Faculty of Electrical Engineering
Bialystok University of Technology

Erasmus+

Communication in IoT systems based on the Bluetooth Low Energy standard

Grażyna Gilewska, PhD

IoT - Internet of Things

system of electronic devices with sensors that can collect, process and exchange data through wired or wireless communication:

- smart home devices (washing machines, cleaning robots, fridges, televisions, air purifiers, thermostats, light bulbs),
- industry,
- energy sector,
- medical devices etc.

https://www.benchmark.pl

IoT - Internet of Things

The condition for the creation of IoT is the compatibility and capabilities of devices in terms of having appropriate sensors and modules responsible for mutual communication.

Due to multiplicity of devices, it is necessary to unambiguously identify a specific device:

- IP addresses
- MAC addresses
- product serial codes

IoT - Internet of Things

The main requirements of the IoT application to the networks used:

- low energy consumption (large number of sensors deployed in remote locations);
- low fees for using network;
- equipment used should be simple and cheap;
- long-distance transmission resistant to disturbances;
- large network capacity to support a large number of devices;
- adequate level of security of the communication network.

Bluetooth

Bluetooth is a standard for short-range wireless communication between various electronic devices such as:

- keyboard,
- computer, laptop,
- smartphone,
- wireless headphones
- and many others.

Bluetooth®

https://www.bluetooth.com

Bluetooth

- Open standard described in the IEEE 802.15.1 specification but no longer supported by the IEEE
- Currently managed by the Bluetooth Special Interest Group (SIG)
- SIG has more than 35,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics.

https://www.bluetooth.com

Bluetooth

- Bluetooth SIG oversees development of the specification, manages the qualification program, and protects the trademarks
- Each manufacturer must comply with the Bluetooth SIG standards in order to market their product as a Bluetooth device
- Technology applies to a network of patents that are licensed for each qualifying device (in 2021 the amount of Bluetooth chips produced was nearly 5 billion units – 5 000 000 000).

Bluetooth®

https://www.bluetooth.com

Why the name Bluetooth

- The name "Bluetooth" was proposed in 1997 by Jim Kardach of Intel, one of the founders of the SIG
- Kardach was fascinated by the first Danish king Harald, who contributed to the unification of rival clans (just like the technology under development)
- King Harald was called Bluetooth
- Kardach proposed Bluetooth as the codename for the short-range wireless technology
- He chose the first character from the runic notation of King Harald's nickname (ᛒ) as the logo of the standard.

https://www.bluetooth.com

Bluetooth - development

In 1999, the Bluetooth SIG published a 1500-page specification for the first version of Bluetooth technology and is constantly working to improve it.

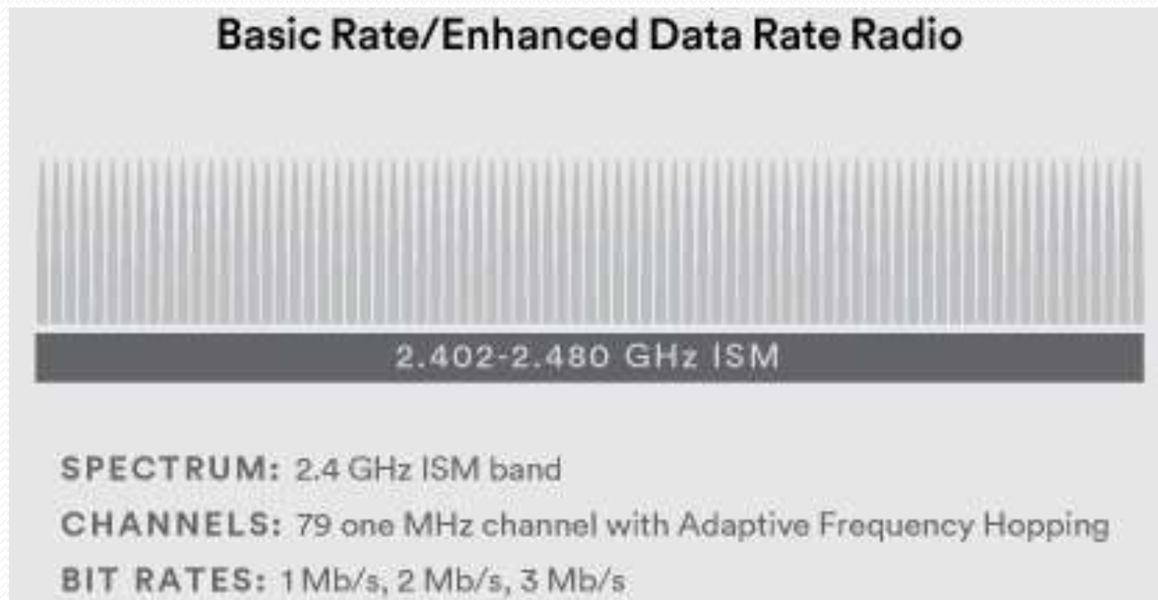
The current version of the specification has more than 3,000 pages and covers the complete system, from the physical layer to the application layer.

https://www.bluetooth.com

Bluetooth - Classic

The Bluetooth Classic is a low power radio that streams data in the 2.4GHz (2.402 - 2.480 GHz) unlicensed frequency band supporting point-to-point device communication.

The spectrum of this band is divided into 79 channels, each of them has a 1MHz band.



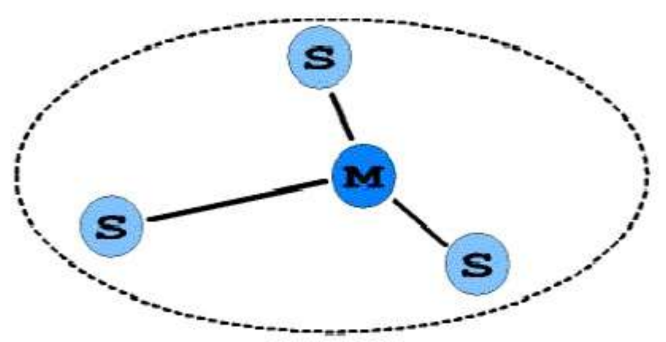
https://www.bluetooth.com

Bluetooth - Classic

- Bluetooth uses a radio technology called frequency-hopping spread spectrum. Bluetooth divides transmitted data into packets (parts), and transmits each packet on one of 79 Bluetooth channels. It usually performs 1600 hops per second, with adaptive frequency-hopping enabled.
- Each packet is sent on a specific channel, after which the air interface selects a new channel on which the next packet will be sent. Thanks to this process, the message is transmitted over the entire available frequency spectrum.
- For this reason, it is required that the transmitter and receiver are properly tuned so that the receiver knows the hop pattern and can receive the packets and then assemble them into a complete message.

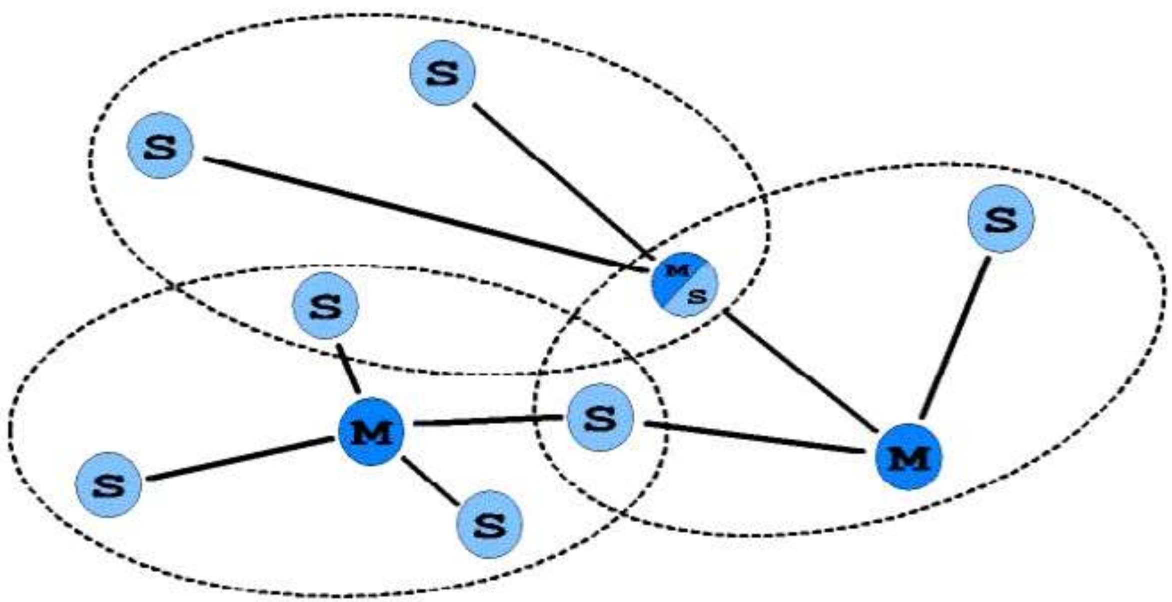
Bluetooth - Classic

- The Bluetooth network model is a proximity network-based communication model. Which means devices can automatically, spontaneously connect whenever they are within range.
- They are based on the master-slave model, when devices are establishing a connection with each other, one is the master and the other is the slave.
- All slave devices that communicate with the master device create a piconet, where the number of active slaves cannot exceed seven.



Bluetooth - Classic

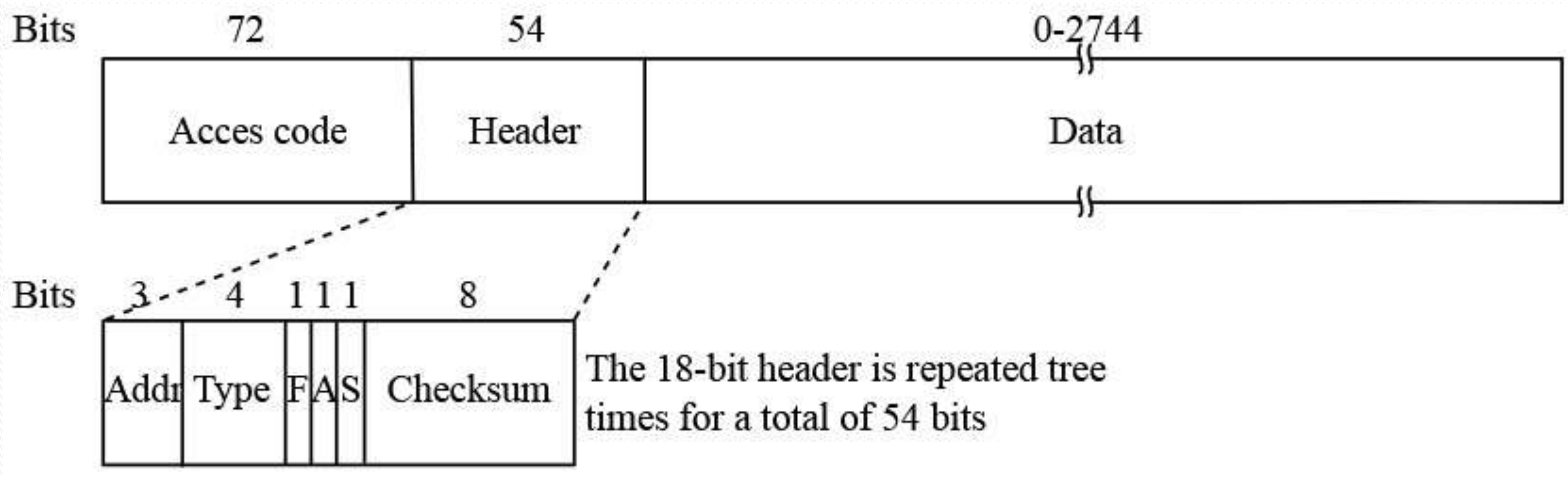
- The Bluetooth Core Specification provides for the connection of two or more piconets to form a scatternet, in which certain devices simultaneously play the master /slave role in one piconet and the slave role in another.
- All devices within a given piconet use the clock provided by the master as the base for packet exchange.



