

## A) Micropython auf einem Wemos D1 mini installieren

1. Board mit Rechner verbinden

2. Treiber für D1 mini herunterladen und installieren: ch341ser\_win.zip, herunterladbar unter:  
<https://wiki.wemos.cc/downloads>

3. Wenn nicht bereits geschehen: Python 3.7 herunterladen und installieren.

*Aufpassen:* Pfad zu Windows hinzufügen, sonst funktioniert der pip-Install über das Internet nicht!

4. Eingabeaufforderung öffnen:

Ausführen > cmd

5. im cmd-Fenster den Befehl ausführen:

`pip install esptool`

6. Gerätemanager öffnen:

Ausführen > `devmgmt.msc`

7. Im Gerätemanager unter Anschlüsse (COM und LPT) den COM-Port des D1 mini herausfinden (z.B. COM3, wird ab jetzt beispielhaft hier im weiteren Tutorial verwendet)

8. In das cmd-Fenster zurückgehen und den Flash-Speicher des D1 mini löschen:

`esptool.py --port COM3 erase_flash`

9. Die aktuelle Firmware („latest stable Firmware“) für ESP8266-Boards hier herunterladen:

<http://micropython.org/download/>

10. Diese Firmware in einen Ordner (z.B. in C:\temp) legen und umbenennen, (z.B. „esp.bin“) und in der Eingabeaufforderung in diesen Ordner wechseln.

11. Die Firmware flashen:

`esptool.py --port COM3 --baud 115200 write_flash --flash_size=detect 0 esp.bin`

12. Überprüfen, ob alles funktioniert hat, Teil 1: CoolTerm herunterladen unter <http://freeware.themeiers.org/CoolTermWin.zip>

Einstellungen auf 115200 Baud, ansonsten defaults; verbinden mit `connect`.

Eintippen des Befehls `print(„hallo welt “*5)` sollte fünf Mal den Text hintereinander ausgeben.

13. Micropython-StandardEinstellung ist die Erzeugung eines Wifi-Accesspoints mit dem Namen

`Micropython-xxxxxx`, wobei die „xxxxx“ die Mac-Adresse des D1 mini ist. Passwort ist `micropython`.

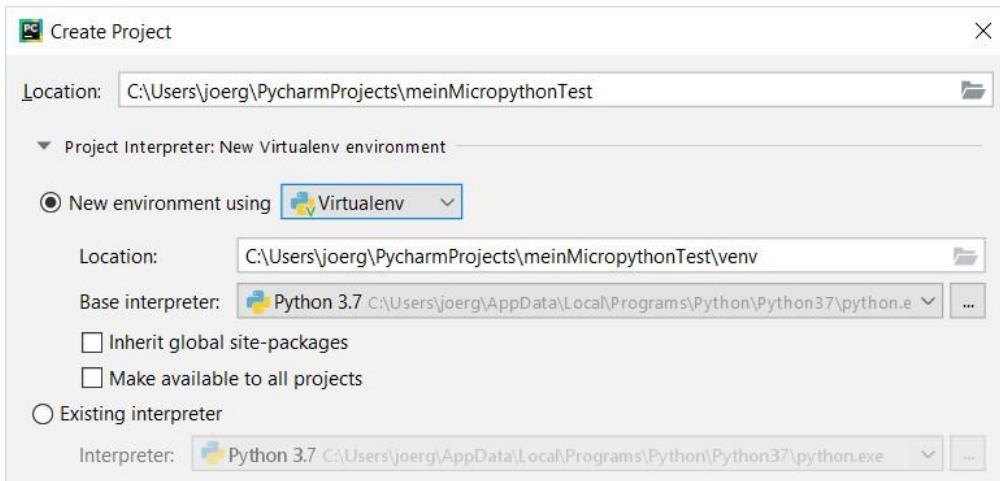
## B) Pycharm installieren und konfigurieren

