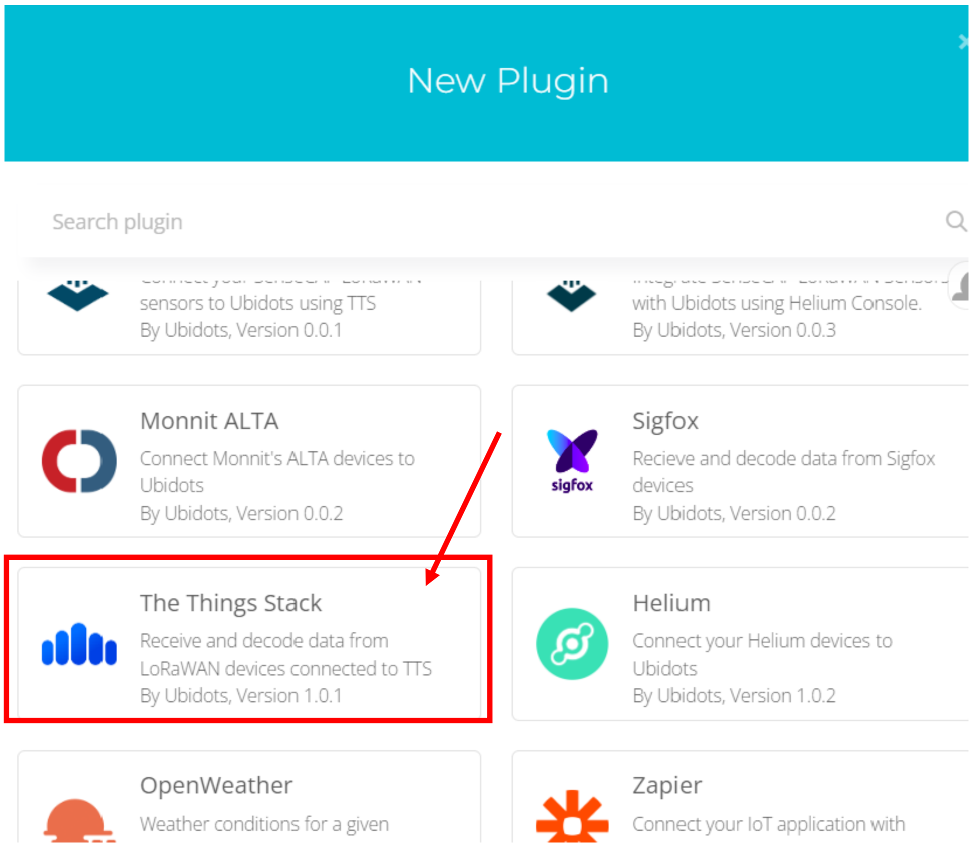
1.15 Click “**Save changes**” button below the script window.

1.16 Go to TTN console again and check if new uplink messages have included “decoded_payload” field with your app data.

1.17 Log into your Ubidots account and on the top of the page main chose “Device” → “Plugins”, then “+” circle on the right.


Ubidots | Data Plugins
https://stem.ubidots.com/app/data-plugins/
ubidots
Devices
Data
Plugins
Devices
Groups
Types
Functions
Plugins
The Things Stack
Receive and decode data from LoRaWAN devices connected to TTS
+
1
2
3

1.18 In “New Plugin” windows select “The ThingsStack”. In plugin creation windows select “Default token” in the “Ubidots Token” field.


New Plugin

Search plugin

sensors to Ubidots using TTS
By Ubidots, Version 0.0.1

with Ubidots using Helium Console.
By Ubidots, Version 0.0.3

Monnit ALTA
Connect Monnit's ALTA devices to Ubidots
By Ubidots, Version 0.0.2

Sigfox
Recieve and decode data from Sigfox devices
By Ubidots, Version 0.0.2

The Things Stack
Receive and decode data from LoRaWAN devices connected to TTS
By Ubidots, Version 1.0.1

