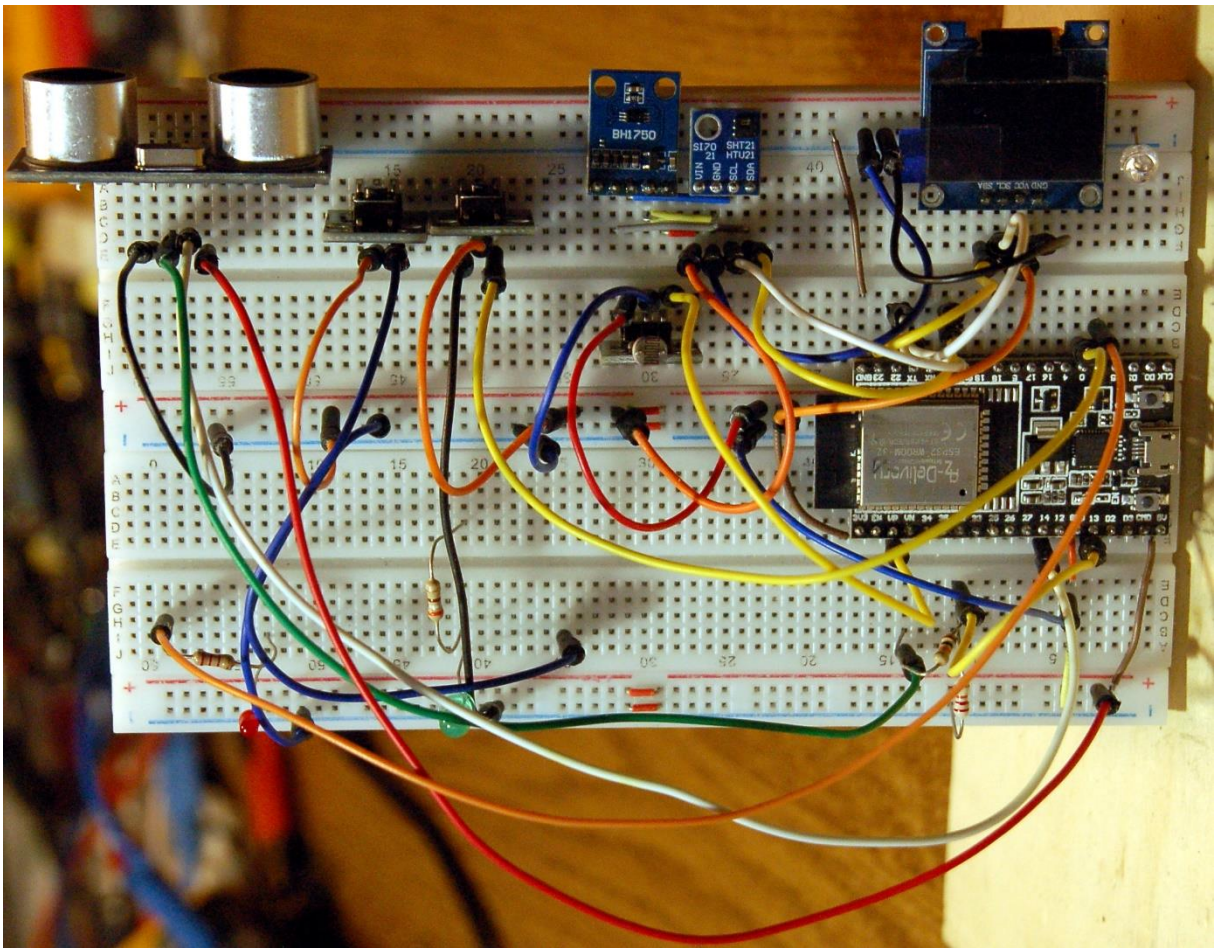

*Aufbau mit ESP32*

Diesen Beitrag gibt es auch als PDF-Dokument.

Nach IFTTT wollen wir uns heute einen weiteren Dienst im Web anschauen. Er wird uns helfen, Messdaten von unserem ESP32 ganz einfach als Grafik-Plot darzustellen. Wie IFTTT ist dieser Service kostenlos und enthält nicht einmal Beschränkungen bezüglich der Anzahl von Applets. War es bei IFTTT beim HTTP-Zugriff die Methode POST, so wird es hier die Methode GET sein. Was ist der Unterschied? Bei GET erfolgt der Request an den Server in einem Zugriff. Sie kennen das vielleicht von der Darstellung von Suchanfragen, etwa der folgenden Form.

https://server.domain.com/dateien/templates?app=openoffice&revision=23

Aus solchen Zeilen baut der Browser einen GET-Request zusammen. Diese Art Anfragen sind auf relativ wenige Zeichen begrenzt, wie viele maximal in Frage kommen entscheidet der jeweilige Server.

Ein POST-Request geschieht in zwei Schritten. Zuerst wird eine Verbindung aufgebaut und im zweiten Schritt werden die Daten verschifft. Diese Art von Anfrage wir verwendet, wenn viele Daten gesendet werden sollen, etwa aus einem HTML-Formular.

Manche Server akzeptieren beide Methoden, andere schreiben die Methode vor. Sie können das ja gerne ausprobieren. Wie ein POST zusammengesetzt wird, zeigt der Beitrag zu IFTTT.

Aber jetzt werfen wir erst einmal einen Blick auf ThingSpeak. So heißt der Dienst, den ich ihnen vorstellen werde. In dieser Episode aus der Reihe

# MicroPython auf dem ESP32 und ESP8266

heute

## ThingSpeak plottet die Werte vom ESP32

Sie vermissen hier den ESP8266? Ja, den habe ich weggelassen, weil sein Speicher für den Treiber des BME280 gehörig zu schmalbrüstig ist, was den Speicherplatz anbelangt. Grundsätzlich können natürlich auch die Werte von einem ESP8266 an ThingSpeak gesendet werden. Nur müssen Sie dann auf andere Sensoren ausweichen, die einen schlankeren Treiber benutzen, wie zum Beispiel der SHT21 oder der AHT10. Bei diesen Modulen fehlt aber der Luftdrucksensor.