<https://blog.jetbrains.com/pycharm/2018/01/micropython-plugin-for-pycharm/>

1. Pycharm Community installieren (vorher sollte Python 3.7 installiert sein)

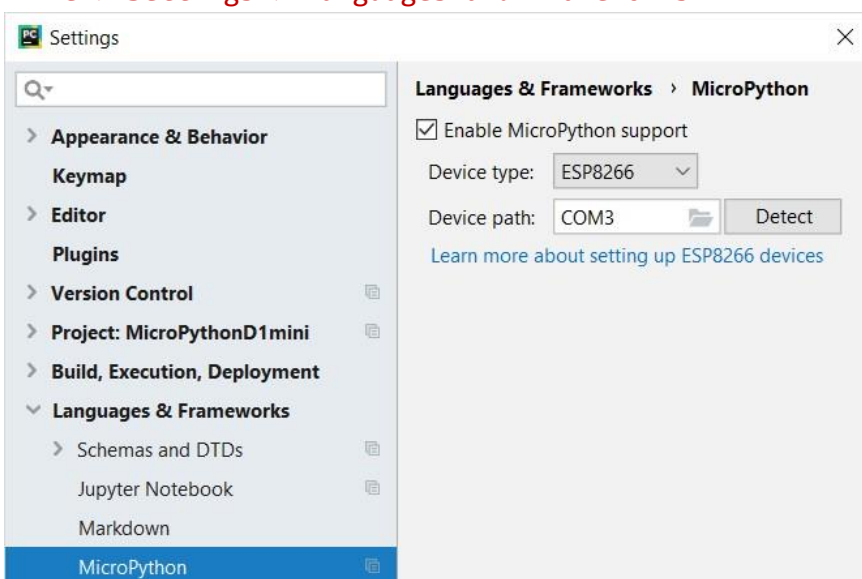
2. Micropython installieren unter: **File > Settings > Plugins** Dort Micropython suchen und installieren.

3. Neues Projekt anlegen mit Einstellung:



4. Konfiguration des D1 mini unter:

**File > Settings > Languages and Frameworks**



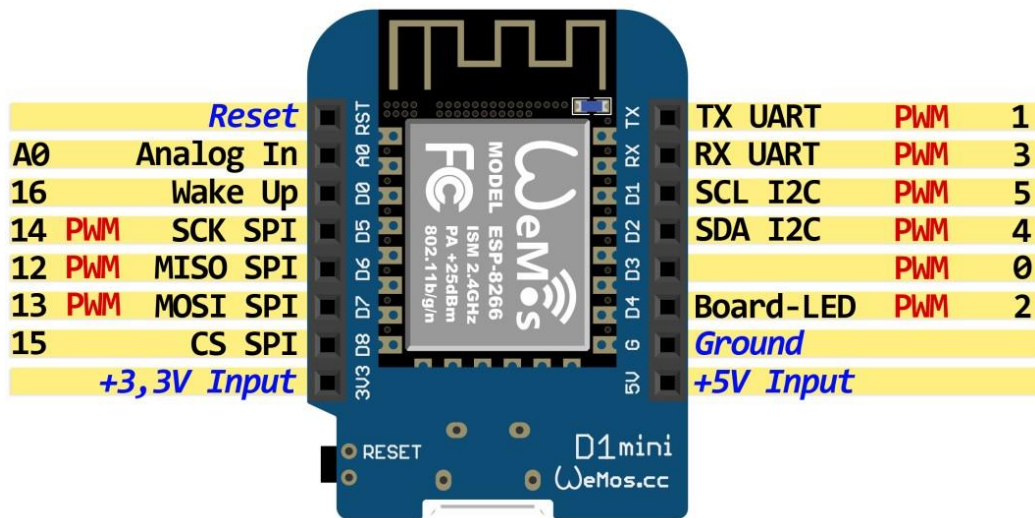
### **ACHTUNG!**

**Beim ersten Start unter Pycharm wird der Device-Path nicht angezeigt, den: Es fehlen noch diverse Treiber, z.B. von Adafruit etc. Diese werden beim Anlegen des ersten Pythonscripts als fehlend angezeigt und lassen sich nachinstallieren. Anschliessend kann man erst per ‚Detect‘ in diesem Fenster den COM-Port auswählen.**

5. Neues Micropython-File in PyCharm anlegen: **ACHTUNG!** Datei muss den Dateinamen besitzen: `main.py`

## C) Python programmieren ☺

<https://micropython-on-wemos-d1-mini.readthedocs.io/en/latest/basics.html#blink>



LED blinken lassen:

```
from machine import Pin
import time

led = Pin(2, Pin.OUT)
while True:
    led(0)
    time.sleep(0.1)
    led(1)
    time.sleep(0.1)
```