Helium
Connect your Helium devices to Ubidots
By Ubidots, Version 1.0.2

OpenWeather
Weather conditions for a given

Zapier
Connect your IoT application with

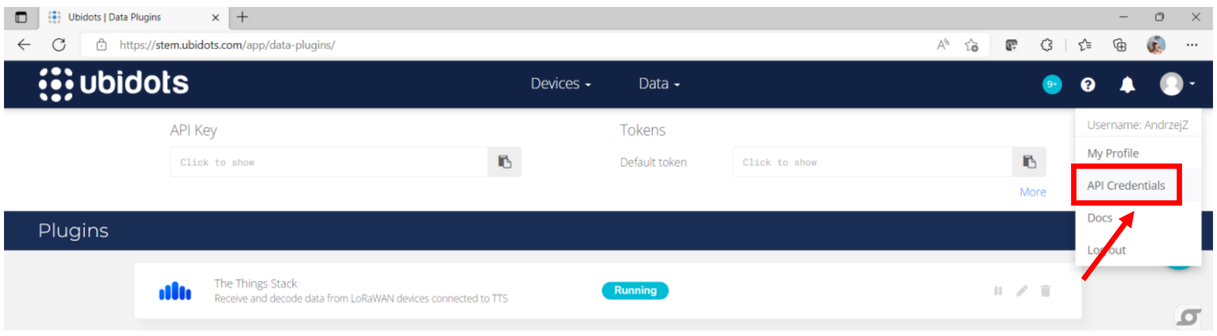
1.19 After creating TTS plugin locate it on the list of installed plugins (probably there will be only this one), click on the "Edit" button, then go to "Decoder", and copy the HTTPs Endpoint URL.

1.20 Go to your TTN account, choose your application, then "Integrations" → "Webhooks" → "Add webhook" → "Custom webhook".

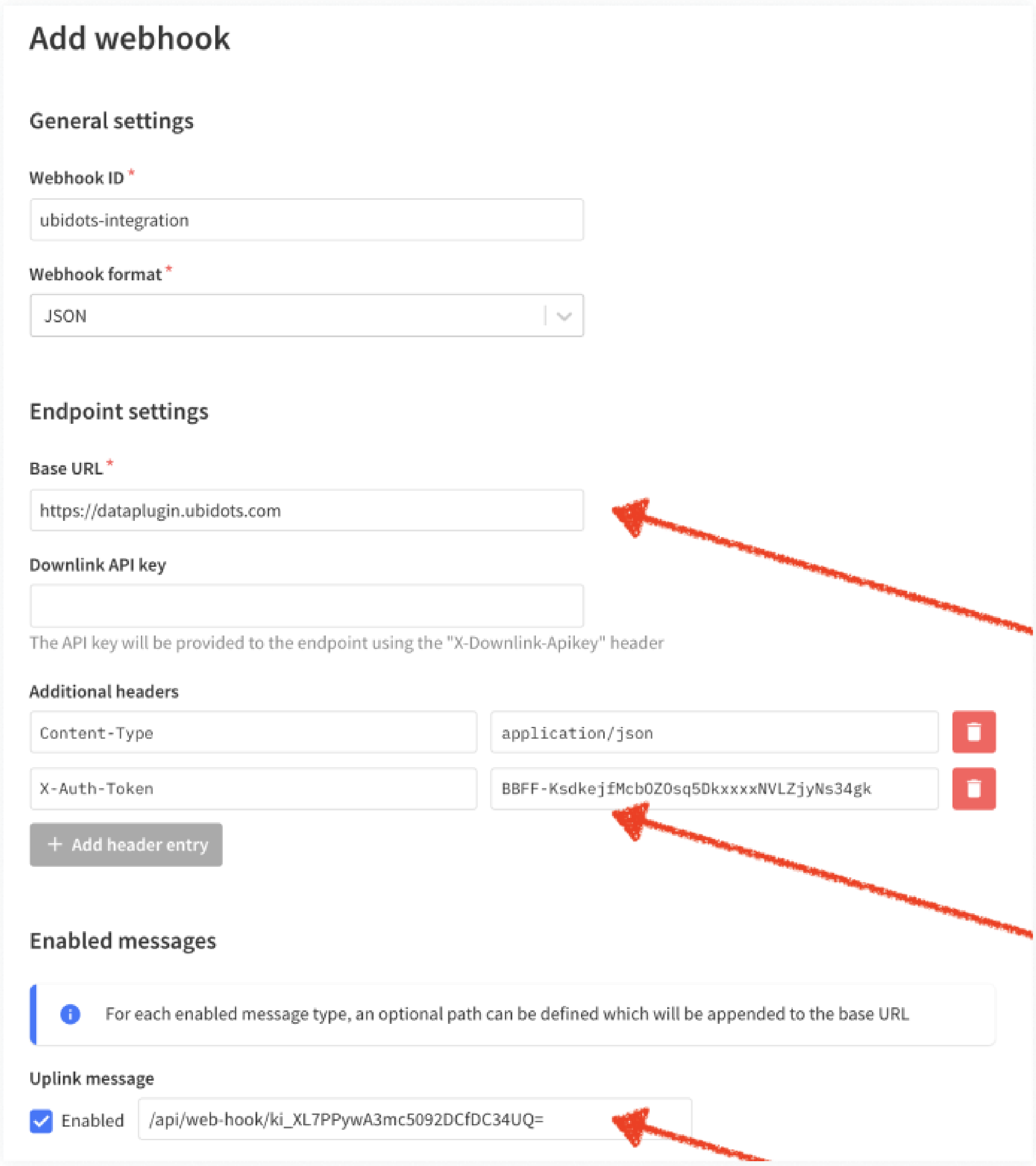
1.21 Choose a unique name for your webhook and fill the following fields:

Base URL: https://dataplugin.ubidots.com

Additional headers: Add an "X-Auth-Token" header with a valid Ubidots token (you can read it from API Credentials option in your Ubidots account.



Uplink message: Mark this checkbox and enter the remaining portion of your Plugin's HTTPs endpoint URL (read in point 1.19), for instance: `/api/web-hook/ki_XL7PPywA3mc5092DCfDC34UQ=`



Add webhook

General settings

Webhook ID *

ubidots-integration

Webhook format *

JSON

Endpoint settings

Base URL *

https://dataplugin.ubidots.com

Downlink API key

The API key will be provided to the endpoint using the "X-Downlink-Apikey" header

Additional headers

Content-Type	application/json
X-Auth-Token	BBFF-KsdkejfMcbOZOsq5DkxxxxNVLZjyNs34gk

Enabled messages

For each enabled message type, an optional path can be defined which will be appended to the base URL

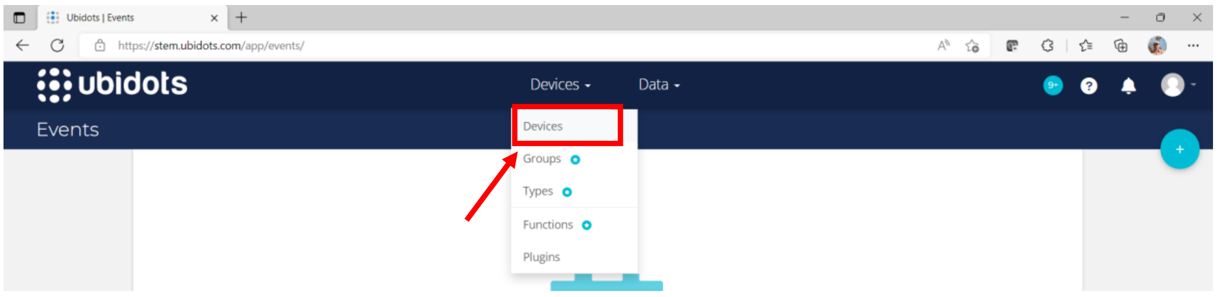
Uplink message

☑ Enabled /api/web-hook/ki_XL7PPywA3mc5092DCfDC34UQ=

1.22 Back to Ubidots, edit the Decoding Function shown in the Decoder section of the plugin to match your customized payload. Change default sample decoder for script from Ubidots_decoder.txt file.

1.23 Run your FiPy application (in Atom).

1.24 In Ubidots choose devices option then select your device.



1.25 Check if data from your application is properly logged in Ubidots service.

1.26 In Ubidots choose "Data" → "Dashboards" and add selected widget (e.g. Gauge) to your dashboard. In widget configuration you must select variable from list of variables received by Ubidots from your FiPy application.