Außerdem ist heute ein weiterer Sensor mit an Bord, ein BH1750. Er wird, wie der BME280, über den I2C-Bus angesteuert und liefert Helligkeitswerte in Lux. Chef vom Dienst ist bei mir ein ESP32 Dev Kit V4. Natürlich eignen sich auch die anderen Familienmitglieder, die in der folgenden Liste aufgeführt sind.

### Hardware

Um den Zustand der Schaltung jederzeit auch direkt vor Ort einsehen zu können, habe ich dem ESP ein Display spendiert. Über die Flash-Taste ist ein geordneter Abbruch des Programms möglich, falls zum Beispiel Aktoren sicher ausgeschaltet werden müssen, wie hier die LED.

| | |
|---|---|
| 1 | ESP32 Dev Kit C unverlötet oder<br>ESP32 Dev Kit C V4 unverlötet oder<br>ESP32 NodeMCU Module WLAN WiFi Development Board mit CP2102 oder<br>NodeMCU-ESP-32S-Kit oder<br>ESP32 Lolin LOLIN32 WiFi Bluetooth Dev Kit |
| 1 | GY-302 BH1750 Licht Sensor Helligkeitsensor |
| 1 | GY-BME280 Barometrischer Sensor für Temperatur, Luftfeuchtigkeit und Luftdruck |
| 1 | LED, zum Beispiel rot |
| 1 | Widerstand 330 Ω |
| 2 | MB-102 Breadboard Steckbrett mit 830 Kontakten |
| 1 | KY-004 Taster Modul |
| diverse | Jumper Wire Kabel 3 x 40 STK. je 20 cm M2M/ F2M / F2F evtl. auch 65Stk. Jumper Wire Kabel Steckbrücken für Breadboard |
| optional | Logic Analyzer |

Damit neben dem Controller noch Steckplätze für die Kabel frei sind, habe ich zwei Breadboards, mit einer Stromschiene dazwischen, zusammengesteckt.

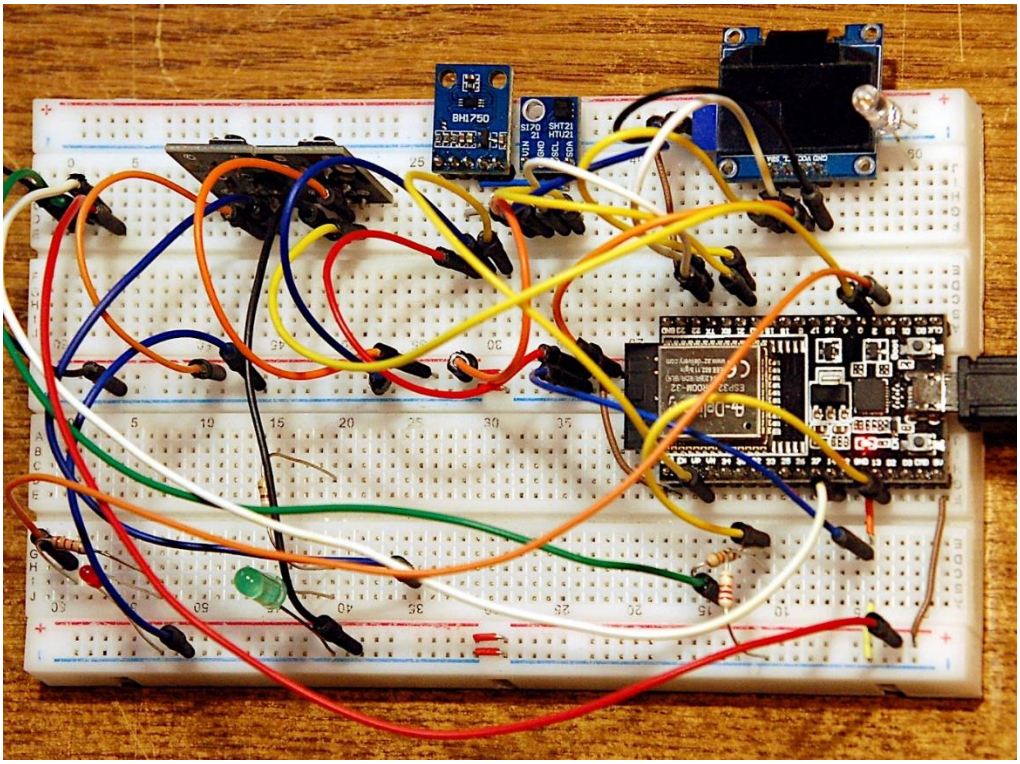


*Abbildung 1: Aufbau Ambiant-Meter*

Und hier ist der Schaltplan.