LED per PWM (Pulsweitenmodulation) ansteuern, Variante 1:

```
from machine import Pin
from machine import PWM
import time

pwm = PWM(Pin(2))
for j in range(10):
    for i in range(50):
        if i < 25:
            pwm.freq(10+i)
            pwm.duty(i*40)
        if i > 25:
            pwm.freq(60-i)
            pwm.duty(2000-i*40)
            time.sleep(0.1)
```

### 3. LED per PWM (Pulsweitenmodulation) ansteuern, Variante 2:

```
from machine import Pin, PWM
import time, math

pwm = PWM(Pin(2), freq=100)

def meinPuls(l, t):
    for i in range(20):
        l.duty(int(math.sin(i / 10 * math.pi) * 500 + 500))
        time.sleep_ms(t)

for i in range(100):
    meinPuls(pwm, 10)
```

### D) getestete und dokumentierte Beispielprogramme für den D1 Mini

Wifi-Netzwerkscan, alle 5 Sekunden wird wiederholt:

```
import network
import time

meinScanner = network.WLAN(network.STA_IF)
meinScanner.active(True)
while True:
    print("-----")
    scanErgebnis = meinScanner.scan()
    for listenelemente in scanErgebnis:
        print(listenelemente)
    print("-----")
    time.sleep(5)
```

Erklärung: „meinScanner“ ist ein **ST**ation **I**nter**F**ace und schaltet den D1 auf Empfang. Im Hauptteil, also der „while“-Schleife, wird periodisch immer wieder gescannt. Das Ergebnis wird als Liste zurückgegeben und sieht zum Beispiel so aus (aufgenommen in Coolterm):

```
(b'WLAN-M6UCD7',          b'\xd0o\x82\xc7\xab\x10',      1,      -89,      3,      0)
(b'Telekom_FON',         b'\xd0o\x82\xc7\xab\x11',      1,      -93,      0,      0)
(b'DIRECT-FWC410',       b'2\xcd\xa754\xd5',           6,      -71,      3,      0)
(b'6YLAPTOP-B270ICP7',  b'\x8a\xb1\x119` ',           6,      -49,      3,      0)
(b'Physik',              b'\x00$\xb2!\<\xea',          6,      -72,      3,      0)
```

```
(ssid,                    bssid,                          channel, RSSI, authmode, hidden).
```

```
SSID:      Service Set Identifier
BSSID:     Basic Service Set Identifier
RSSI:      Received Signal Strength Indication
```

## OLED-Shield, ssd1306:

Der Bildschirm wird zunächst gelöscht, anschliessend wird in fünf Zeilen jeweils ein kleiner Text geschrieben. Der Text bleibt 10 Sekunden sichtbar, danach wird das Display ausgeschaltet.



```
import ssd1306
from machine import I2C, Pin
import time

i2cObjekt = I2C(sda=Pin(4), scl=Pin(5))           # i2C-Kommunikation initialisieren
oledObjekt = ssd1306.SSD1306_I2C(64, 48, i2cObjekt) # OLED ansprechbar schalten
oledObjekt.fill(0)                               # Bildschirm löschen

oledObjekt.text('Zeile 1', 0, 1)
oledObjekt.text('Zeile 2', 8, 10)
oledObjekt.text('Zeile 3', 0, 20)
oledObjekt.text('Zeile 4', 8, 30)
oledObjekt.text('Zeile 5', 0, 40)
oledObjekt.show()                               # Vorbereitungs-Puffer wird auf Bildschirm kopiert
time.sleep(10)                                  # 10 Sekunden warten ...
oledObjekt.poweroff()                           # ...danach Bildschirm abschalten. Spart Strom ;-)
```

---

Auf dem Bildschirm ist eine kleine Animation zu sehen, die unendlich oft abgespielt wird („while True“): Der Kontrast wird alle 0,1 Sekunden verändert, der aktuelle Kontrastwert wird als Zahlenwert ausgegeben. Dieser Zahlenwert wächst von 1 bis 255 und wird danach wieder auf 0 zurückgesetzt. Aufpassen: Der Befehl „oledObjekt.show()“ benötigt verhältnismäßig viel Zeit. Bei zeitkritischen Anwendungen berücksichtigen.

```
import ssd1306
from machine import I2C, Pin
import time

i2cObjekt = I2C(sda=Pin(4), scl=Pin(5))
oledObjekt = ssd1306.SSD1306_I2C(64, 48, i2cObjekt)
oledObjekt.fill(0)
aktuellerkontrast = 0

while True:
    aktuellerkontrast = aktuellerkontrast + 1   # Kontrastwerte zwischen ...
    aktuellerkontrast = aktuellerkontrast % 255 # ...1 und 255
    oledObjekt.contrast(aktuellerkontrast)      # aktuellen Kontrast setzen
    oledObjekt.text('Kontrast:', 0, 1)
    oledObjekt.text(str(aktuellerkontrast), 20, 15)
    for i in range(0, aktuellerkontrast):
        oledObjekt.pixel(int(aktuellerkontrast/4), 30, 1) # Pixel malen!
    oledObjekt.show()                             # Bildschirm löschen
    time.sleep(0.1)
    oledObjekt.fill(0)
```