1.27 Observe if widget properly receives and shows your data.

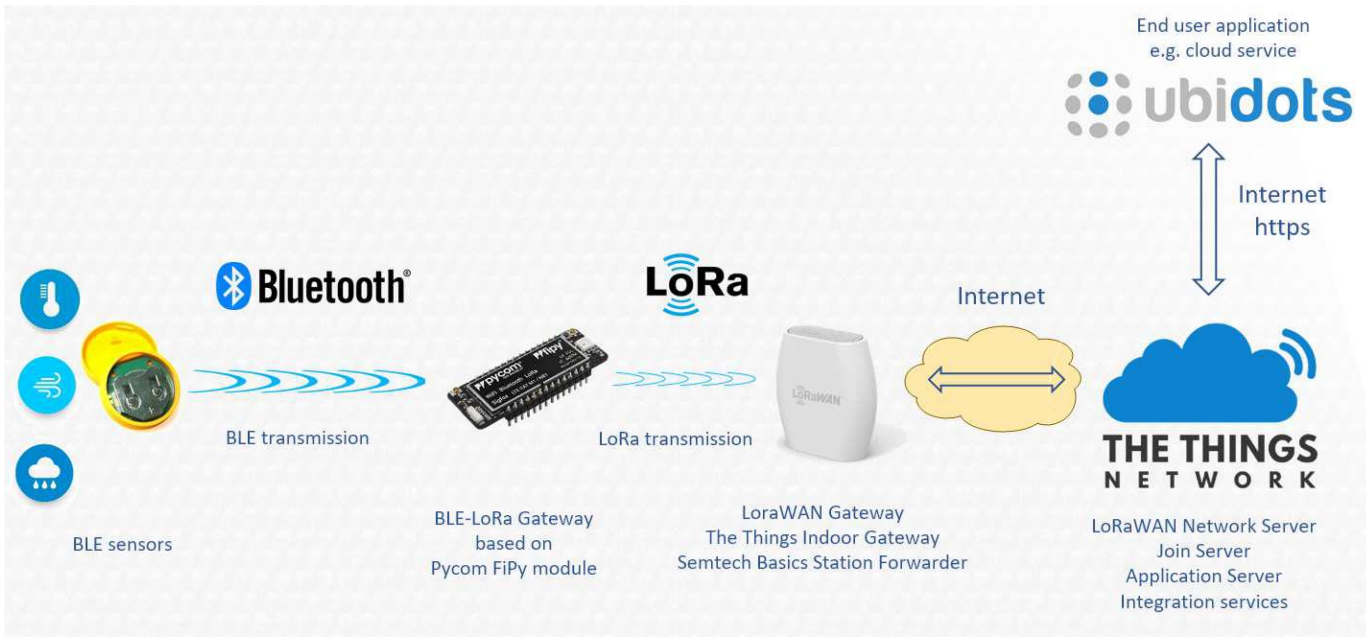
1.28 Modify FiPy application to increase frequency of generated sinewave.

1.28 Modify FiPy application in order to add new variable sent to Ubidots cloud. For instance you can add new variable similar to existed one (sinewave) but with cosinus function instead of sinus. Or you can try to work out your own concept.

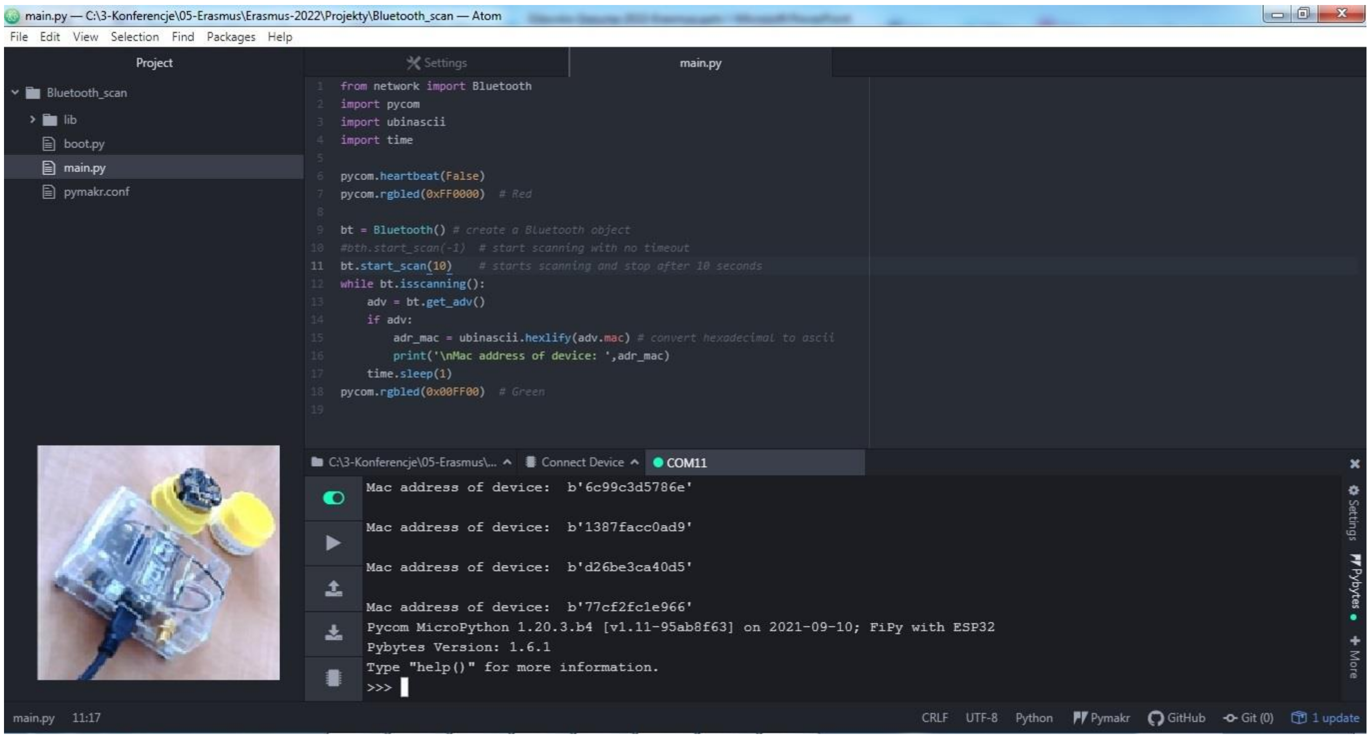
Be aware that in order to have your new variable sent and decoded properly you need to modify definition of the `user_data` structure in FiPy application as well as you have to introduce a related change in TTN decoder script. Ubidots decoder does no need to be modified.