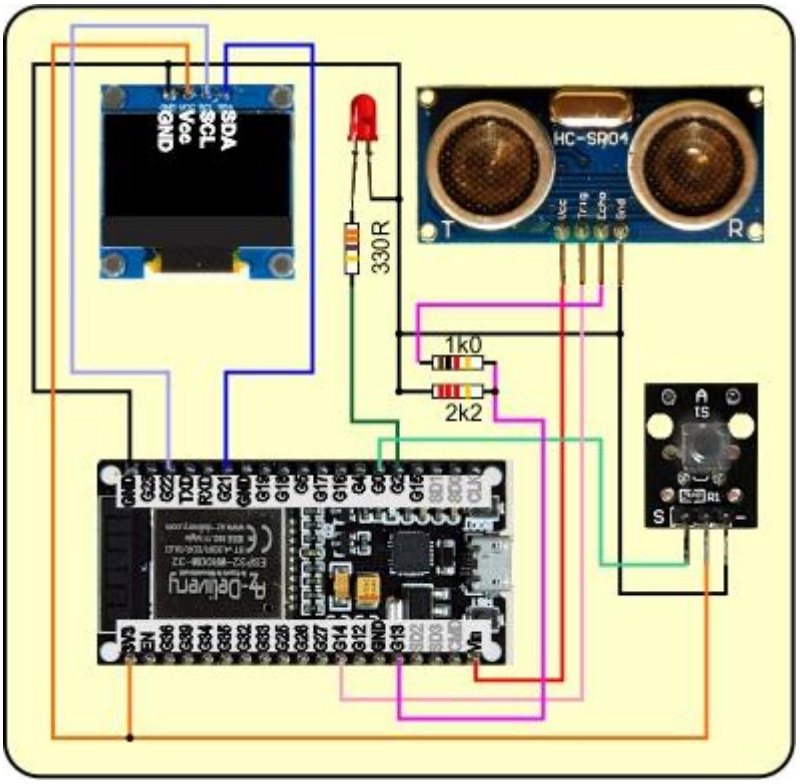


*Abbildung 2: Schaltung Ambiant-Meter*

# Die Software

## Fürs Flashen und die Programmierung des ESP32:

Thonny oder
µPyCraft

## Verwendete Firmware für den ESP32:

v1.19.1 (2022-06-18) .bin

## Verwendete Firmware für den ESP8266:

**v1.19.1 (2022-06-18) .bin**

## Die MicroPython-Programme zum Projekt:

ssd1306.py Hardwaretreiber für das OLED-Display
oled.py API für das OLED-Display
bme280.py Treiber für das BME280-Modul
bh1750.py Treiber für den Lichtsensor BH1750
urequests.py Treibermodul für den HTTP-Betrieb
timeout.py Softwaretimer-Modul
thingSpeak.py Demoprogramm für ThingSpeak

# MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung (english version). Darin gibt es auch eine Beschreibung, wie die Micropython-Firmware (Stand 18.06.2022) auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, damit der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm kompilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Makro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

## Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

## Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton, oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

## Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein kompiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

# Wir bauen einen ThingSpeak-Account

Folgen sie mir auf die Startseite von ThingSpeak.

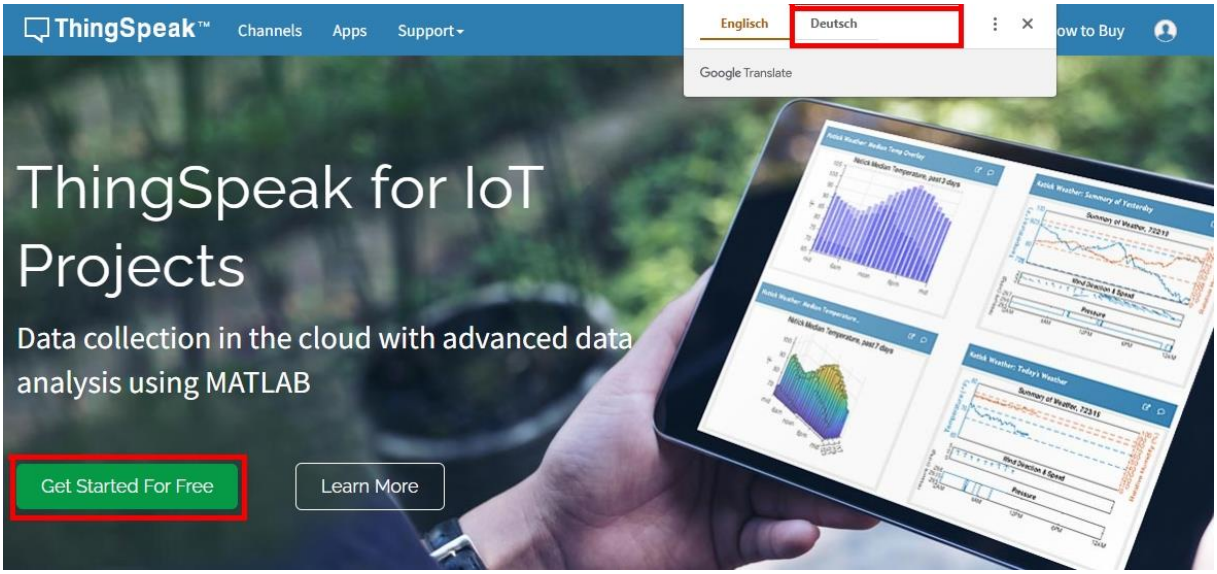


*Abbildung 3: Startfenster*

Beim ersten Aufruf erscheint oben die Sprachauswahl. Weiter geht es mit **Get startet for free**.


*Abbildung 4: Account einrichten*

Als erstes muss man einen MathWorks-Account einrichten. Geben Sie hier Ihre e-Mail-Adresse ein. Mit **Next** geht es weiter zur Erfassung einiger persönlicher Daten.

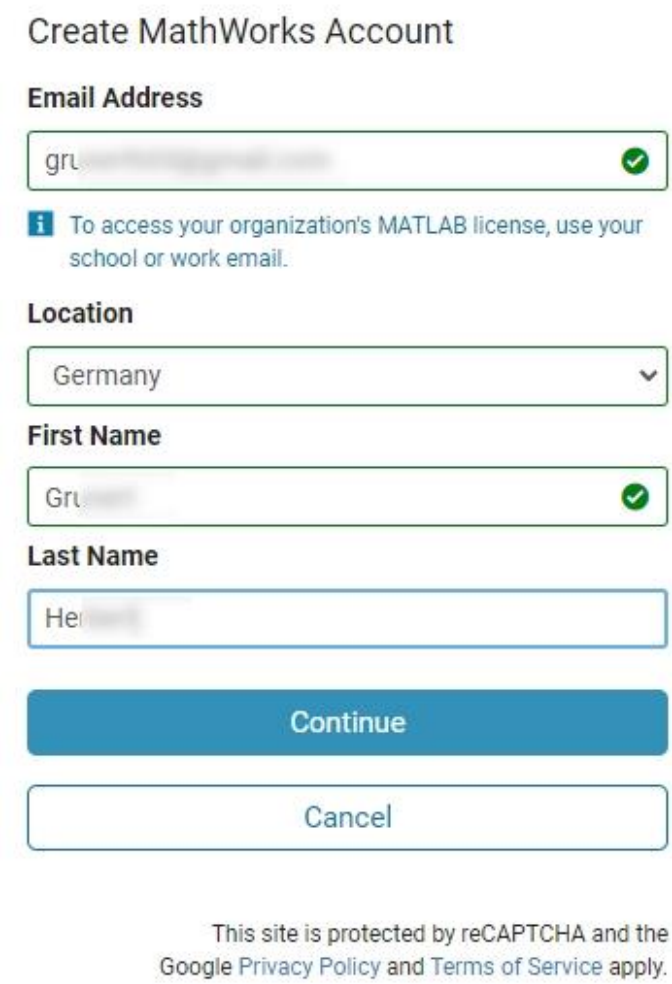


*Abbildung 5:Persönliche Daten erfassen*

**Continue**, führt zum nächsten Fenster. Setzen Sie den Haken bei **Use this email for my MathWorks Account** – **Continue**.