## Netzwerk: Wemos als Access-Point

Konfiguration des Wemos als Access-Point, also als Router, in den man sich einloggen kann. Es existieren 5 verschiedene Sicherheitsstufen (open, WEP, WPA\_PSK, WPA2\_PSK, WPA\_WPA2\_PSK), mit denen das Netz konfiguriert werden kann. von diesen Möglichkeiten ist eine im Quelltext aktiv geschaltet, die anderen vier sind auskommentiert. Man kann durch Änderung der Auskommentierung „umschalten“.

```
import network
apObjekt = network.WLAN(network.AP_IF) # AccessPoint-InterFace
apObjekt.active(True) # aktiviert den Wemos als Accesspoint

#apObjekt.config(essid="infAG", authmode=network.AUTH_OPEN)
apObjekt.config(essid="infAG", authmode=network.AUTH_WEP, password="12345678")
#apObjekt.config(essid="infAG", authmode=network.AUTH_WPA_PSK, password="12345678")
#apObjekt.config(essid="infAG", authmode=network.AUTH_WPA2_PSK, password="12345678")
#apObjekt.config(essid="infAG", authmode=network.AUTH_WPA_WPA2_PSK,
password="12345678")

print("\n\nIPCONFIG nach Reset:      ", apObjekt.ifconfig())
print("IP adresse, Netzmaske, Gateway, DNS\n-----\n")

apObjekt.ifconfig(("10.10.0.1", "255.255.255.0", "10.10.0.1", "8.8.8.8"))

print("")
print("IPCONFIG nach Connect: ", apObjekt.ifconfig())
print("IP adresse, Netzmaske, Gateway, DNS\n-----\n")
```

---

## Netzwerk: Wemos als Endpunkt, „Station“

Konfiguration eines zweiten Wemos als Station, die sich mit dem obigen Wemos-Access-Point verbinden kann.

Verbindet sich die Wemos-Station mit dem obigen Access-Point, so erhält sie automatisch eine IP!

Man kann – wenn man möchte – per ‚ifconfig‘ analog dem Accesspoint oben auch eigene IP-Konfiguration setzen.

```
import network

meineStation = network.WLAN(network.STA_IF) # Station-InterFace

print("\n\nIPCONFIG nach Reset:      ", meineStation.ifconfig())
print("IP adresse, Netzmaske, Gateway, DNS\n-----\n")

if not meineStation.isconnected():
    meineStation.active(True)
    meineStation.connect('infAG', '12345678')
    print("ich versuch noch, mich zu verbinden: ", meineStation.isconnected())
    print("mein aktueller Status:", meineStation.status())
    while not meineStation.isconnected():
        pass

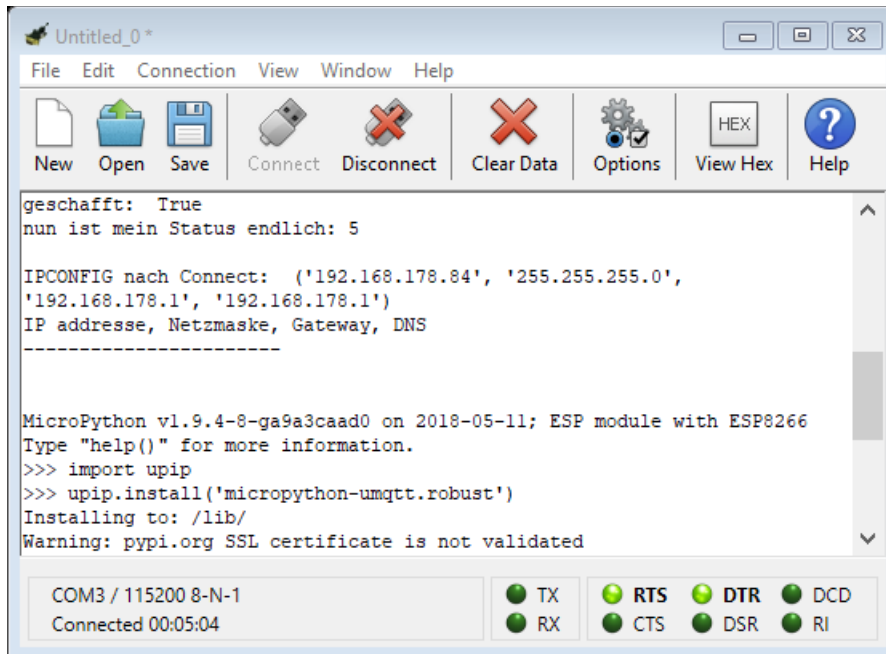
print("geschafft: ", meineStation.isconnected())
print("nun ist mein Status endlich:", meineStation.status())
#0: STAT_IDLE - keine Verbindung,
#1: STAT_CONNECTING - verbindet sich gerade,
#2: STAT_WRONG_PASSWORD - falsches Passwort,
#3: STAT_NO_AP_FOUND - Accesspoint antwortet nicht,
#4: STAT_CONNECT_FAIL - irgendwelche anderen Probleme,
#5: STAT_GOT_IP - Juhu, es hat geklappt.
print("")
print("IPCONFIG nach Connect: ", meineStation.ifconfig())
print("IP adresse, Netzmaske, Gateway, DNS\n-----\n")
```