Part 2: Sending data from BLE sensor to the Ubidots cloud through LoRaWAN The Things Network

In this point you will enrich functionality of FiPy application created in Part 1. In addition to sending synthetic data application will send also real measurement data received from BLE sensor. The figure below shows the block structure of the enhanced application.ing:



2.1. Scan the environment and read MAC addresses of BLE devices sending advertisements.


main.py — C:\3-Konferencje\05-Erasmus\Erasmus-2022\Projekty\Bluetooth_scan — Atom
File Edit View Selection Find Packages Help
Project
Bluetooth_scan
lib
boot.py
main.py
pymakr.conf
Settings
main.py
C:\3-Konferencje\05-Erasmus\...
Connect Device
COM11
main.py 11:17
CRLF UTF-8 Python Pymakr GitHub Git (0) 1 update

```python
from network import Bluetooth
import pycom
import ubinascii
import time

pycom.heartbeat(False)
pycom.rgbled(0xFF0000)  # Red

bt = Bluetooth() # create a Bluetooth object
#bth.start_scan(-1)  # start scanning with no timeout
bt.start_scan(10)    # starts scanning and stop after 10 seconds
while bt.isscanning():
    adv = bt.get_adv()
    if adv:
        adr_mac = ubinascii.hexlify(adv.mac) # convert hexadecimal to ascii
        print('\nMac address of device: ',adr_mac)
    time.sleep(1)
pycom.rgbled(0x00FF00)  # Green

```

```
Mac address of device:  b'6c99c3d5786e'

Mac address of device:  b'1387facc0ad9'

Mac address of device:  b'd26be3ca40d5'

Mac address of device:  b'77cf2fc1e966'
Pycom MicroPython 1.20.3.b4 [v1.11-95ab8f63] on 2021-09-10; FiPy with ESP32
Pybytes Version: 1.6.1
Type "help()" for more information.
>>>
```

Use the following commands (file Bluetooth1.txt):

```
from network import Bluetooth
import pycom
import ubinascii
import time

pycom.heartbeat(False)
pycom.rgbled(0xFF0000)  # Red

bt = Bluetooth() # create a Bluetooth object
#bth.start_scan(-1)  # start scanning with no timeout
bt.start_scan(15)    # starts scanning and stop after 10 s
while bt.isscanning():
   adv = bt.get_adv()
   if adv:
      adr_mac = ubinascii.hexlify(adv.mac) # convert hexadecimal to
                                                                     a
                                                                     s
                                                                     c
                                                                     i
                                                                     i
      print('Mac address of device: ',adr_mac)
   time.sleep(1)
pycom.rgbled(0x00FF00)  # Green
```

The method used above

`bluetooth.get_adv()`

returns a tuple with the following structure:

`(mac, addr_type, adv_type, rssi, data)`

`mac` - mac address of the device that sent the advertisement

`addr_type` - address type

`adv_type` - advertisement type received

`rssi` - signed integer with the signal strength of the advertisement

`data` - contains the complete 31 bytes of the advertisement message

2.2. Write a script for the task

Using the method of:

```
bluetooth.get_adv()
```

read the following parameters of the BLE device around and comment on the obtained results:

```
mac , addr_type, adv_type,
```