Bluetooth - Classic

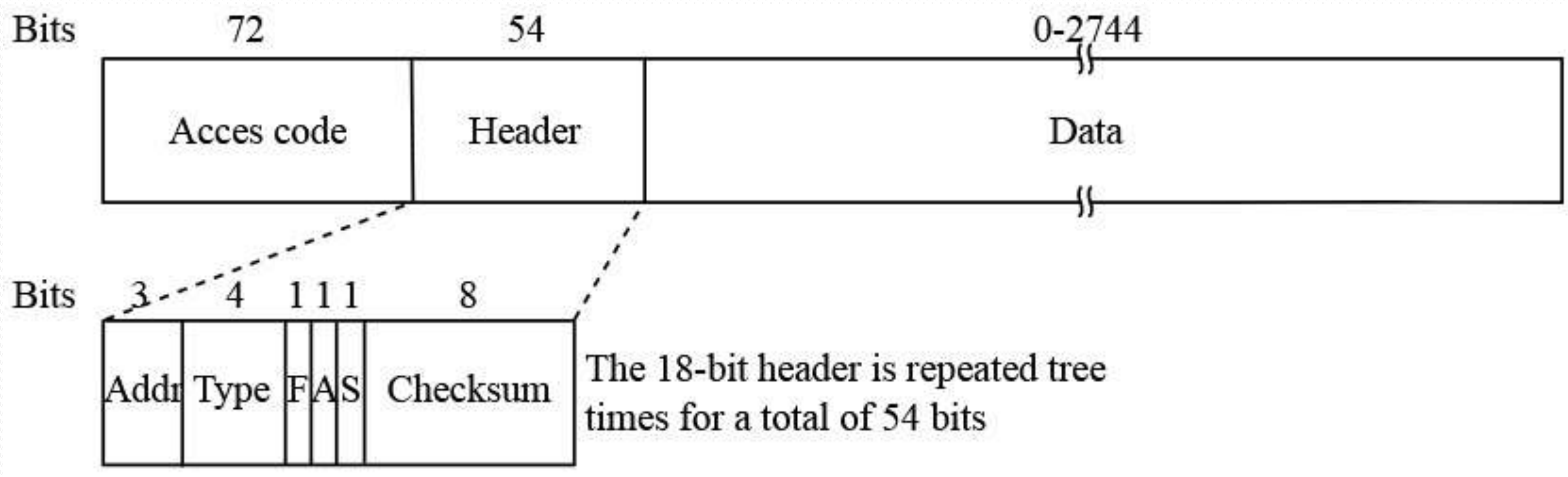
- Each Bluetooth device has a unique address provided by the manufacturer. Which excludes the situation that two different Bluetooth devices, e.g. a mouse and a keyboard, will stop connecting to the computer and start with each other.
- Receiver sensitivity is the measure of the minimum signal strength a receiver can interpret.
- Bluetooth technology specifies that devices must be able to achieve a minimum receiver sensitivity of -70 dBm to -82 dBm, depending on the physical layer used.

Bluetooth - frame structure



- Access code - identifies the master for the slave device within the range of two master devices to which the transmission is to take place.
- Header - 54 bits - 18 bits of header repeated three times, on the receiving side, all three copies of each bit are checked.

Bluetooth - frame structure



- Data - up to 2744 bits containing data (for a 5-slot frame). For a one-shot transmission, the frame contains 240 bits of the data field..

Bluetooth

Ranges of Bluetooth devices by power-class:

Power class	Maximum output power	Typical range
1	100 mW	100 m
1.5	10 mW	20 m
2	2.5 mW	10 m
3	1 mW	1 m

The effective range varies depending on propagation conditions, antenna configurations and battery conditions.

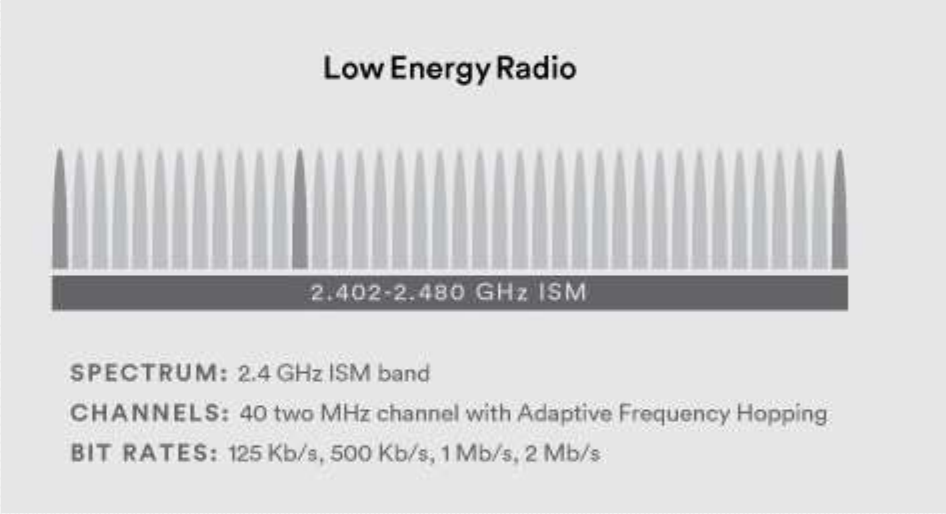
Comparison of the basic parameters of the Bluetooth version:

Version	Publication date	Maximum bandwidth
Bluetooth 1.0a, 1.0B	1999	21 kb/s
Bluetooth 1.1	2001	124 kb/s
Bluetooth 1.2	2003	721 kb/s
Bluetooth 2.0	2005	2,1 Mb/s
Bluetooth 2.1	2007	2,1 Mb/s
Bluetooth 3.0	2009	24 Mb/s
Bluetooth 4.0 + LE	2009	24 Mb/s lub 1 Mb/s *
Bluetooth 4.1	2013	25 Mb/s lub 1 Mb/s *
Bluetooth 4.2	2014	25 Mb/s lub 1 Mb/s *
Bluetooth 5.0	2016	50 Mb/s lub 2 Mb/s *
Bluetooth 5.1	2019	50 Mb/s lub 2 Mb/s *

* for wearables devices.

BLE - Bluetooth Low Energy (Bluetooth LE)

The Bluetooth Low Energy (BLE) radio is designed for very low power operation, uses the same 2.4GHz frequency band, but divides it into 40 channels with a width of 2MHz each.



https://www.bluetooth.com

BLE - Bluetooth Low Energy (Bluetooth LE)

The Bluetooth Low Energy (BLE) is designed to operate with very low energy consumption.

BLE supports multiple communication topologies:

- point-to-point
- broadcast
- mesh

BLE now includes features that enable one device to determine the presence, distance, and direction of another device.

https://www.bluetooth.com

BLE - Bluetooth Low Energy (Bluetooth LE)

BLE greatly reduces power consumption by turning off the data transmission module when nothing is transmitted.

Contrary to the classic bluetooth module, the transmission is carried out on the basis of intervals and not a permanent connection.

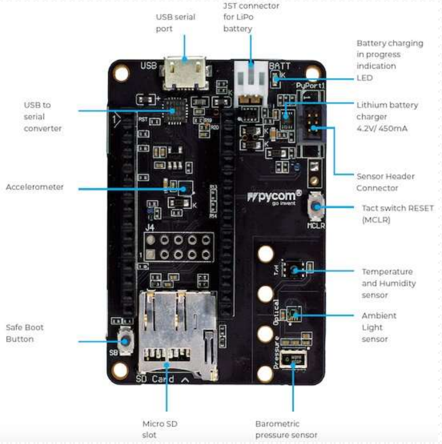
https://www.bluetooth.com

Applied systems and devices:

IoT systems will be used, consisting of FiPy ESP32 Pycom modules that allow for communication with the Bluetooth network, and a Pysense expansion board with additional sensors of environmental parameters.

- FiPy with ESP32 Pycom
- Pysense expansion board
- iNode Care Sensor PHT

https://pycom.io

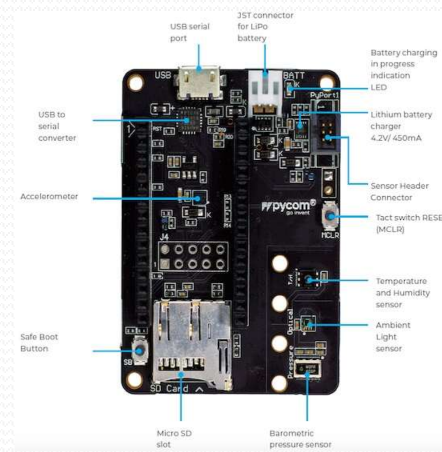


https://inode.pl/

FiPy with ESP32 Pycom:

enables communication using Bluetooth LE network and Pysense expansion board with additional sensors of environmental parameters.

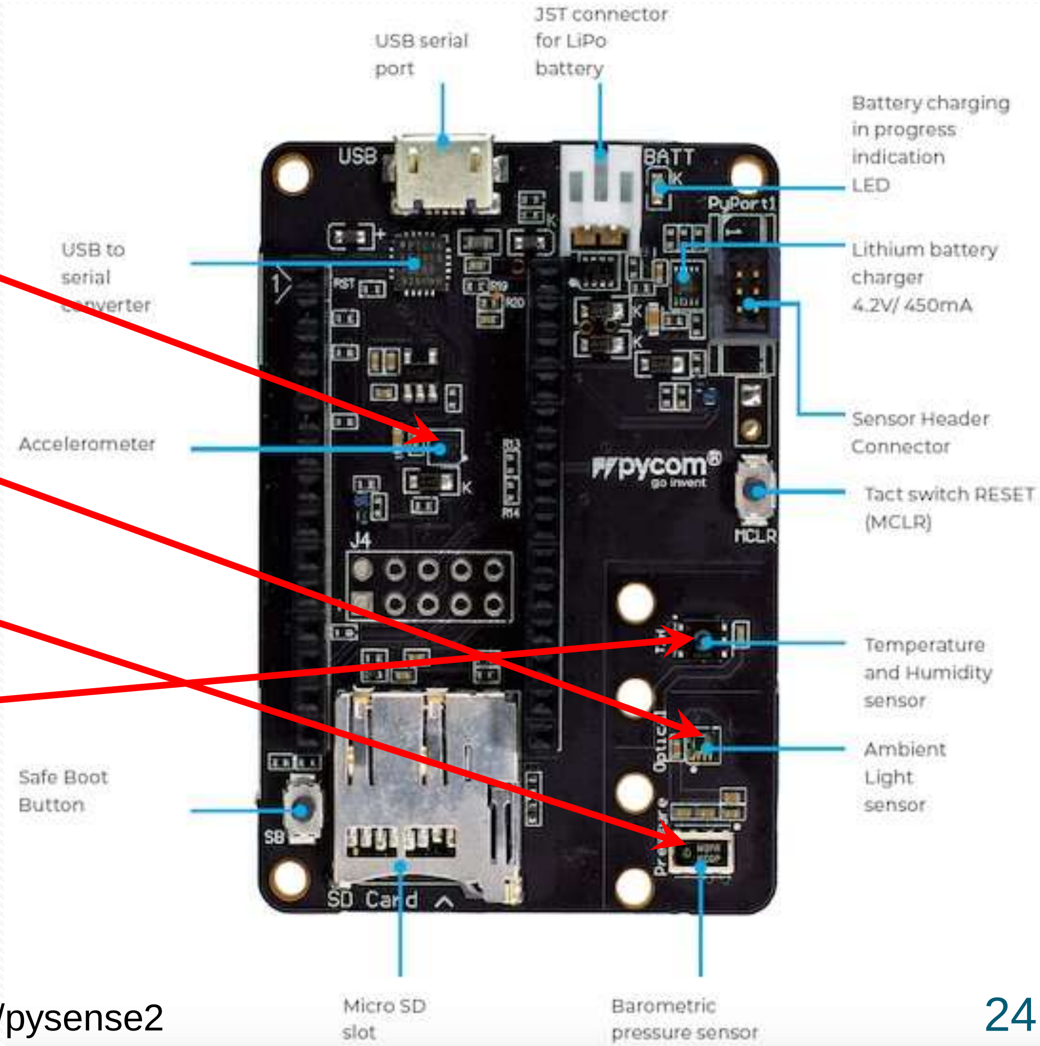
https://pycom.io



Pysense shield:

enables the measurement of environmental parameters using 5 sensors:

- Accelerometer (LIS2HH12)
- Light sensor (LTR329ALS01)
- Pressure sensor (MPL3115A2)
- Temperature / Humidity sensor (SI7006A20)