weitere Infos unter: <http://docs.micropython.org/en/v1.9.3/esp8266/library/network.html>

## MQTT mit dem Wemos

### 1. Installation der MQTT-Bibliotheken auf dem Wemos

Da wir den Wemos als IoT-Gerät konfigurieren wollen, sollte ein MQTT-Protokoll für Maschine-Maschine-Kommunikation darauf laufen. Dies geschieht von der Coolterm-Konsole aus:



```
Untitled_0 *
File Edit Connection View Window Help
New Open Save Connect Disconnect Clear Data Options View Hex Help
geschafft: True
nun ist mein Status endlich: 5

IPCONFIG nach Connect: ('192.168.178.84', '255.255.255.0',
'192.168.178.1', '192.168.178.1')
IP adresse, Netzmaske, Gateway, DNS
-----

MicroPython v1.9.4-8-ga9a3caad0 on 2018-05-11; ESP module with ESP8266
Type "help()" for more information.
>>> import upip
>>> upip.install('micropython-umqtt.robust')
Installing to: /lib/
Warning: pypi.org SSL certificate is not validated

COM3 / 115200 8-N-1
Connected 00:05:04
● TX ● RTS ● DTR ● DCD
● RX ● CTS ● DSR ● RI
```

Im REPL-Mode der Coolterm-Konsole folgende Befehle eingeben:

```
import upip
upip.install('micropython-umqtt.robust')
```

```
from umqtt.robust import MQTTClient
```

Wenn der letzte „**import** MQTTClient „- Befehl keinen Fehler ergibt, dann ist es ‚geschafft‘!

### 2. Installation der Universellen-Identifizier-Bibliotheken auf dem Wemos

Als nächstes benötigt der eigene Wemos einen Universell eindeutigen Identifizierer – also einen individuellen Namen, den man nicht zufällig wiederholen kann. Dieser Name generiert sich der Wemos selbst in Form eines sogenannten UUID (Universally Unique Identifier, siehe [https://de.wikipedia.org/wiki/Universally\\_Unique\\_Identifier](https://de.wikipedia.org/wiki/Universally_Unique_Identifier)) Den generiert man sich selbst mit der uuid-Pythonlibrary, die man genauso wie oben herunterlädt:

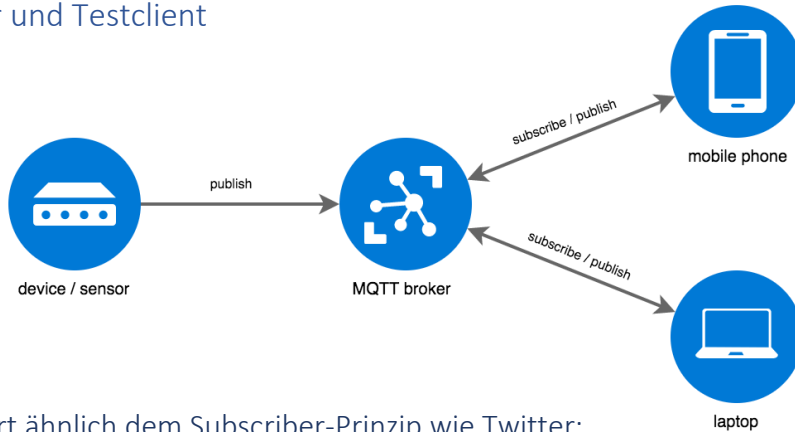
```
import upip
upip.install('micropython-uuid')
```

```
from uuid import uuid4
print (uuid4())
```