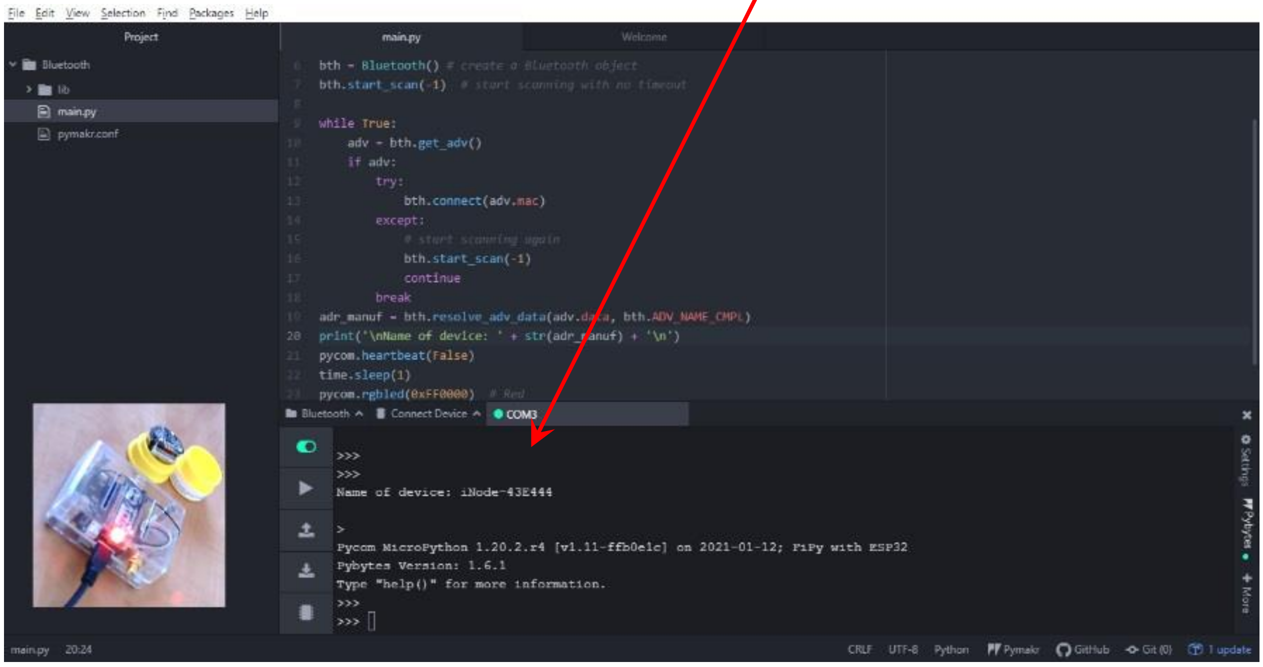
Use the constants to do task:

<u>Advertisement type:</u> Bluetooth.CONN_ADV, Bluetooth.CONN_DIR_ADV, Bluetooth.DISC_ADV, Bluetooth.NON_CONN_ADV, Bluetooth.SCAN_RSP

<u>Address type</u>: Bluetooth.PUBLIC_ADDR, Bluetooth.RANDOM_ADDR, Bluetooth.PUBLIC_RPA_ADDR, Bluetooth.RANDOM_RPA_ADDR

https://docs.pycom.io/firmwareapi/pycom/network/bluetooth

2.3. Scan the environment and get requested data type of BLE devices sending advertisements (name).



```
bth = Bluetooth() # create a Bluetooth object
bth.start_scan(-1)  # start scanning with no timeout

while True:
    adv = bth.get_adv()
    if adv:
        try:
            bth.connect(adv.mac)
        except:
            # start scanning again
            bth.start_scan(-1)
            continue
        break
adr_manuf = bth.resolve_adv_data(adv.data, bth.ADV_NAME_CMPL)
print('\nName of device: ' + str(adr_manuf) + '\n')
pycom.heartbeat(False)
time.sleep(1)
pycom.rgbled(0xFF0000)  # Red
```

```
>>>
>>>
Name of device: iNode-43E444

>
Pycom MicroPython 1.20.2.r4 [v1.11-ffb0e1c] on 2021-01-12; FiPy with ESP32
Pybytes Version: 1.6.1
Type "help()" for more information.
>>>
>>>
```

Use the following commands (file: Bluetooth2.txt):

```
from network import Bluetooth
import pycom
import ubinascii
import time

pycom.heartbeat(False)
pycom.rgbled(0xFF0000)   # Red

bt = Bluetooth() # create a Bluetooth object
#bth.start_scan(-1)   # start scanning with no timeout
bt.start_scan(10)     # starts scanning and stop after 10 seconds
while bt.isscanning():
    adv = bt.get_adv()
    if adv:
        adr_manuf = bt.resolve_adv_data(adv.data, bt.ADV_NAME_CMPL)
        print('\nName of device: ' , str(adr_manuf) , '\n')
    time.sleep(1)
pycom.rgbled(0x00FF00)  # Green
```

The method used above

```
        bluetooth.resolve_adv_data(data, data_type)
```

returns the requested `data_type` if present

`data` - bytes object with the complete advertisement data

`data_type` - data type to resolve from from the advertisement data.