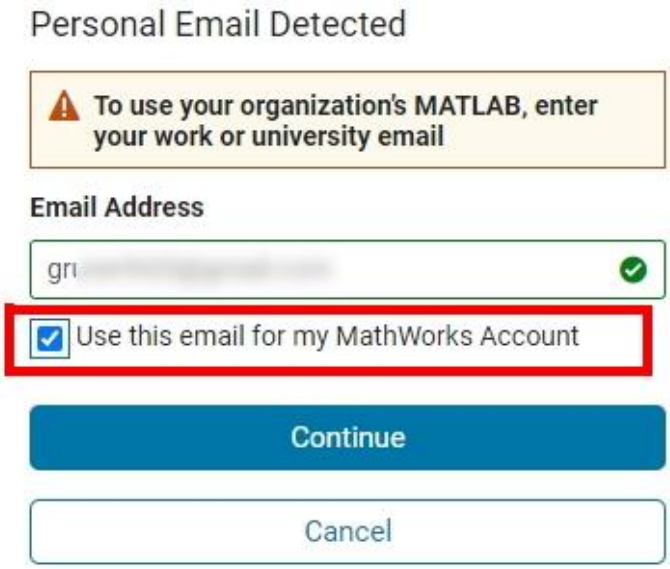


*Abbildung 6: Private Adresse für den Account verwenden*

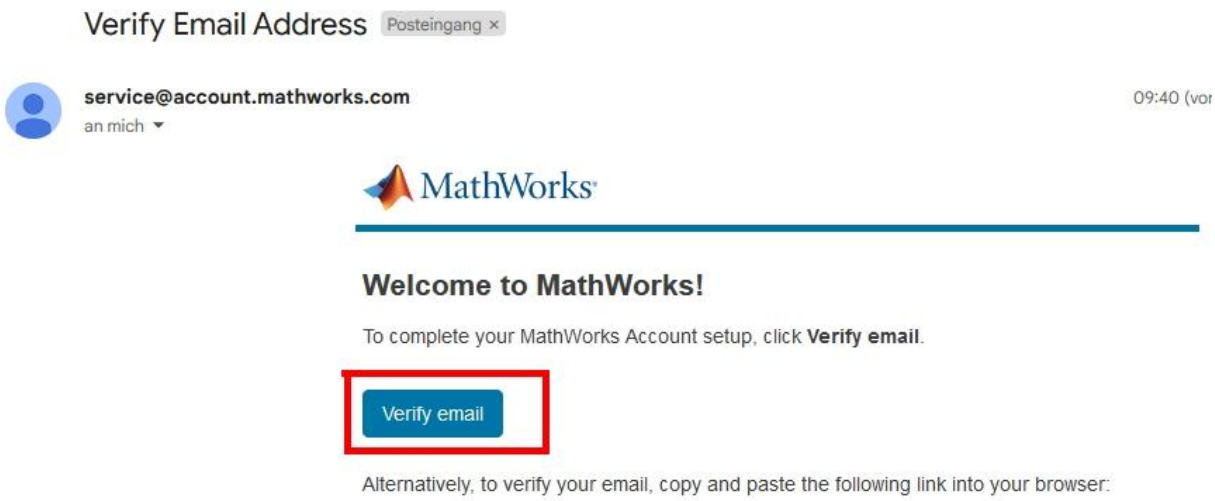


*Abbildung 7: Mailadresse verifizieren*

Jetzt sollten Sie in Ihre Mailbox schauen, ob Sie eine Mail bekommen haben, mit der Sie per Link Die Mail-Adresse bestätigen.

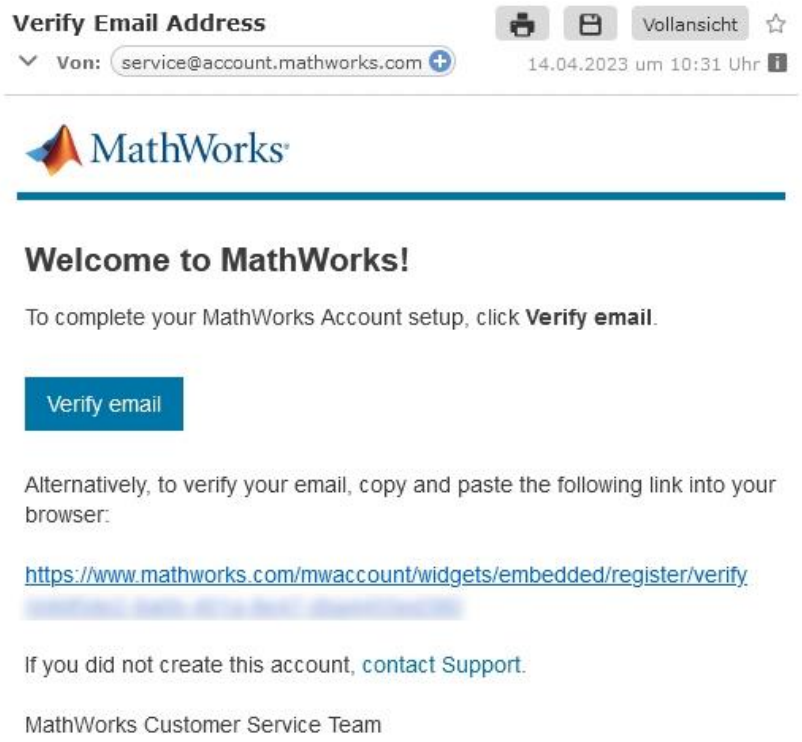


*Abbildung 8: e-Mailadresse bestätigen*

Nun wird noch ein Passwort für den neuen Account erwartet. Es muss nicht das von Ihrem e-Mail-Provider sein. Achtung, das Passwort wird nicht verifiziert, das Häkchen setzen und - **Continue**.