Da kommt dann ein Ergebnis raus, was ungefähr so aussieht:

```
4d9bdf54-7e21-42e9-bb7d-6d1c0c83b6c0
```

## 2. MQTT-Broker und Testclient



MQTT funktioniert ähnlich dem Subscriber-Prinzip wie Twitter:

Ein Twitter-User erstellt eine Nachricht, diese wird über den Twitter-Server allen Subscribern zugesendet.

Ein MQTT-Publisher erstellt eine Nachricht, diese wird über den MQTT-Broker allen Subscribern zugesendet.

Das MQTT (Message Queuing Telemetry Transport) Protokoll ist dabei im Vergleich zu http deutlich einfacher und robuster aufgebaut, und das Aufsetzen eines Netzwerks ist einfach und geradeaus.

### 2.1 Installation und Aufsetzen des Brokers

Man muss also zunächst einen **MQTT-Broker** auf einem Rechner im WLAN-Netzwerk erstellen. Dazu nutzt man am besten ‚Mosquitto‘:

<http://www.steves-internet-guide.com/install-mosquitto-broker/#>

Auf dieser Seite findet sich eine aktuelle Portable-Version mit allen benötigten Dateien! Sehr bequem ☺

Nach Download & Auspacken geht man mit einem cmd-Fenster ins mosquitto-Verzeichnis und startet den Broker:

```
C:\WINDOWS\system32\cmd.exe - mosquitto
Microsoft Windows [Version 10.0.17134.523]
(c) 2018 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\joerg>cd..

C:\Users>cd..

C:\>cd tmp

C:\tmp>cd mosquitto

C:\tmp\mosquitto>mosquitto
```

**Das Fenster bitte auf keinen Fall schliessen! Mosquitto läuft nur, solange dieses Fenster geöffnet ist.**



## 2.2 Test-Subscriber (& Client)

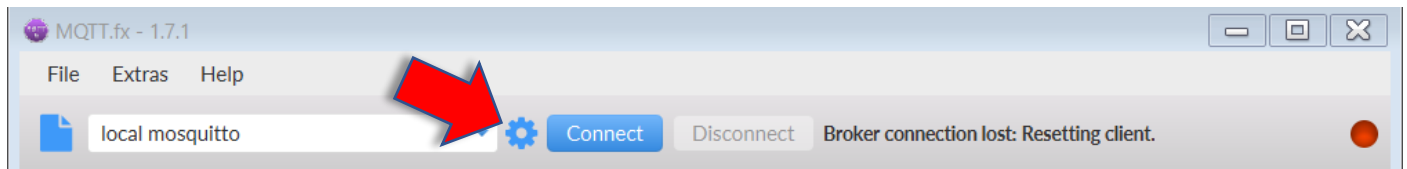
Anschließend benötigt man noch einen Test-Subscriber, den erhält man am einfachsten hier:

<https://mqttfx.jensd.de/>

Den kann man beispielsweise auf dem Broker-Rechner oder auf einem beliebigen weiteren Rechner im WLAN-Netzwerk aufsetzen. Ist MQTT.fx installiert, muss man nur noch die IP des eigenen Mosquitto-Servers eintragen; diese erhält man am besten über den Befehl „ipconfig“ im cmd-Fenster:

```
Drahtlos-LAN-Adapter WLAN:  
  
Verbindungsspezifisches DNS-Suffix: schule.local  
Verbindungslokale IPv6-Adresse . . : fe80::8c8d:57c3:85ec:4b3e%20  
IPv4-Adresse . . . . . : 10.31.239.242  
Subnetzmaske . . . . . : 255.240.0.0  
Standardgateway . . . . . : 10.16.1.1
```

Diese IP trägt man in das Eigenschafts-Fenster von MQTT.fx ein:



Danach kann man MQTT.fx mit dem Broker connecten.