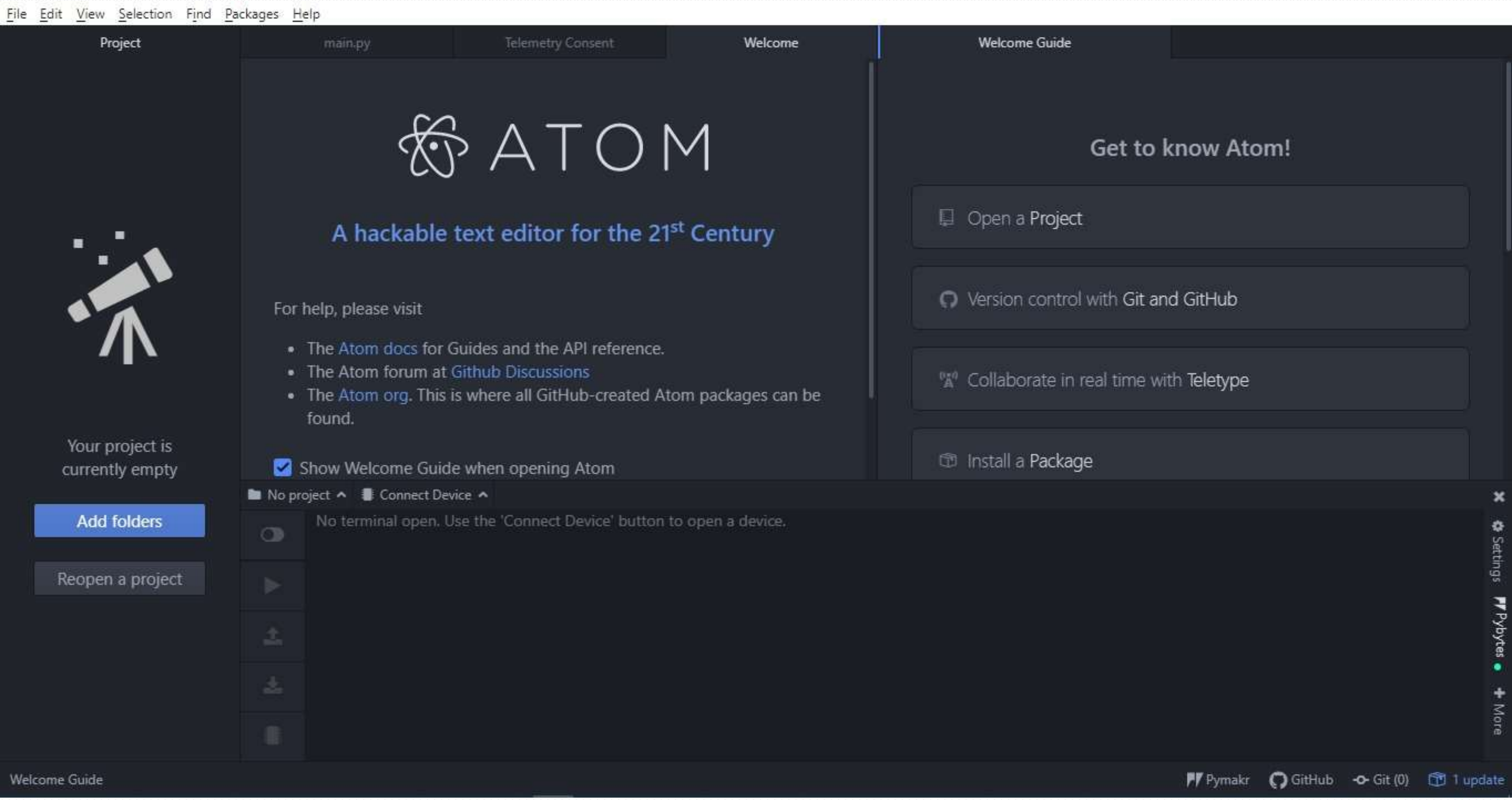


Preparation of environment for device programming

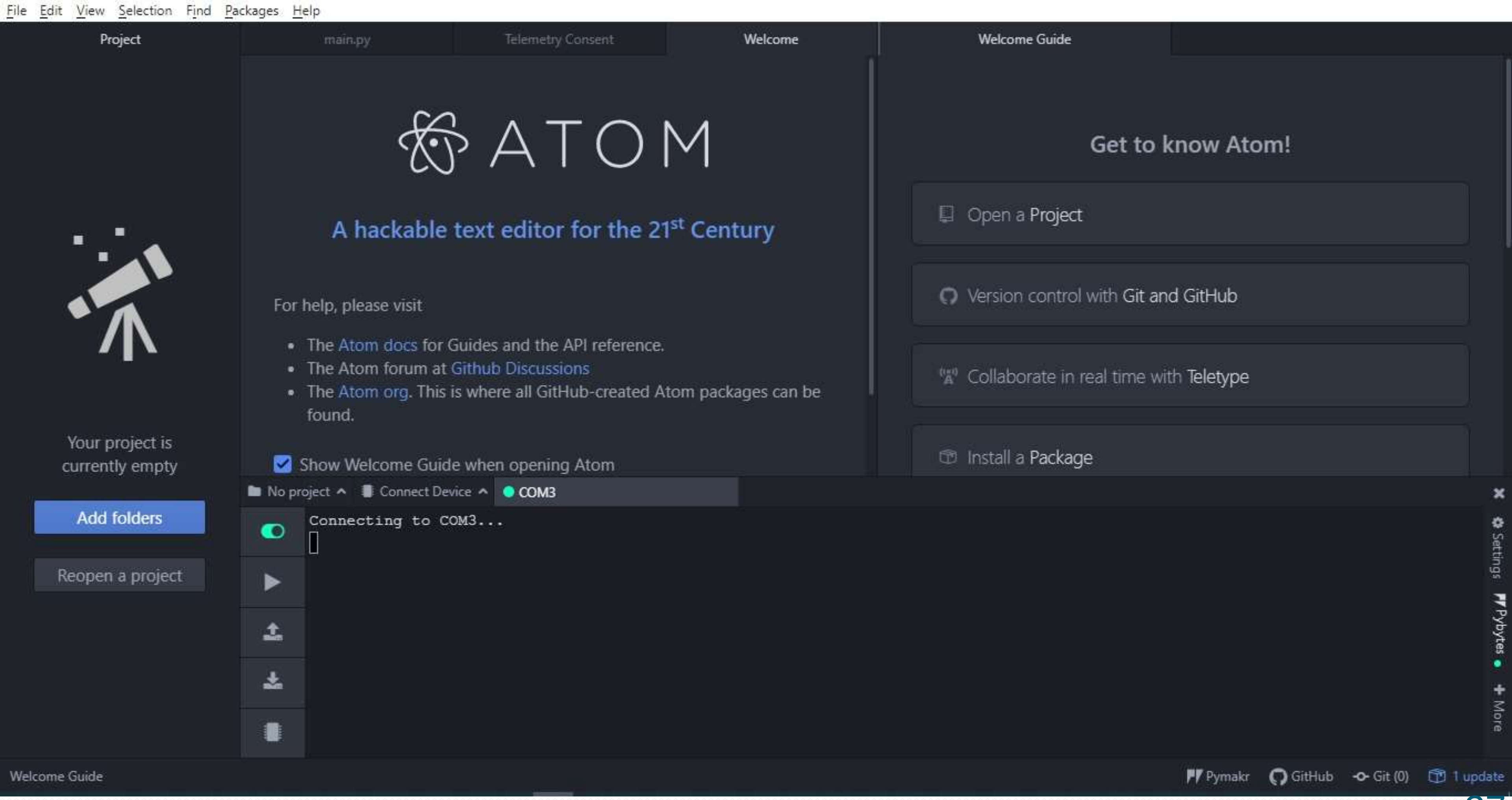
Use Atom Text Editor & Pymakr Plugin environment:

- download and install Atom (https://atom.io)
- install the official Pycom Pymakr Plugin via Atom
- connect FiPy module to computer via USB
- test some basic MicroPython commands

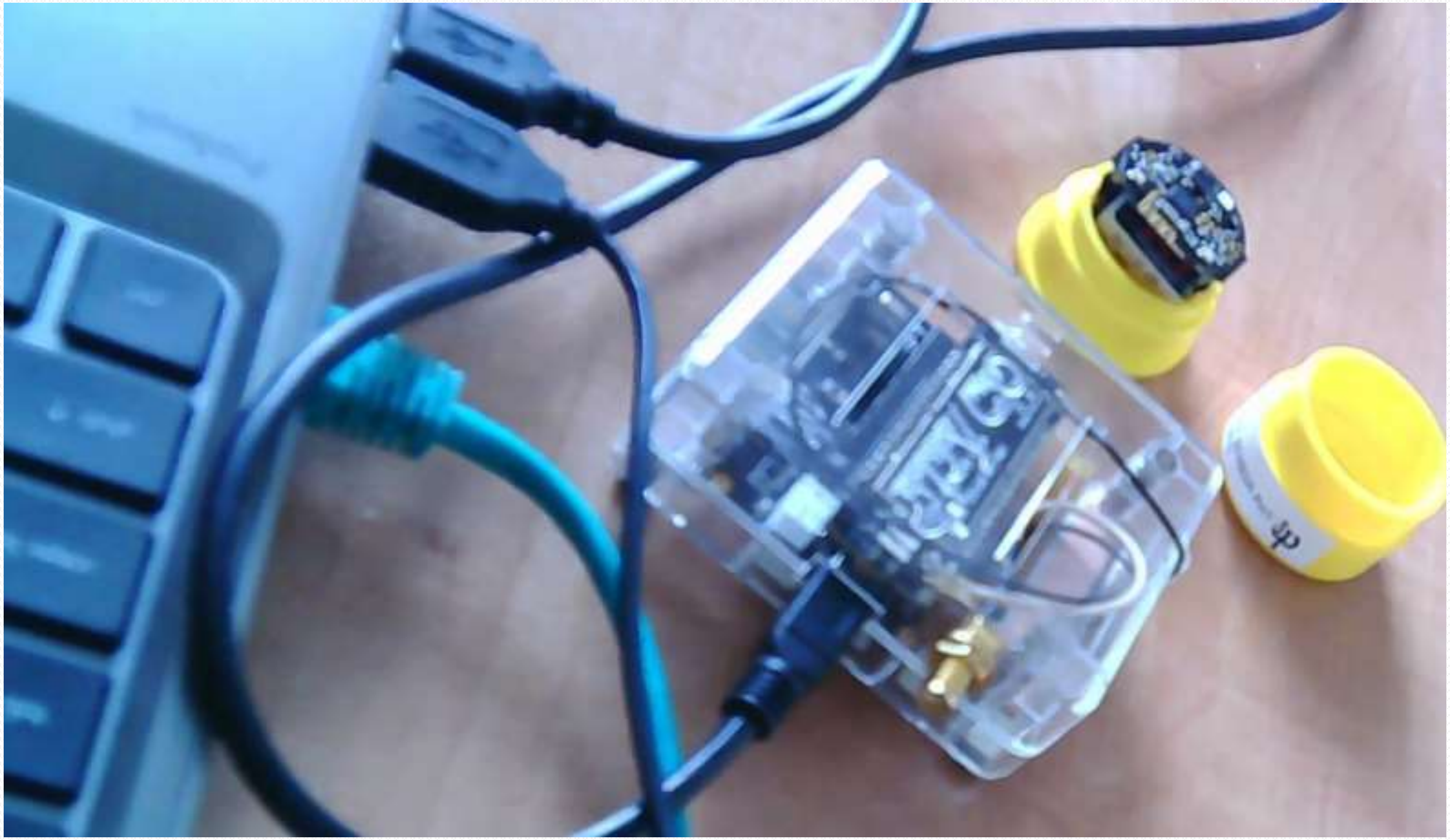
Atom and Pycom Pymakr Plugin:



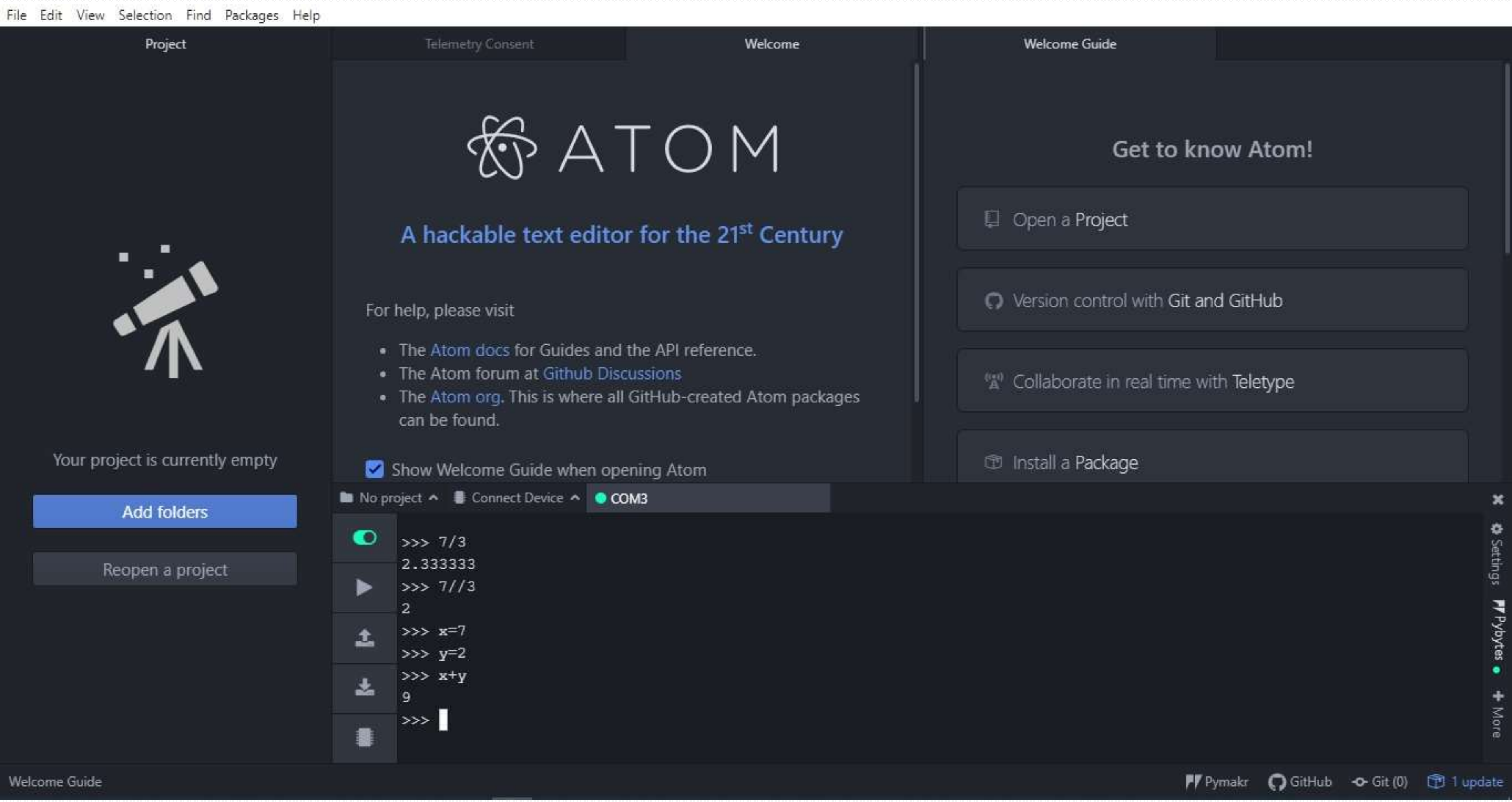
Connected FiPy module to computer via USB (COM3)



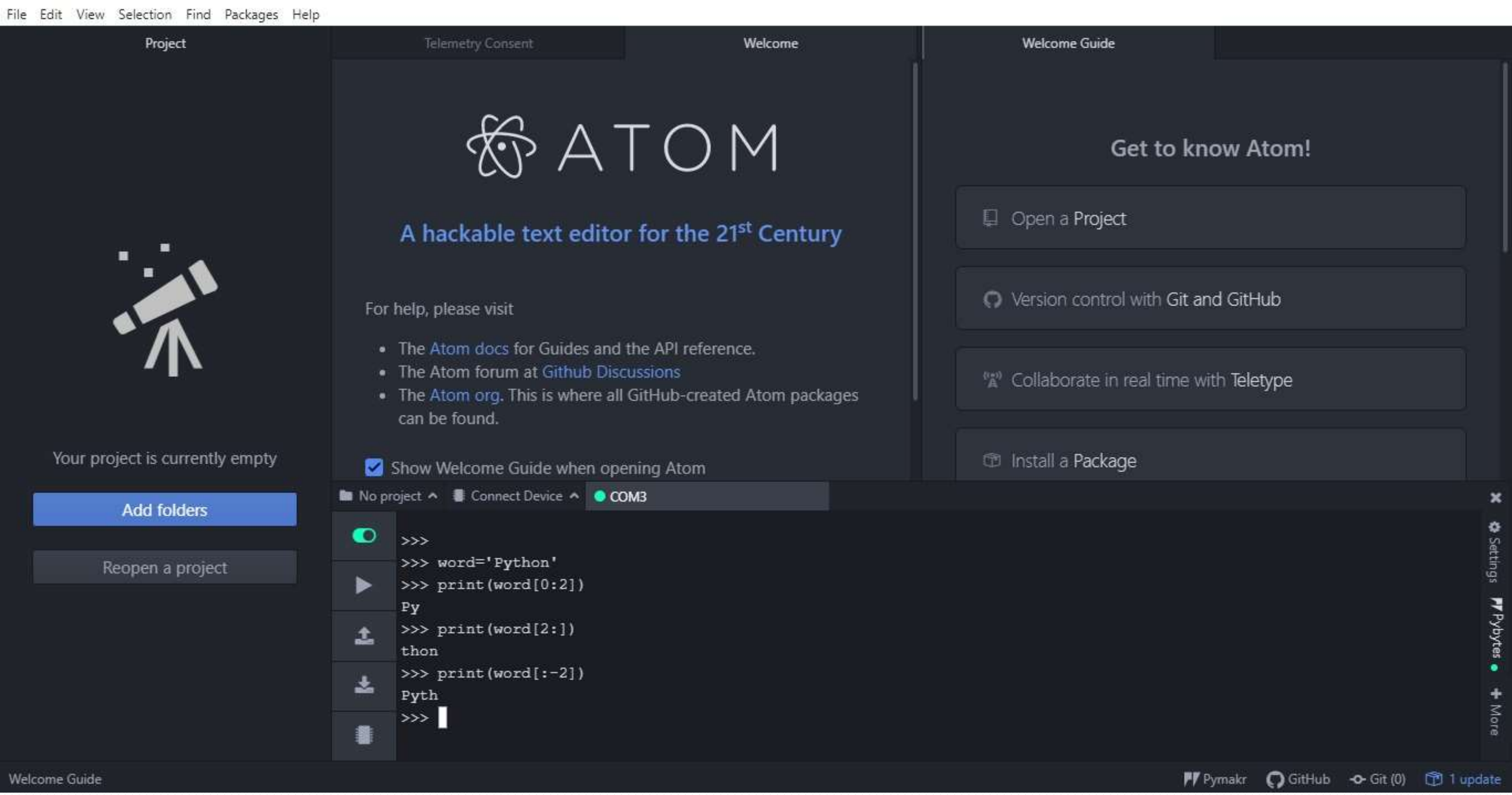
Devices used for tasks:



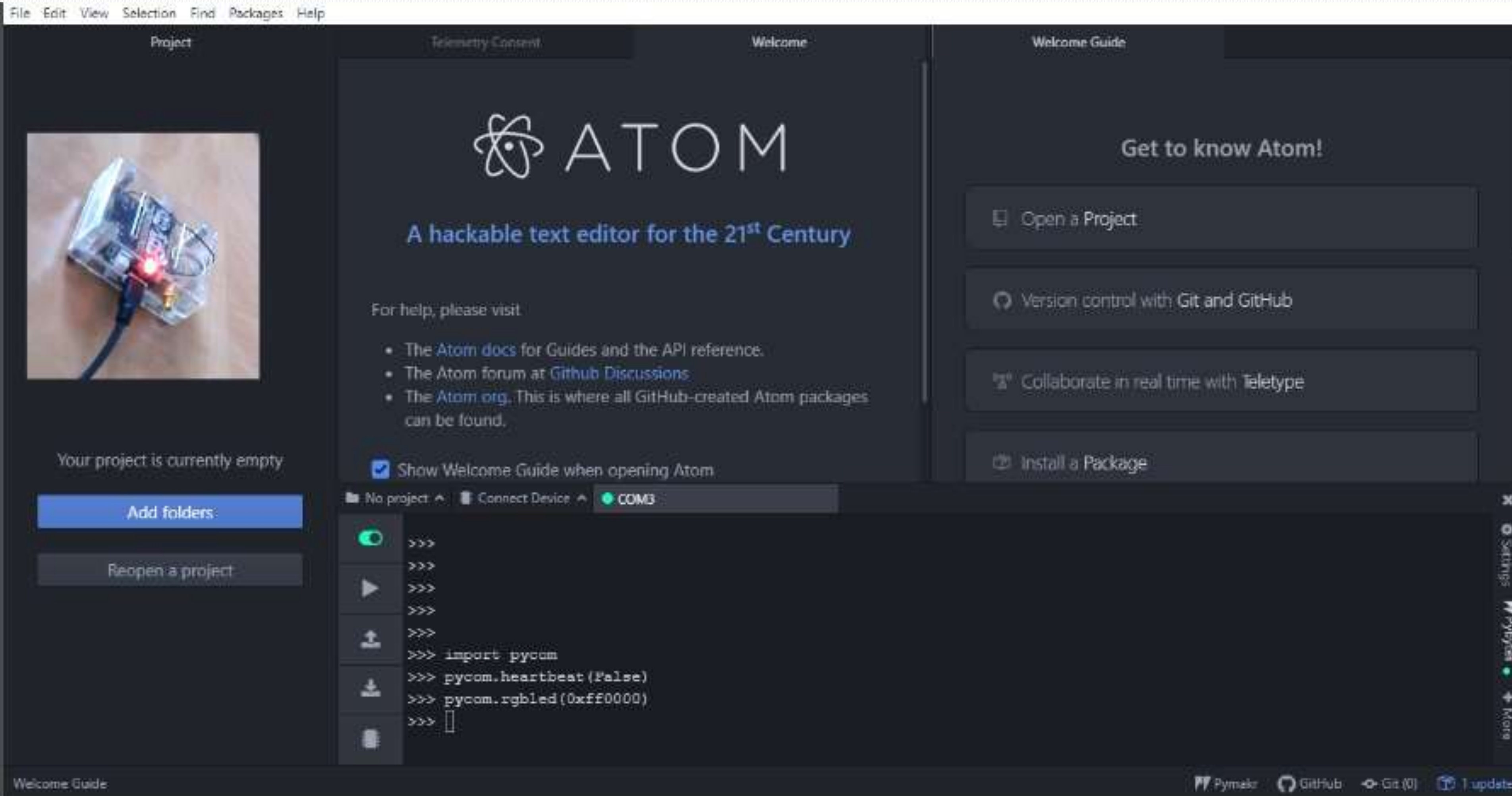
Test some basic MicroPython commands:



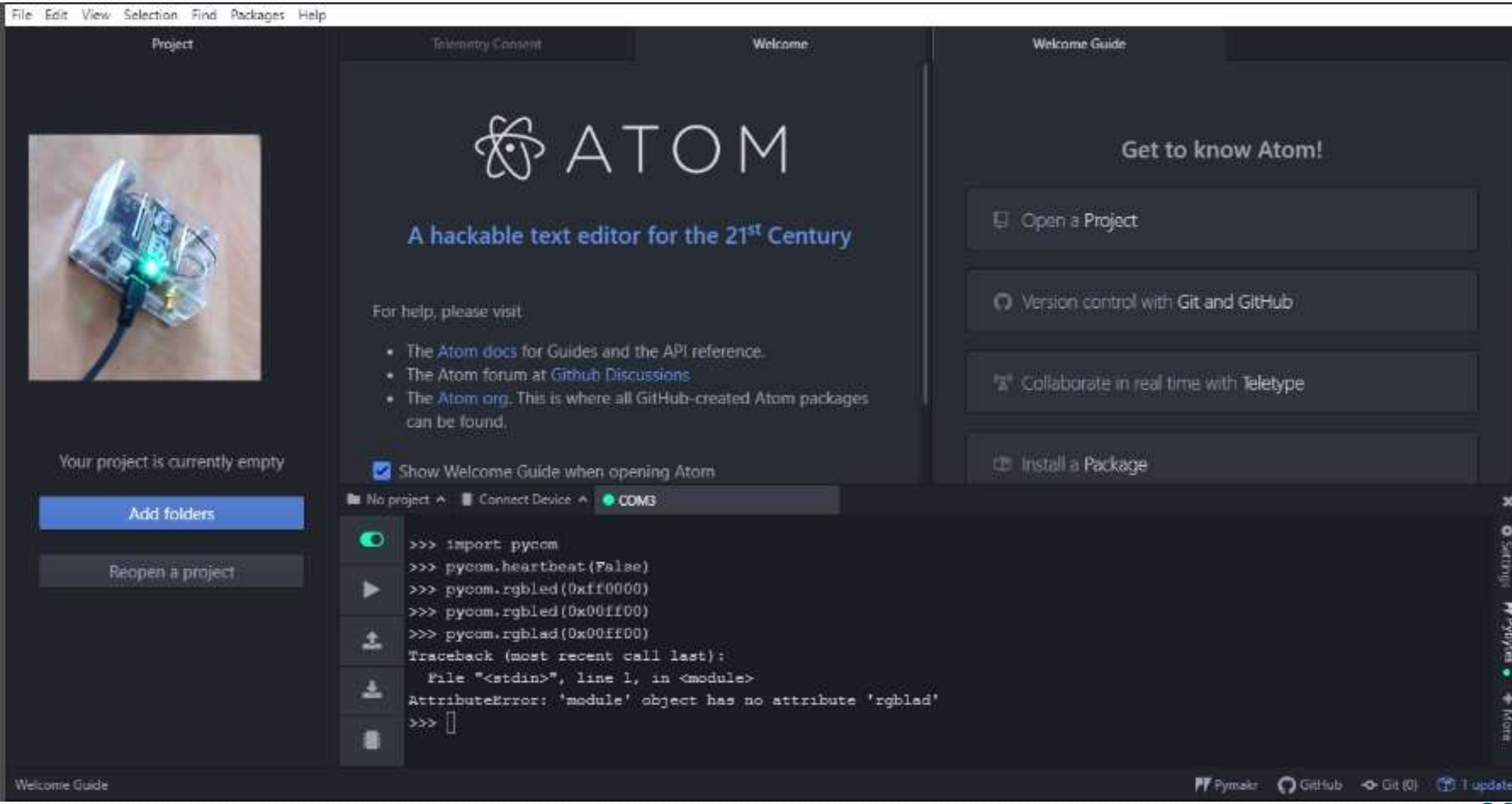
Test some basic MicroPython commands:



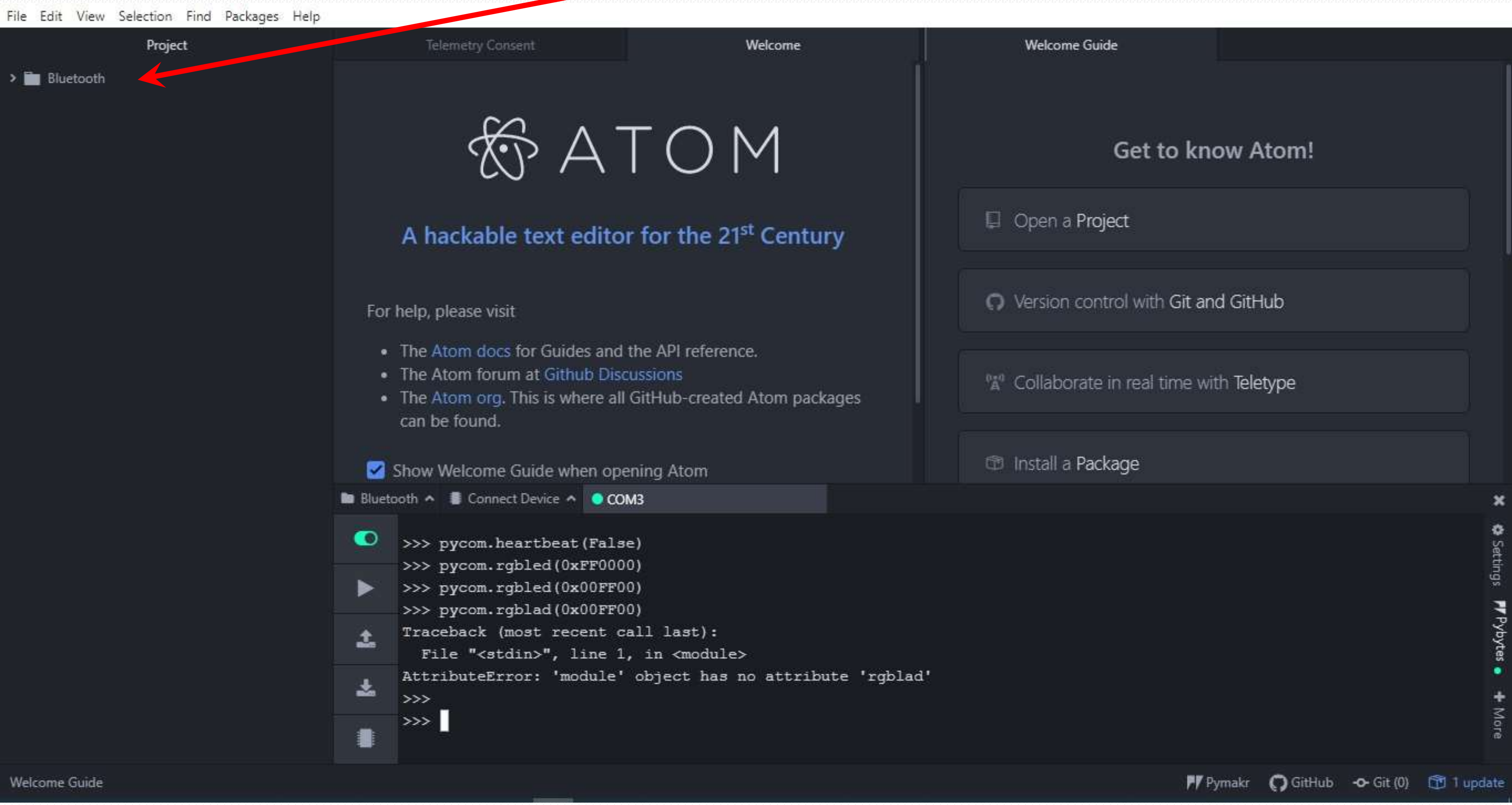
Programming of the FiPy module
(set red color fo LED)



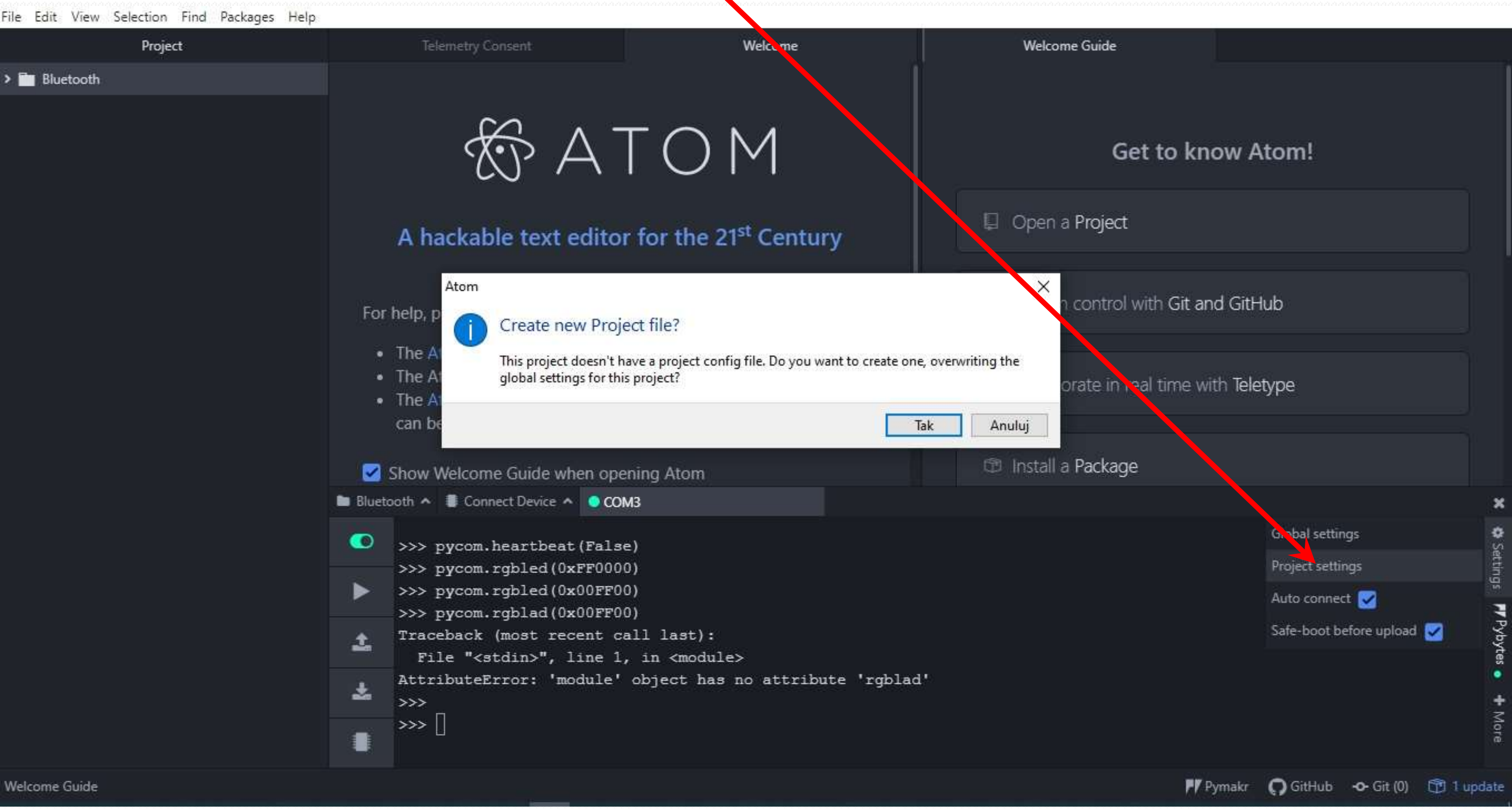
Programming of the FiPy module
(set green color fo LED and response to an incorrect command)



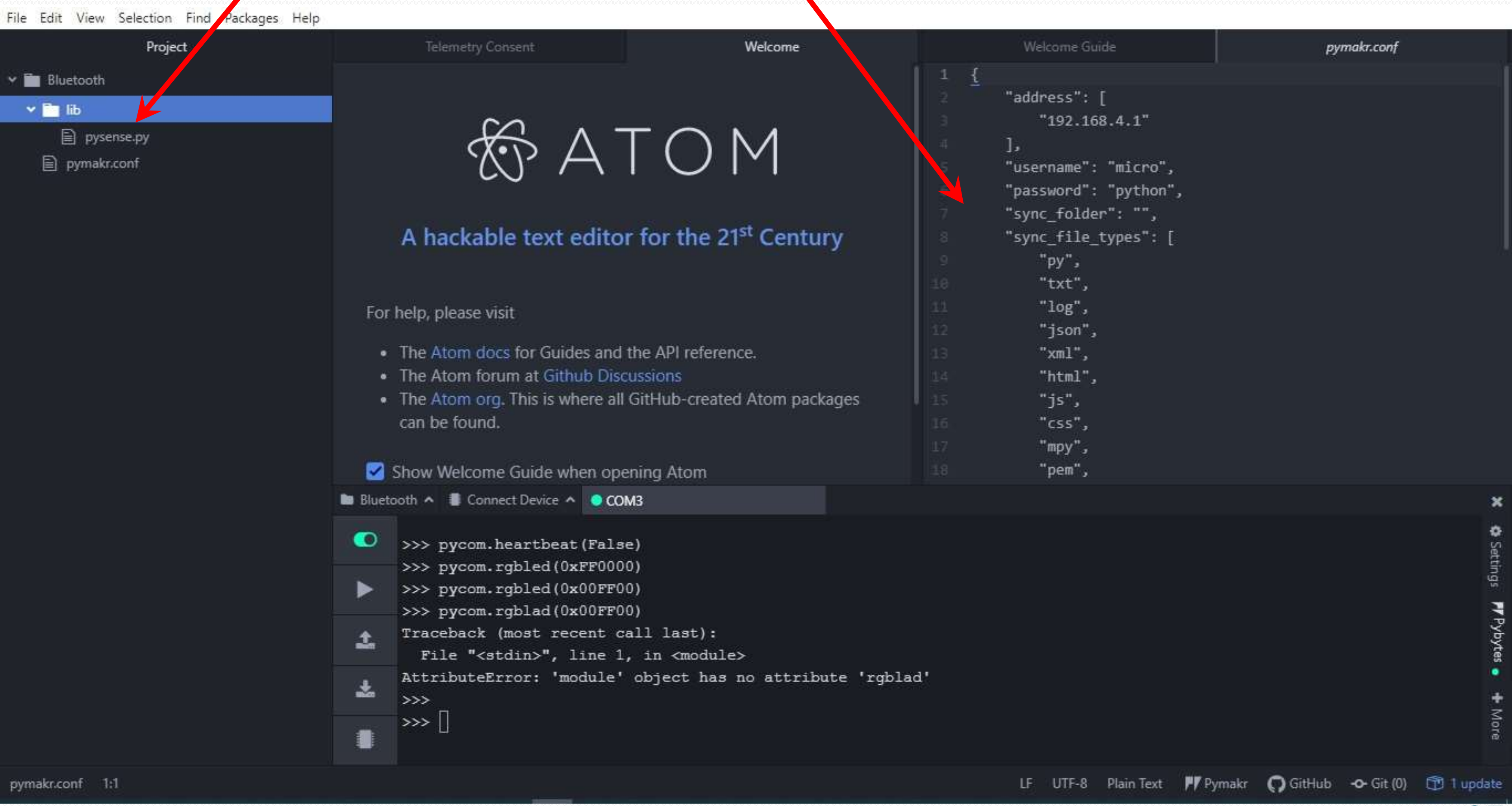
Create project directory:



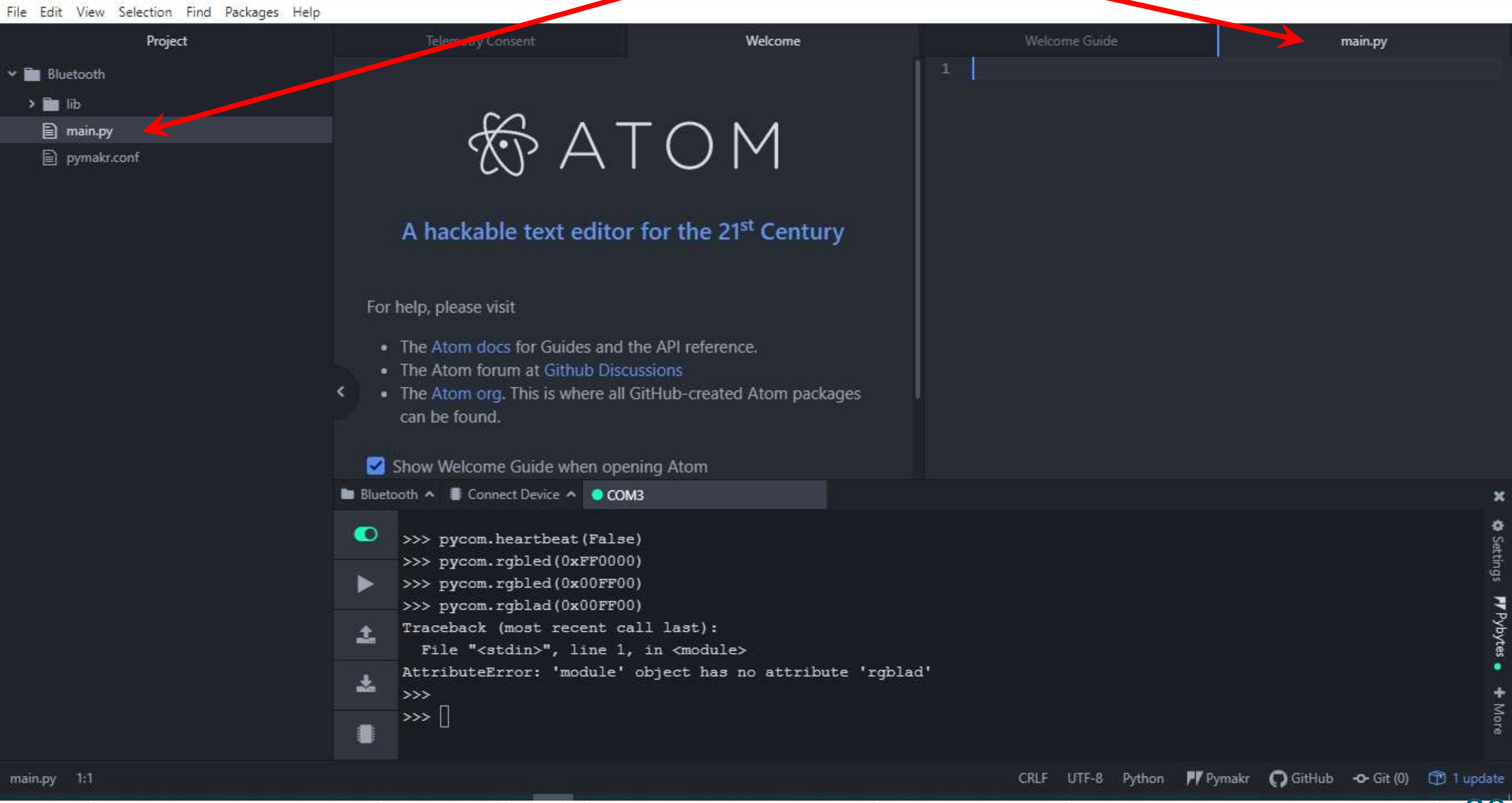
Configure project settings



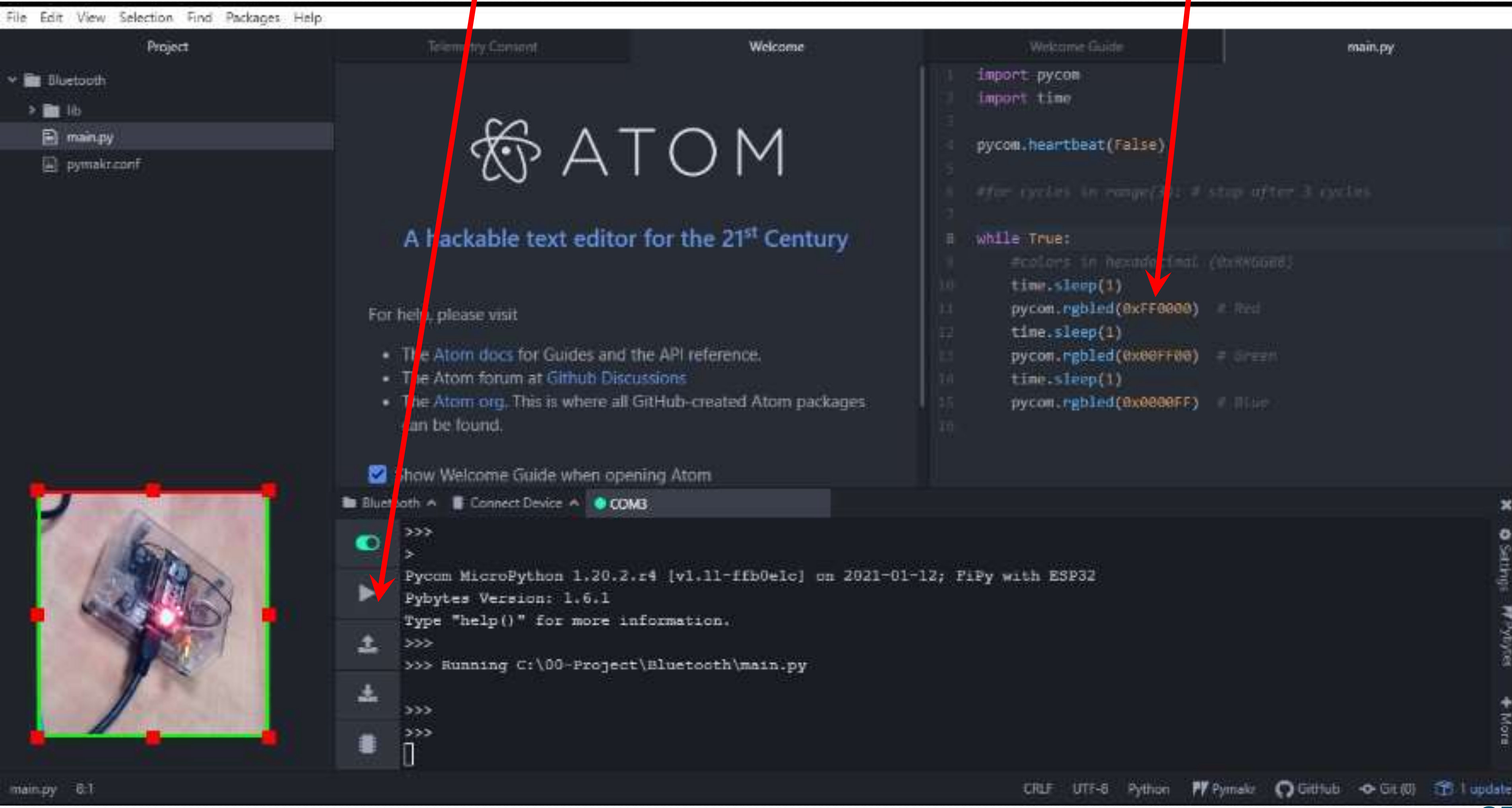
The contents of the config file and additional library