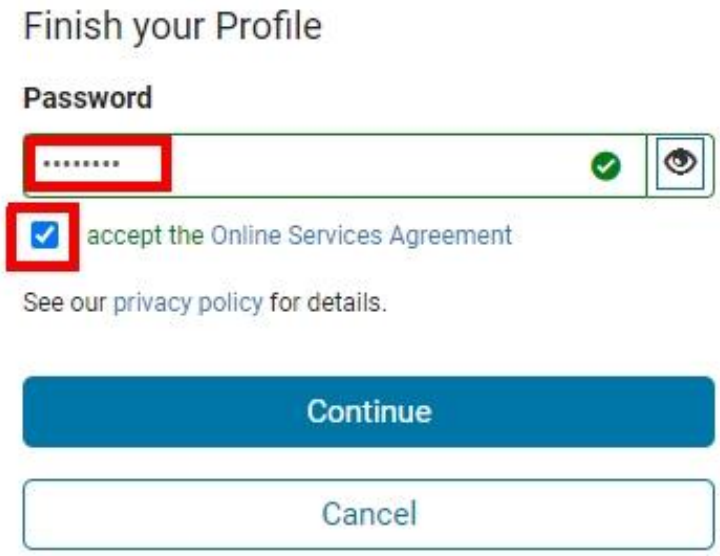


*Abbildung 9: Passwort erzeugen*

Im nachfolgenden Fenster wählen Sie einen der Anwendungsfälle aus. Dann landen Sie auf der Seite auf der Sie Ihre Kanäle verwalten können. Weil noch keiner existiert, klicken Sie auf **Neuer Kanal**. Falls Sie von der Startseite von ThingSpeak kommen, öffnen Sie im Menü **Kanäle** den Punkt **Meine Kanäle**. Damit landen Sie auch auf der Verwaltungseite.


*Abbildung 10: Startfenster nach Erstellung des Accounts*

Klicken Sie auf **Neuer Kanal**.

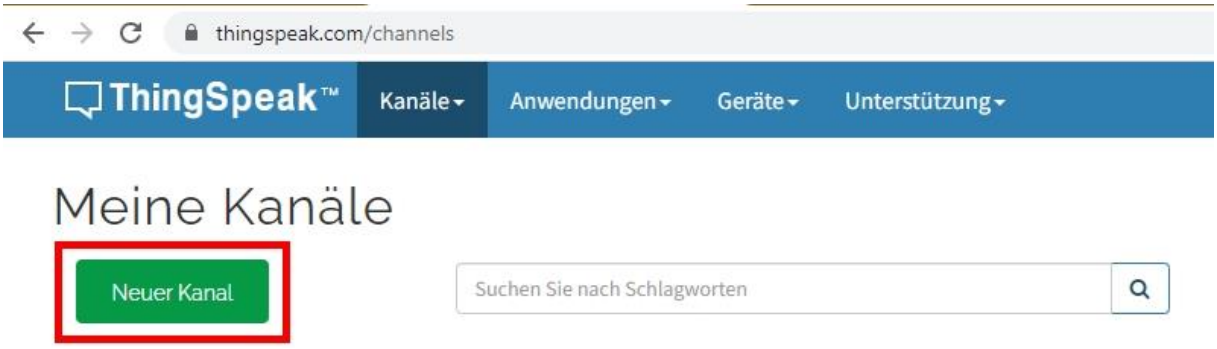


*Abbildung 11: Neuen Kanal erzeugen*

Für unser Projekt brauchen wir vier Felder für Temperatur, rel. Luftfeuchte, Luftdruck und Helligkeit. Als Name habe ich Tageswerte gewählt. Bevor die Felder beschriftet werden können, muss das Häkchen gesetzt werden. Dann – **Save Channel**.

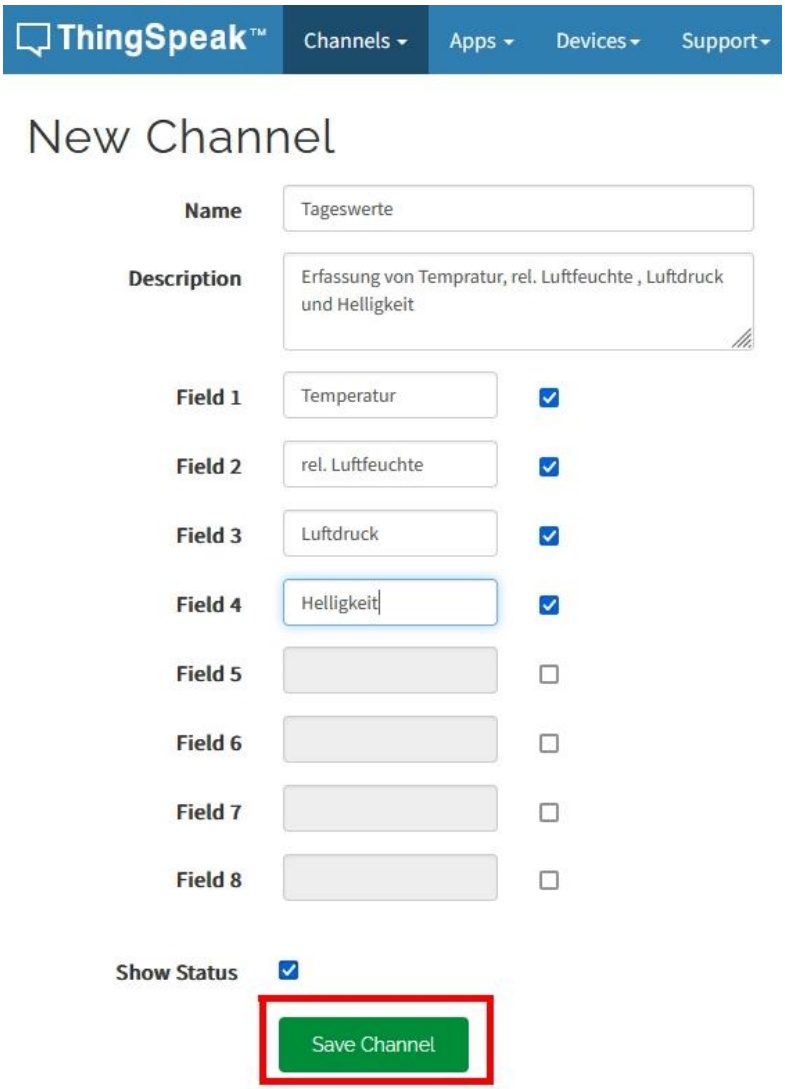


*Abbildung 12:Meine Felder*

Als Zusammenfassung bekommen wir eine Übersicht angezeigt.