Wemos über Broker mit Client verbinden und Nachrichten senden:

Jetzt nur noch den Wemos an den Broker binden, dann ist man fertig:

```
import network  
  
meineStation = network.WLAN(network.STA_IF) # Station-InterFace  
  
if not meineStation.isconnected():  
    meineStation.active(True)  
    meineStation.connect('Physik', '180130JKG')  
    print("ich versuch noch, mich zu verbinden: ", meineStation.isconnected())  
    print("mein aktueller Status:", meineStation.status())  
    while not meineStation.isconnected():  
        pass  
  
from uuid import uuid4  
meinName = uuid4()  
print (meinName)  
  
from umqtt.robust import MQTTClient  
import utime  
meinKleinerWemos = MQTTClient('meinName', '10.31.239.242')  
meinKleinerWemos.connect()  
  
durchlauf = 0  
while True:  
    meinKleinerWemos.publish('/mein/test', str(utime.ticks_ms()))  
    utime.sleep(3)
```

Falls noch nicht geschehen, sollte man für Zugriffe auf das Wemos-Dateisystem das Adafruit Micropython-Tool installieren. Dazu im cmd-Fenster folgenden Befehl eingeben:

```
pip install adafruit-ampy
```

Anzeigen, welche Dateien sich auf dem Wemos gerade befinden:

```
import os
print("\n\n\n")
print(os.listdir())
```

## I2C-Scanner

```
import machine
i2c = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))

print('Scan i2c bus...')
devices = i2c.scan()

if len(devices) == 0:
    print("No i2c device !")
else:
    print('i2c devices found:', len(devices))

    for device in devices:
        print("Decimal address: ", device, " | Hexa address: ", hex(device))
```

## Der Lichtsensor BH1750:

Hier wird zunächst in einer Datei der Treiber abgespeichert, auf den dann im Hauptprogramm zugegriffen werden kann.

### Datei „ambientL.py“

```
import utime
class BH1750():
    PWR_OFF = 0x00
    PWR_ON = 0x01
    RESET = 0x07

    CONT_LOWRES = 0x13
    CONT_HIRES_1 = 0x10
    CONT_HIRES_2 = 0x11
    ONCE_HIRES_1 = 0x20
    ONCE_HIRES_2 = 0x21
    ONCE_LOWRES = 0x23

    def __init__(self, bus, addr=0x23):
        self.bus = bus
        self.addr = addr
        self.off()
        self.reset()

    def off(self):
        self.set_mode(self.PWR_OFF)

    def on(self):
        self.set_mode(self.PWR_ON)

    def reset(self):
        self.on()
        self.set_mode(self.RESET)

    def set_mode(self, mode):
        self.mode = mode
        self.bus.writeto(self.addr, bytes([self.mode]))

    def luminance(self, mode):
        if mode & 0x10 and mode != self.mode:
            self.set_mode(mode)
        if mode & 0x20:
            self.set_mode(mode)
        utime.sleep_ms(24 if mode in (0x13, 0x23) else 180)
        data = self.bus.readfrom(self.addr, 2)
        factor = 2.0 if mode in (0x11, 0x21) else 1.0
        return (data[0]<<8 | data[1]) / (1.2 * factor)
```

### Datei „main.py“

Das Hauptprogramm bindet via Import-Befehl die Teiberdatei ein:

```
import utime
import machine
from ambientL import BH1750
i2cVerbindung = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))
s = BH1750(i2cVerbindung)

while True:
    print(s.luminance(BH1750.ONCE_HIRES_1))
    utime.sleep_ms(500)
```