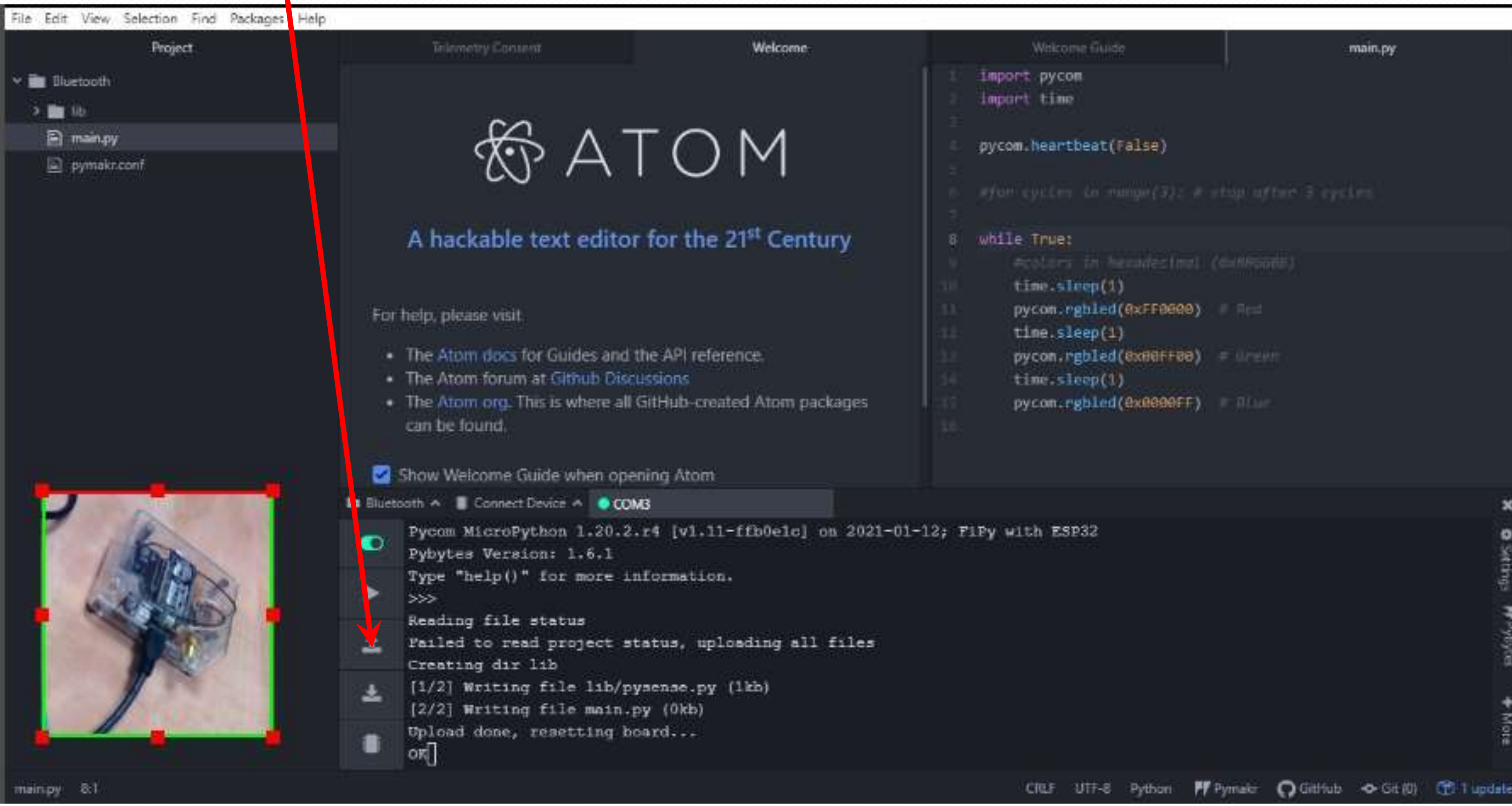
Create the main project file



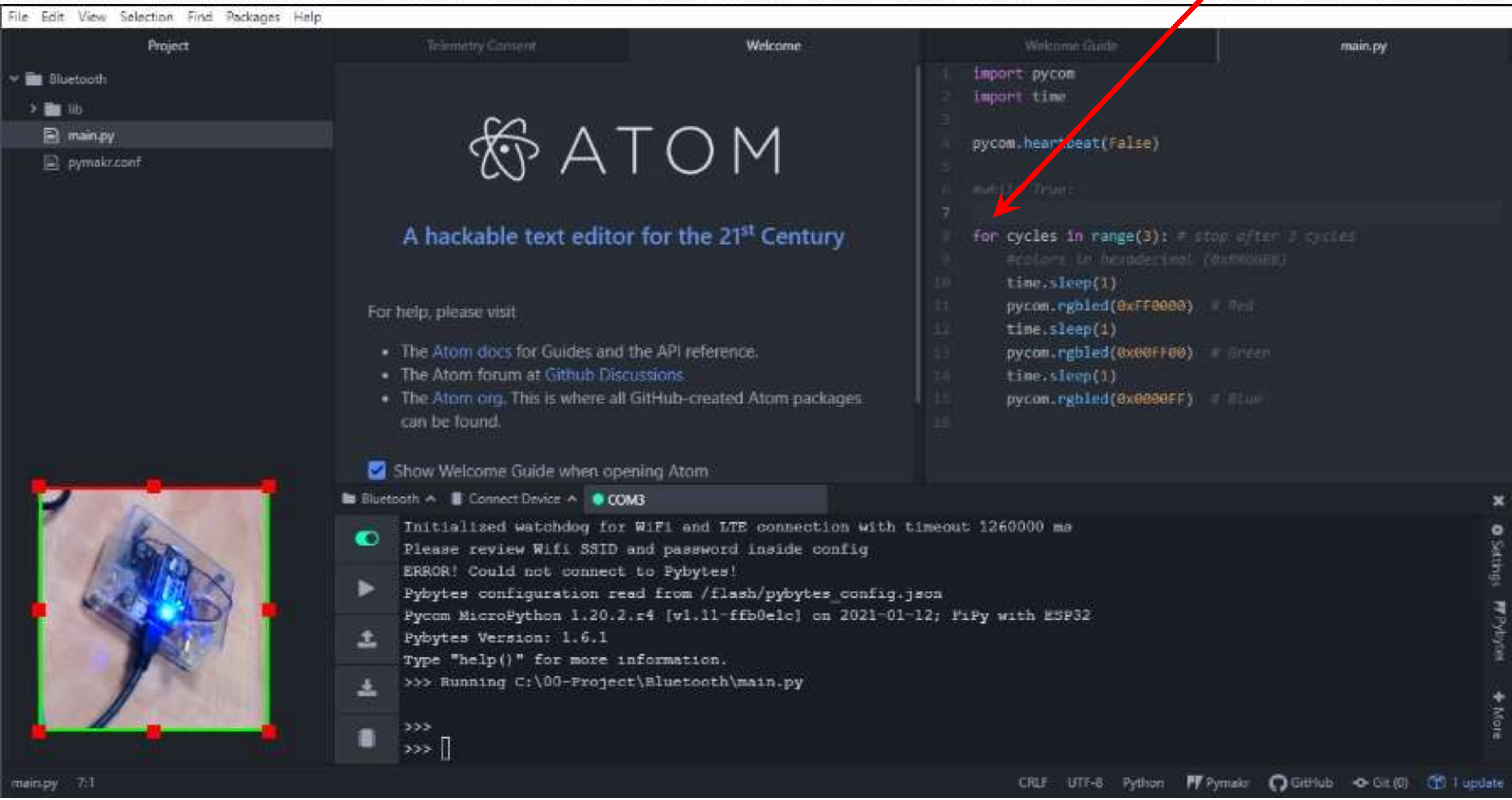
Create and run project that changes LED color every 1 second



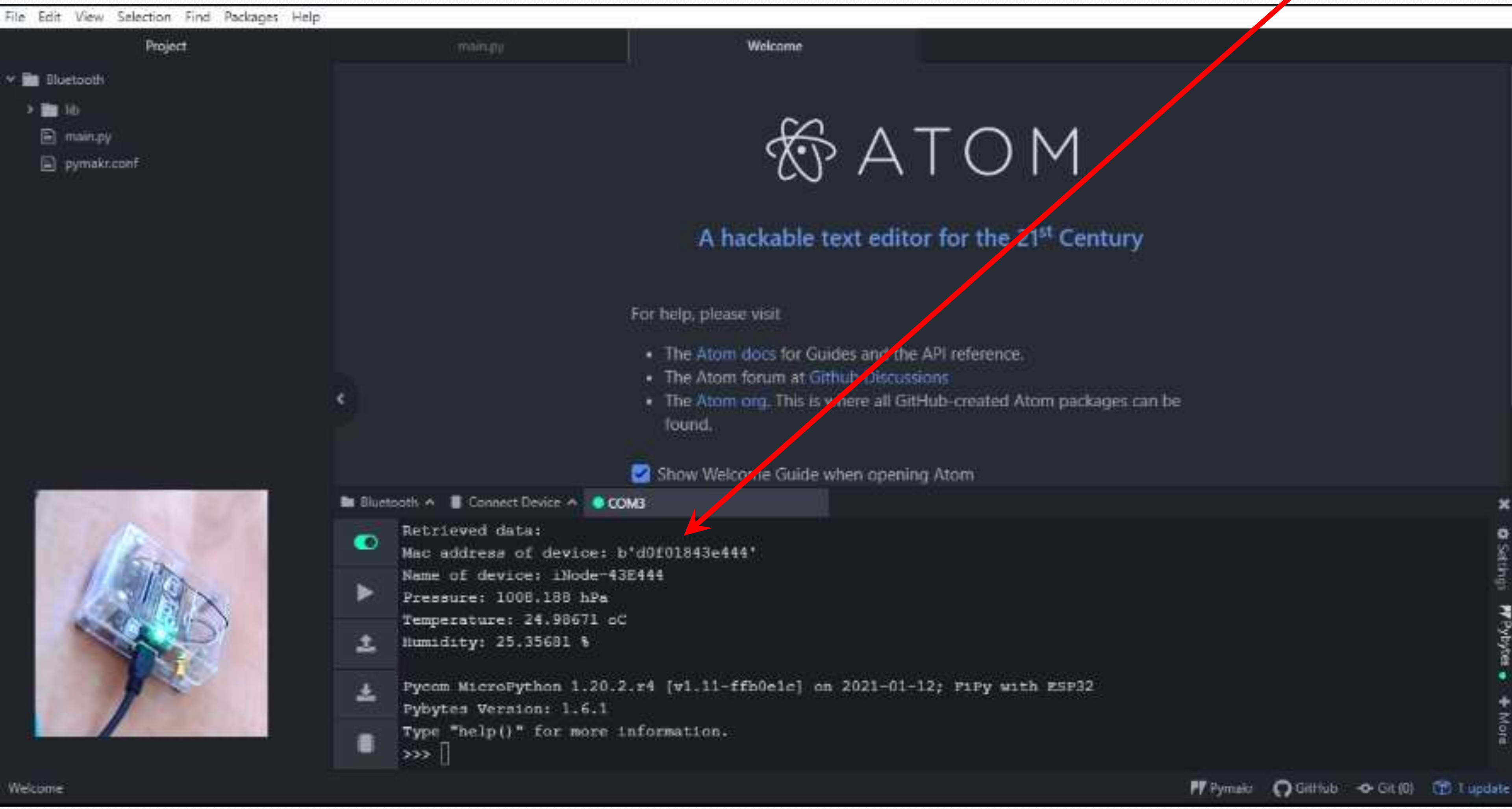
Upload project to device



Changed project: infinite While-ioop to For-loop

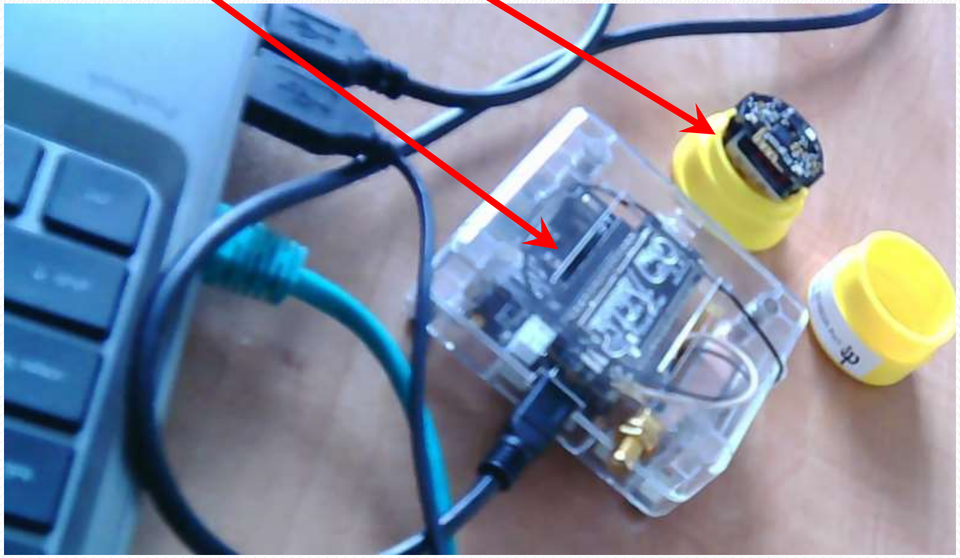


Task topic: Connect to a BLE Device and read data



Devices used for task:

- FiPy with ESP32 Pycom and Pysense expansion board
- iNode Care Sensor PHT



Use property constructors:

- `class network.Bluetooth`

methods:

- `bluetooth.start_scan`
- `bluetooth.get_adv()`
- `bluetooth.resolve_adv_data`

and commands:

- `ubinascii.hexlify`

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

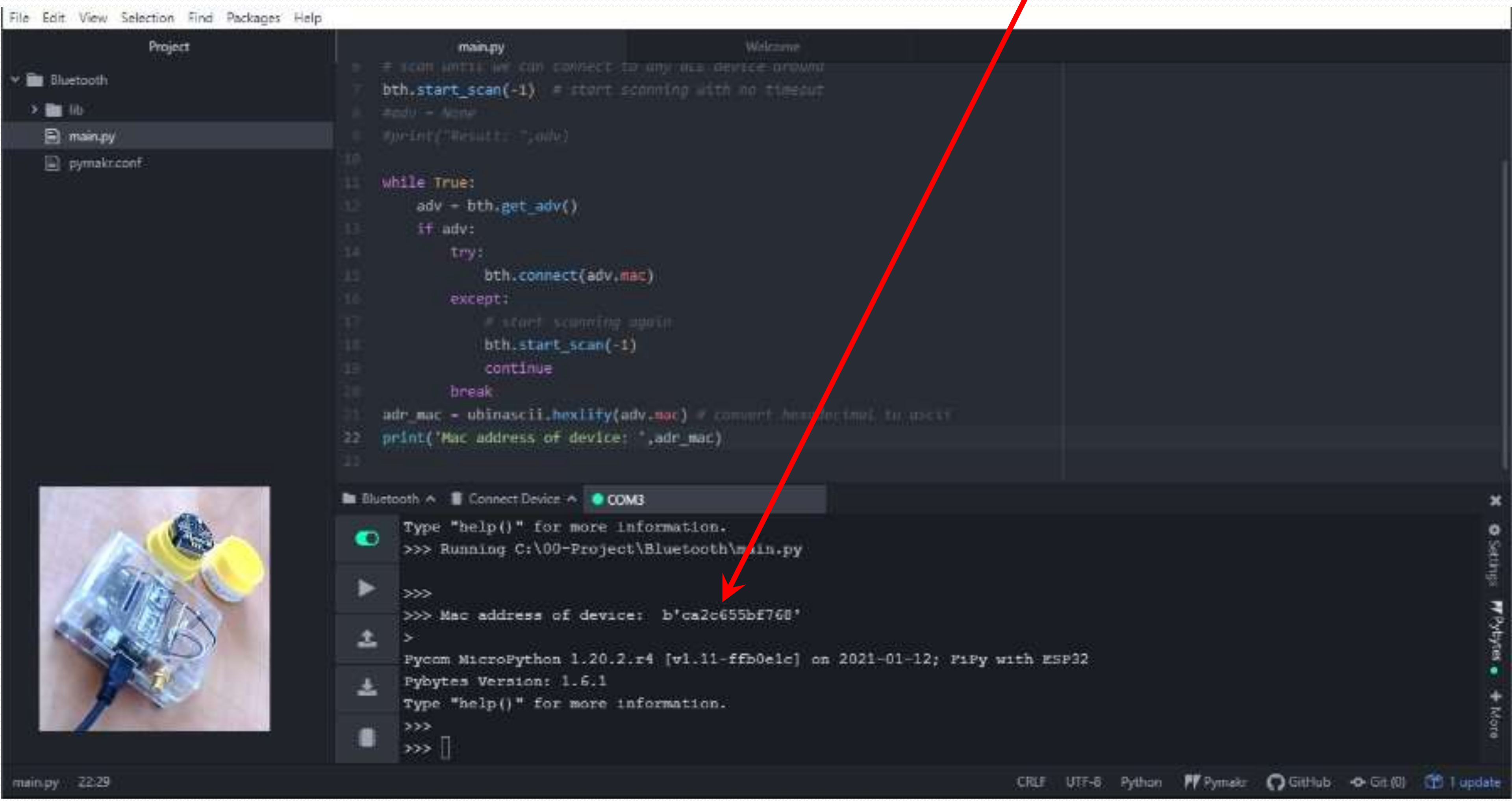
Realise the task in 3 parts:

- Connecting to a device that is sending advertisements and receiving advertisement data
- Connecting to the device sending advertisements and receiving device data
- Connecting to the device sending advertisements and receiving measurement data

Part 1 Scan until we can connect to any BLE device around:

```
from network import Bluetooth
import ubinascii
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_mac = ubinascii.hexlify(adv.mac) # convert hexadecimal to ascii
print('Mac address of device: ',adr_mac)
```

Part 1 Scan until we can connect to any BLE device around:



Part 1 Scan until we can connect to any BLE device around:

We used the method:

```
bluetooth.get_adv()
```

gets tuple which has 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

Part 1 Write a script for the task:

Using the method of:

```
bluetooth.get_adv()
```

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

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

Constants:

Advertisement type: `Bluetooth.CONN_ADV, Bluetooth.CONN_DIR_ADV, Bluetooth.DISC_ADV, Bluetooth.NON_CONN_ADV, Bluetooth.SCAN_RSP`

Address type: `Bluetooth.PUBLIC_ADDR, Bluetooth.RANDOM_ADDR, Bluetooth.PUBLIC_RPA_ADDR, Bluetooth.RANDOM_RPA_ADDR`

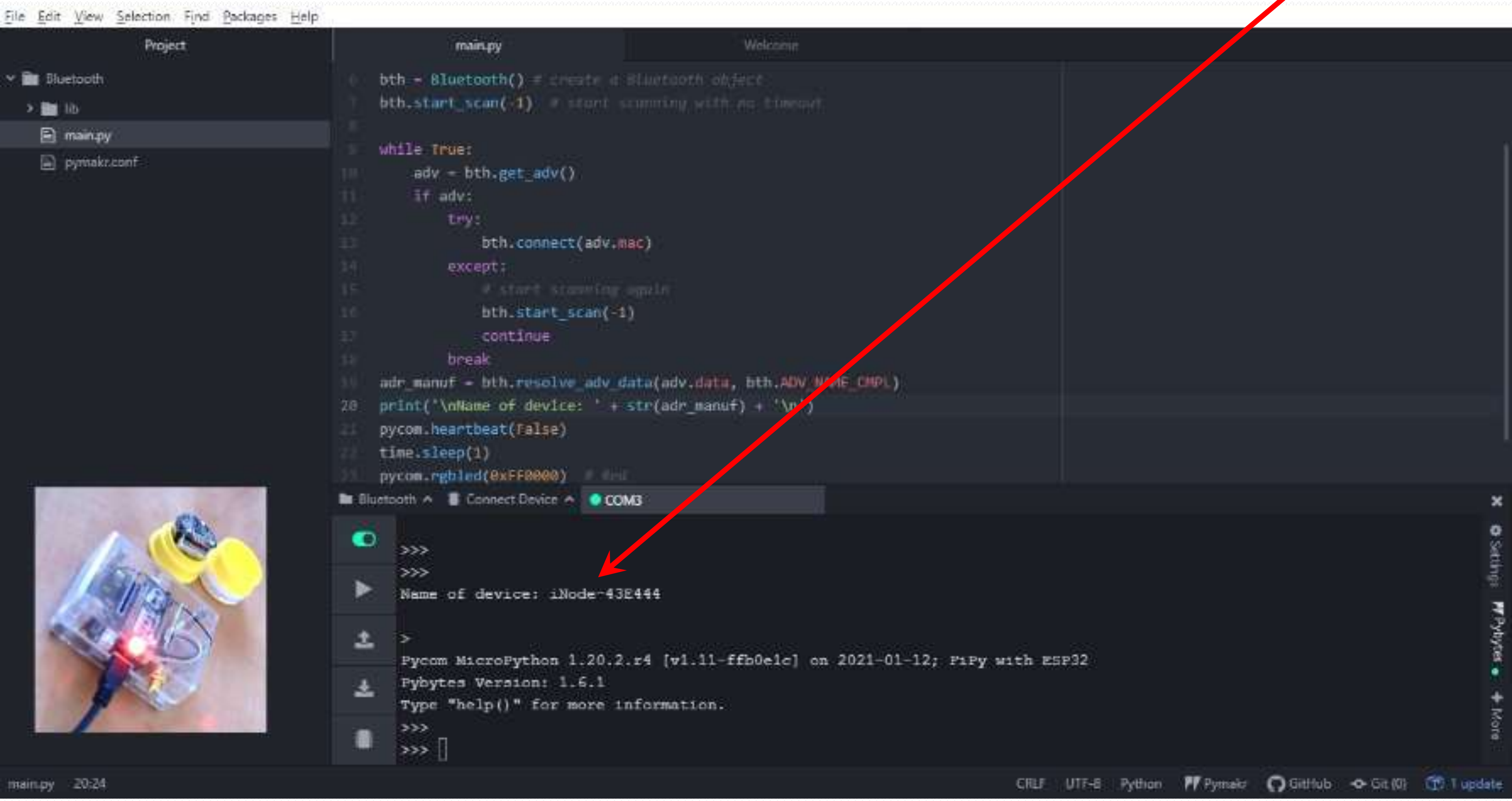
https://docs.pycom.io/firmwareapi/pycom/network/bluetooth/

Part 2 Connect to BLE device and get requested data type:

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

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
```

Part 2 Connect to BLE device and get requested data type:



Part 2 Connect to BLE device and get requested data type:

We used the method:

```
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.

Part 2 Write a 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
```

Constants:

```
Advertisement data type: 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
```

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

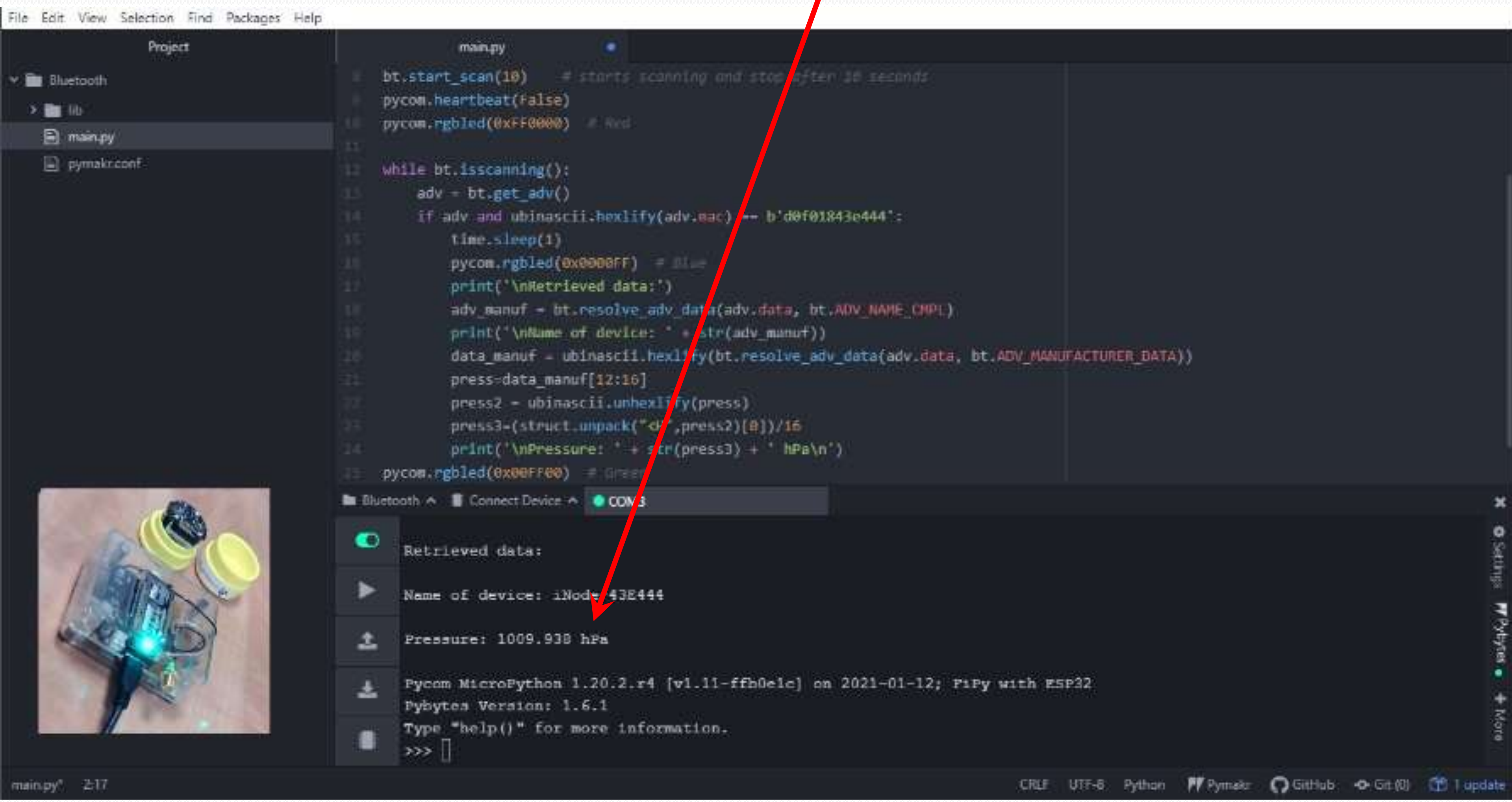
Part 3 Connect to BLE device with known MAC number and receive measurement data:

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

bt = Bluetooth()
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
```

Part 3 Connect to BLE device with known MAC number and receive measurement data:



Part 3 Connect to BLE device with known MAC number and receive measurement data:

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

Part 3 Write a script for the task:

Using iNode Manufacturer Specific Data iNode Care Sensor PHT

converting hexadecimal to ascii by `ubinascii`
decoding Little endian by `struct`

Calculation of humidity (`H[%]`):
`H = (125 * rawHumidity * 4 / 65536) - 6`

Calculation of temperature (`T[°C]`):
`T = (175.72 * rawTemperature * 4 / 65536) - 46.85`

https://inode.pl

Connect to BLE device with known name: `iNode-43E444`
and read the following parameters of the BLE device we connected to:

`Temperature, Humidity`