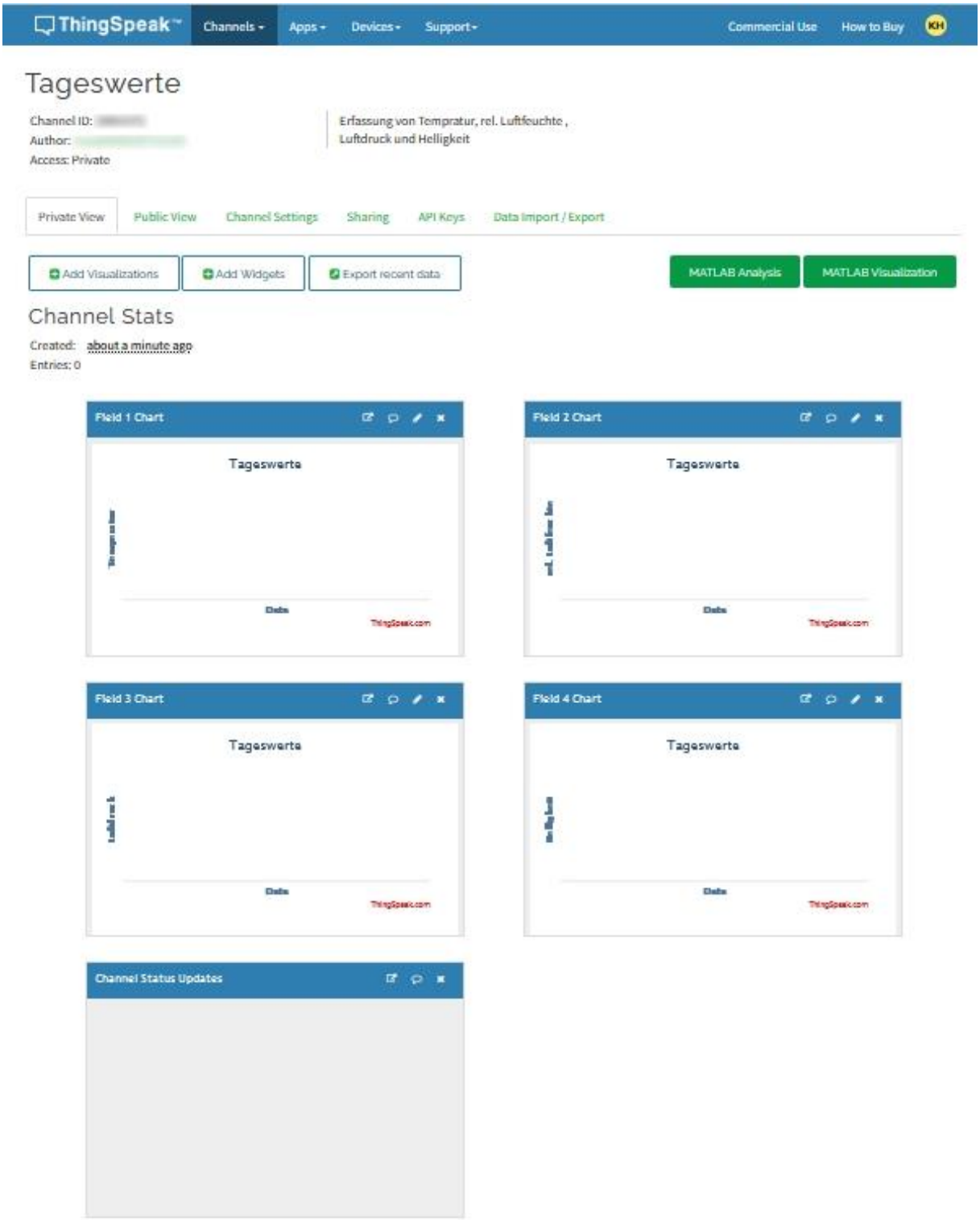


*Abbildung 13: Zusammenfassung*

Im Ordner API Keys bekommen Sie alle erforderlichen Informationen, die zum Senden von Requests nötig sind. Da sind erst einmal die Schlüssel für den Schreib und Lesezugriff.

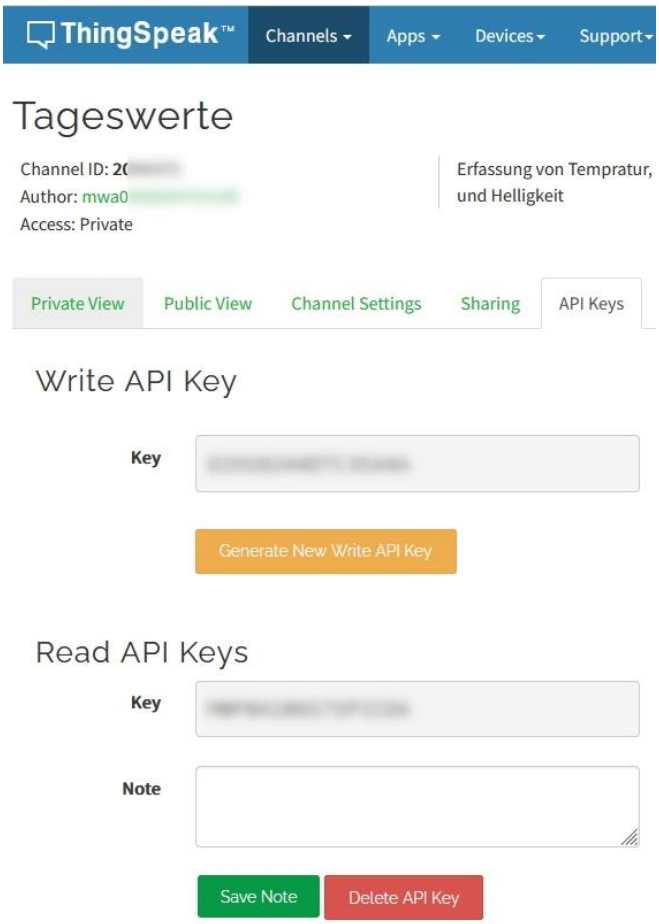


*Abbildung 14: Die Zugriffsschlüssel für schreiben und lesen*

Außerdem finden Sie rechts unten die URL-Zeilen für den Schreib- und Lese-Zugriff. Die Zeilen können ohne das **GET** in das Adressfeld eines Browsers kopiert werden, um den gewünschten Vorgang auszulösen. Wir werden die Zeile für das Schreiben in erweiterter Form für unsere vier Messwerte ins Programm kopieren.