File „sht30.py“ (<https://github.com/rsc1975/micropython-sht30>)

```
from machine import I2C, Pin
import time

DEFAULT_I2C_ADDRESS = 0x45

class SHT30():
    POLYNOMIAL = 0x131 #  $P(x) = x^8 + x^5 + x^4 + 1 = 100110001$ 

    ALERT_PENDING_MASK = 0x8000 # 15
    HEATER_MASK = 0x2000 # 13
    RH_ALERT_MASK = 0x0800 # 11
    T_ALERT_MASK = 0x0400 # 10
    RESET_MASK = 0x0010 # 4
    CMD_STATUS_MASK = 0x0002 # 1
    WRITE_STATUS_MASK = 0x0001 # 0

    MEASURE_CMD = b'\x2C\x10'
    STATUS_CMD = b'\xF3\x2D'
    RESET_CMD = b'\x30\xA2'
    CLEAR_STATUS_CMD = b'\x30\x41'
    ENABLE_HEATER_CMD = b'\x30\x6D'
    DISABLE_HEATER_CMD = b'\x30\x66'

    def __init__(self, scl_pin=5, sda_pin=4, delta_temp=0, delta_hum=0,
i2c_address=DEFAULT_I2C_ADDRESS):
        self.i2c = I2C(scl=Pin(scl_pin), sda=Pin(sda_pin))
        self.i2c_addr = i2c_address
        self.set_delta(delta_temp, delta_hum)
        time.sleep_ms(50)

    def init(self, scl_pin=5, sda_pin=4):
        self.i2c.init(scl=Pin(scl_pin), sda=Pin(sda_pin))

    def is_present(self):
        return self.i2c_addr in self.i2c.scan()

    def set_delta(self, delta_temp=0, delta_hum=0):
        self.delta_temp = delta_temp
        self.delta_hum = delta_hum

    def _check_crc(self, data):
        crc = 0xFF
        for b in data[:-1]:
            crc ^= b;
            for _ in range(8, 0, -1):
                if crc & 0x80:
                    crc = (crc << 1) ^ SHT30.POLYNOMIAL;
                else:
                    crc <<= 1
        crc_to_check = data[-1]
        return crc_to_check == crc

    def send_cmd(self, cmd_request, response_size=6, read_delay_ms=100):
        try:
            self.i2c.start();
            self.i2c.writeto(self.i2c_addr, cmd_request);
            if not response_size:
                self.i2c.stop();
                return
            time.sleep_ms(read_delay_ms)
            data = self.i2c.readfrom(self.i2c_addr, response_size)
            self.i2c.stop();
            for i in range(response_size // 3):
                if not self._check_crc(data[i * 3:(i + 1) * 3]): # pos 2 and 5 are CRC
                    raise SHT30Error(SHT30Error.CRC_ERROR)
            if data == bytearray(response_size):
                raise SHT30Error(SHT30Error.DATA_ERROR)
            return data
        except OSError as ex:
            if 'I2C' in ex.args[0]:
                raise SHT30Error(SHT30Error.BUS_ERROR)
            raise ex
```



```

def clear_status(self):
    return self.send_cmd(SHT30.CLEAR_STATUS_CMD, None);

def reset(self):
    return self.send_cmd(SHT30.RESET_CMD, None);

def status(self, raw=False):
    data = self.send_cmd(SHT30.STATUS_CMD, 3, read_delay_ms=20);
    if raw:
        return data
    status_register = data[0] << 8 | data[1]
    return status_register

def measure(self, raw=False):
    data = self.send_cmd(SHT30.MEASURE_CMD, 6);
    if raw:
        return data
    t_celsius = (((data[0] << 8 | data[1]) * 175) / 0xFFFF) - 45 + self.delta_temp;
    rh = ((data[3] << 8 | data[4]) * 100.0) / 0xFFFF + self.delta_hum;
    return t_celsius, rh

def measure_int(self, raw=False):
    data = self.send_cmd(SHT30.MEASURE_CMD, 6);
    if raw:
        return data
    aux = (data[0] << 8 | data[1]) * 175
    t_int = (aux // 0xffff) - 45;
    t_dec = (aux % 0xffff * 100) // 0xffff
    aux = (data[3] << 8 | data[4]) * 100
    h_int = aux // 0xffff
    h_dec = (aux % 0xffff * 100) // 0xffff
    return t_int, t_dec, h_int, h_dec

class SHT30Error(Exception):
    BUS_ERROR = 0x01
    DATA_ERROR = 0x02
    CRC_ERROR = 0x03

    def __init__(self, error_code=None):
        self.error_code = error_code
        super().__init__(self.get_message())

    def get_message(self):
        if self.error_code == SHT30Error.BUS_ERROR:
            return "Bus error"
        elif self.error_code == SHT30Error.DATA_ERROR:
            return "Data error"
        elif self.error_code == SHT30Error.CRC_ERROR:
            return "CRC error"
        else:
            return "Unknown error"

```

File „main.py“

```

from sht30 import SHT30
import time

sensor = SHT30()
while True:
    temperature, humidity = sensor.measure()
    print('Temperature:', temperature, '°C, RH:', humidity, '%')
    time.sleep_ms(1000)

```