2.4. Write next script for the task

Using the method of:

```
        bluetooth.resolve_adv_data(data, data_type)
```

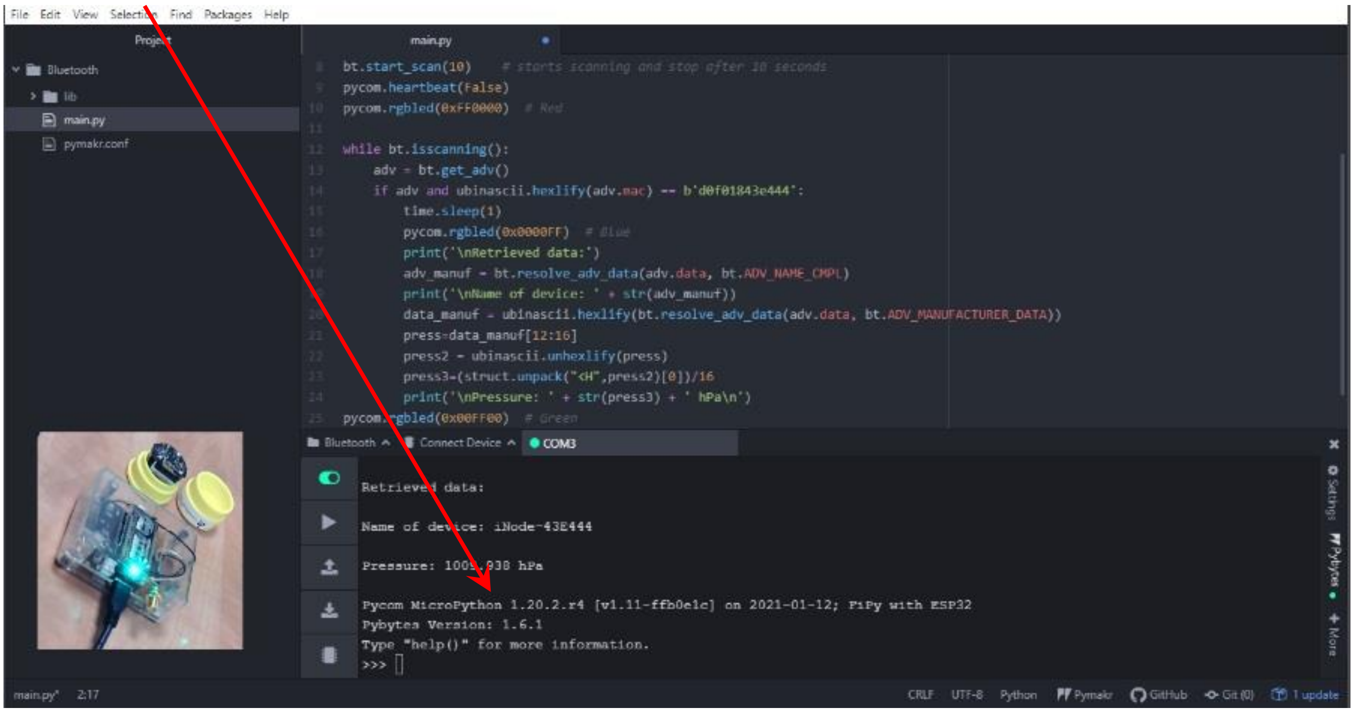
read the following parameters of the BLE device we connected to:

```
        flag, short name, manufacturer data
```

Use the constants to do task:

<u>Advertisement data type:</u> Bluetooth.ADV_FLAG, Bluetooth.ADV_16SRV_PART, Bluetooth.ADV_T16SRV_CMPL, Bluetooth.ADV_32SRV_PART, Bluetooth.ADV_32SRV_CMPL, Bluetooth.ADV_128SRV_PART, Bluetooth.ADV_128SRV_CMPL, Bluetooth.ADV_NAME_SHORT, Bluetooth.ADV_NAME_CMPL, Bluetooth.ADV_TX_PWR, Bluetooth.ADV_DEV_CLASS, Bluetooth.ADV_SERVICE_DATA, Bluetooth.ADV_APPEARANCE, Bluetooth.ADV_ADV_INT, Bluetooth.ADV_32SERVICE_DATA, Bluetooth.ADV_128SERVICE_DATA, Bluetooth.ADV_MANUFACTURER_DATA