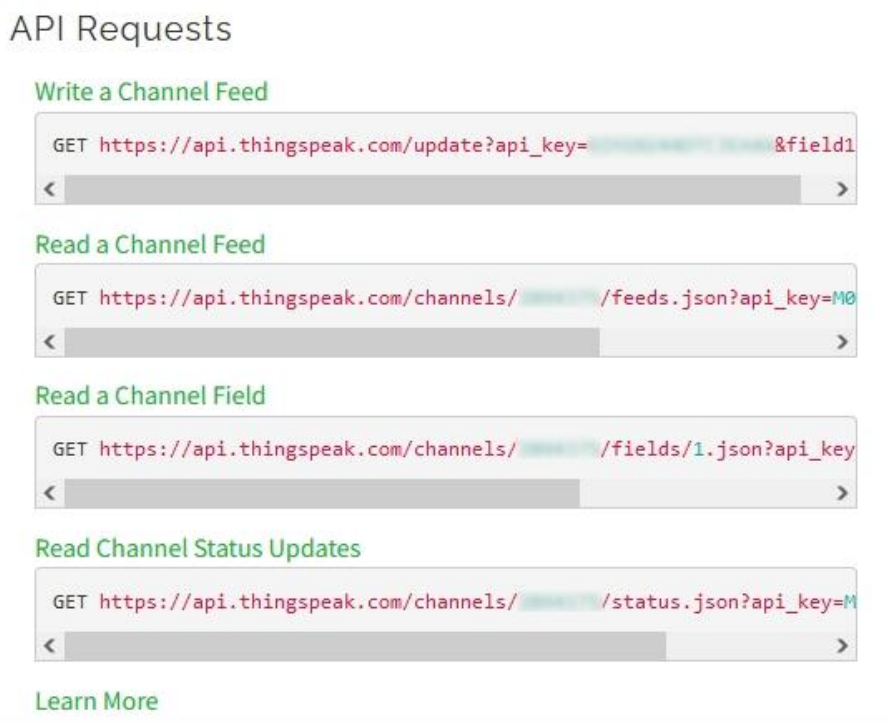


*Abbildung 15: Zugriff auf den Kanal mit GET*

## Das Programm

Wie üblich beginnen wir mit dem Importgeschäft. Folgende Dateien müssen zuvor in den Flash des ESP32 hochgeladen werden: ssd1306.py, oled.py, bh1750.py, bme280.py und timeout.py. Alles andere wird durch den Kernel von MicroPython zur Verfügung gestellt.

```
from machine import Pin, SoftI2C
from time import sleep
from oled import OLED
from bh1750 import BH1750
from bme280 import BME280
import requests
import network, socket
import sys, os
from timeout import *
import gc
```

Eine Besonderheit stellt der Import von **timeout** dar. Durch den Stern werden sämtliche Objekte direkt in den globalen Scope (ak Namensraum) des Hauptprogramms **ThingSpeak**.py übernommen. Damit entfällt das, sonst übliche, Objekt-Prefix beim Aufruf der Funktionen.

Es folgt die Bekanntgabe der API-Schlüssel und der Kanalnummer.

```
thspkWriteKey="---Ihr Schreibschlüssel---"
thspkReadKey="---Ihr Leseschlüssel---"
thspkCannel="---Ihre Kanalnummer---"

intervall=1000 # ms bis zur ersten Erfassung
folgeIntervall=60000

mySSID = 'EMPIRE_OF_ANTS'; myPass = "nightingale"
```

Das Intervall vom Start der Hauptschleife bis zur ersten Erfassung setze ich hier auf 1 eine Sekunde, die Folgeintervalle auf eine Minute. Es folgen die Credentials für die Kontaktaufnahme mit dem WLAN-Router.

Der Programmbaustein für das Instanziieren des I2C-Busses wäre hier nicht erforderlich, weil eh nur ein ESP32 das Programm stemmen kann. Ich spare mir halt nach dem Einfügen durch copy and paste das Trimmen.

```
if sys.platform == "esp8266":
    i2c=SoftI2C(scl=Pin(5),sda=Pin(4))
elif sys.platform == "esp32":
    i2c=SoftI2C(scl=Pin(22),sda=Pin(21))
else:
    raise RuntimeError("Unknown Port")
```

Mit dem I2C-Objekt erzeugen wir gleich das Display-Objekt, die BME280- und die BH1750-Instanz.

```
d=OLED(i2c,heightw=64) # 128x64-Pixel-Display
d.contrast(255)
d.writeAt("AMBIANT DATA",2,0)

bme=BME280(i2c)
bh=BH1750(i2c)
```

Es folgen ein Tasten- und LED-Objekt.

```
taste=Pin(0,Pin.IN,Pin.PULL_UP)
red=Pin(14,Pin.OUT,value=0)
```

Das Dictionary (kurz Dict) connectStatus übersetzt die nichtssagenden Nummern, die die Funktion **nic.status**() beim Verbindungsaufbau mit dem Router zurückgibt, in Klartext.

```
connectStatus = {
    1000: "STAT_IDLE",
    1001: "STAT_CONNECTING",
    1010: "STAT_GOT_IP",
    202:  "STAT_WRONG_PASSWORD",
    201:  "NO AP FOUND",
    5:    "UNKNOWN",
    0: "STAT_IDLE",
    1: "STAT_CONNECTING",
    5: "STAT_GOT_IP",
    2:  "STAT_WRONG_PASSWORD",
    3:  "NO AP FOUND",
    4:  "STAT_CONNECT_FAIL",
    }
```

Mit der Funktion **hexMac**() erfahren wir die MAC-Adresse des Station-Interfaces des Controllers im Klartext. Diese muss im Router eingetragen werden, sonst verweigert dieser dem Controller den Zugang. Meistens geschieht der Eintrag über das Menü WLAN – Sicherheit. Das genaue Vorgehen verrät das Handbuch des Routers.