2.5. Connect to BLE device with known MAC number and receive measurement data



```
bt.start_scan(10)    # starts scanning and stop after 10 seconds
pycom.heartbeat(False)
pycom.rgbled(0xFF0000)  # Red

while bt.isscanning():
    adv = bt.get_adv()
    if adv and ubinascii.hexlify(adv.mac) == b'd0f01843e444':
        time.sleep(1)
        pycom.rgbled(0x0000FF)  # Blue
        print('\nRetrieved data:')
        adv_manuf = bt.resolve_adv_data(adv.data, bt.ADV_NAME_CMPL)
        print('\nName of device: ' + str(adv_manuf))
        data_manuf = ubinascii.hexlify(bt.resolve_adv_data(adv.data, bt.ADV_MANUFACTURER_DATA))
        press=data_manuf[12:16]
        press2 = ubinascii.unhexlify(press)
        press3=(struct.unpack("<H",press2)[0])/16
        print('\nPressure: ' + str(press3) + ' hPa\n')
pycom.rgbled(0x00FF00)  # Green
```

```
Retrieved data:

Name of device: iNode-43E444

Pressure: 1009.938 hPa

Pycom MicroPython 1.20.2.r4 [v1.11-ffb0e1c] on 2021-01-12; FiPy with ESP32
Pybytes Version: 1.6.1
Type "help()" for more information.
>>>
```

Use the following commands (file Bluetooth3.txt):

```
from network import Bluetooth
import ubinascii
import struct
import pycom
import time

bt = Bluetooth()
bt.start_scan(20)    # starts scanning and stop after
10 s
pycom.heartbeat(False)
pycom.rgbled(0xFF0000)  # Red

while bt.isscanning():
   adv = bt.get_adv()
   if adv and ubinascii.hexlify(adv.mac) ==
b'd0f01843e444':

     pycom.rgbled(0x0000FF)  # Blue

     adv_manuf = bt.resolve_adv_data(adv.data,
                                     bt.ADV_NAME_CMPL
                                     )

     print('\nName of device: ' , str(adv_manuf))

     data_manuf =
                    ubinascii.hexlify(bt.resolve_adv
                    _data (adv.data,
                    bt.ADV_MANUFACTURER_DATA))

     press=data_manuf[12:16]

     press2 = ubinascii.unhexlify(press)

     press3=(struct.unpack("<H",press2)[0])/16

     print('\nPressure: ' + str(press3) + ' hPa\n')

pycom.rgbled(0x00FF00)  # Green
```

Above we used iNode Manufacturer Specific Data:

iNode Care Sensor PHT (0x9D)

```
12 9D 01 C0 00 00 4F 3E 3F 19 95 12 03 00 3C C0
91 99 BB A2 CC 23 AC 82
```

12	bit 2: `rtto` bit 3: `lowBattery`
9D	iNode Care Sensor PHT
01 c0	`groupsAndBattery` (uint16le);
00 00	Alarms (uint16le);
4f 3e	`rawPressure` (uint16le);
3f 19	`rawTemperature` (uint16le);
95 12	`rawHumidity` (uint16le);

`03 00`	`rawTime1` (uint16le);
`3c c0`	`rawTime2 (uint16le)`
`91 99 bb a2 cc 23 ac 82`	AES128 digital signature for the above data

https://inode.pl

2.6. Write next script for the task

Connect to BLE device with known name: iNode-43E444

and read the following parameters of the BLE device we connected to:

`Temperature, Humidity`

Using iNode Manufacturer Specific Data iNode Care Sensor PHT

converting hexadecimal to ascii by `ubinascii`

decoding Little endian by `struct`

Calculation of pressure (P [mbar]):

```
P = rawPressure / 16
```

Calculation of temperature (T [°C]):

```
T = (175.72 * rawTemperature * 4 / 65536) - 46.85

if T < -30

T = -30

if T > 70
```

```
T = 70
```

Calculation of humidity (H [%]):

```
H = (125 * rawHumidity * 4 / 65536) - 6
if H < 1
H = 1
if H > 100
H = 100
```

2.8. Modify FiPy application in order to add new variable sent to Ubidots cloud eg . Pressure, Temperature