Wir tasten das Bytes-Objekt, das uns der Funktionsaufruf **nic.config('mac')** liefert, Zeichen für Zeichen ab, machen daraus eine Hexadezimalzahl und bauen daraus den String auf, den die Funktion zurückgibt.

```
def hexMac(byteMac):
  """
  Die Funktion hexMAC nimmt die MAC-Adresse im Bytecode
  entgegen und bildet daraus einen String fuer die Rueckgabe
  """
  macString =""
  for i in range(0,len(byteMac)):     # Fuer alle Bytewerte
    macString += hex(byteMac[i])[2:]  # ab Position 2 bis Ende
    if i <len(byteMac)-1 :            # Trennzeichen
      macString +="-"
  return macString
```

Händisch sieh das so aus:

**>>> nic.config('mac')**
*b'\xf0\x08\xd1\xd1m\x14'*

**>>> hex(m[0])**
*'0xf0'*

**>>> hex(m[0]) [2:]**
*'0xf0'*

**>>> hex(m[0])[2:]**
*'f0'*

Es folgt die Verbindungsaufnahme mit dem WLAN-Router. Dazu schalten wir das Station-Interface ein, der Controller arbeitet ja als Client. Die Schaltsekunde danach ist wichtig und vermeidet interne WLAN-Fehler, die sporadisch auftreten.

```
# ********************* Bootsequenz ************************
#
nic = network.WLAN(network.STA_IF)  # erzeugt WiFi-Objekt nic
nic.active(True)                    # nic einschalten
sleep(1)

MAC = nic.config('mac') # binaere MAC-Adresse abrufen und
myMac=hexMac(MAC)    # in eine Hexziffernfolge umgewandelt
print("STATION MAC: \t"+myMac+"\n") # ausgeben
```

Wir lesen die MAC-Adresse aus, lassen sie in Klartext umformen und im Terminal ausgeben.

```
if not nic.isconnected():
  nic.connect(mySSID, myPass)
  print("Status: ", nic.isconnected())
  d.writeAt("WLAN connecting",0,1)
  points="..........."
  n=1
  while nic.status() != network.STAT_GOT_IP:
    print(".",end='')
    d.writeAt(points[0:n],0,2)
    n+=1
    sleep(1)
```

Wenn die Schnittstelle noch keine Verbindung zum Router hat, stellen wir mit **connect**() eine her. Dabei werden SSID und Passwort übertragen. Der Status wird abgefragt und ausgegeben. Solange wir vom DHCP-Server auf dem Router noch keine IP-Adresse bekommen haben, wird im Sekundenabstand ein Punkt ausgegeben. Das sollte nicht länger als 4 bis 5 Sekunden dauern.

Dann fragen wir den Status ab und lassen uns die Verbindungsdaten, IP-Adresse, Netzwerkmaske und Gateway, mitteilen.

```
print("\nStatus: ",connectStatus[nic.status()])
d.clearAll()
STAconf = nic.ifconfig()
print("STA-IP:\t\t",STAconf[0],"\nSTA-NETMASK:\t",STAconf[1],\
                    "\nSTA-GATEWAY:\t",STAconf[2] ,sep='')
d.writeAt(STAconf[0],0,0)
d.writeAt(STAconf[1],0,1)
d.writeAt(STAconf[2],0,2)
sleep(3)
d.clearAll()
d.writeAt("AMBIANT DATA",2,0)
```

```
nextMeasurement=TimeOutMs(intervall)
```

Die erste Messwertaufnahme wird vorbereitet, indem wir den Timer **nextMeasurement** auf 1000 ms stellen. Dann geht es in die Mainloop, die nur zwei Events bedient, den abgelaufenen Timer und die Tastenabfrage.

Dazu noch eine kurze Bemerkung. **TimeOut**() ist eine Funktion, in deren Codekörper eine weitere Funktion mit dem Namen **compare**() deklariert wird. **compare**() ist eine sogenannte Closure. Das besondere daran ist, dass diese Funktion nach dem Verlassen der umschließenden Funktion **TimeOut**() nicht kompostiert wird, sondern weiterhin referenziert werden kann. Das wird durch zwei Umstände möglich gemacht. Der eine ist, dass **compare**() die an **TimeOut**() übergebene Zeit referenziert und ferner den Zeitstempel in **start**. Der zweite Grund ist, dass **TimeOut**() eine Referenz auf **compare**() zurückgibt. Wir speichern sie hier in **nextMeasurement** ab. **nextMeasurement** ist damit callable, wir referenzieren beim Aufruf letztlich die Closure **compare**(). Wenn Sie mehr über Closures erfahren wollen, folgen Sie diesem Link.

```
def TimeOutMs(t):
    start=ticks_ms()

    def compare():
        nonlocal start
        if t==0:
            return False
        else:
            return int(ticks_ms()-start) >= t
    print (id(compare))
    return compare
```

**>>> nextMeasurement=TimeOutMs(intervall)**
*1073713376*
**>>> id(nextMeasurement)**
*1073713376*

Hier haben Sie den Beweis, die IDs von **compare**() und **nextMeasurement**() sind identisch.

In der Hauptschleife fragen wir zuerst den Timer ab, indem wir die Funktion **compare**() via **nextMeasurement**() aufrufen. Solange der Timer noch läuft, erhalten wir als Rückgabe **False**. Ist aber **ticks_ms – start** gößer als 1000, bekommen wir **True** und betreten den Codekörper des if-Konstrukts.

```
while 1:
    if nextMeasurement():
        red.on()
        nextMeasurement=TimeOutMs(folgeIntervall)
        temp=bme.calcTemperature()
        hum=bme.calcHumidity()
        pres=bme.calcPressureNN()
        lum=bh.luminanz()
```

Die LED wird angeschaltet und der Timer neu gestartet, dieses Mal mit dem 60-Sekunden-Intervall. Danach holen wir die Werte von den Sensoren und geben sie auf dem Display aus. Das **False** in den ersten vier Zeilen bewirkt, dass der Text nur in den Datenpuffer im ESP32 geschrieben wird. In der 5. Zeile kommt der optionale Keyword-Parameter **show** mit dem Defaultwert **True** zum Tragen. Erst jetzt wird der Pufferinhalt zum Display geschickt. Dieses Vorgehen beruhigt das Flackern der Anzeige.

```
        d.clearFT(0,1,15,4,False)
        d.writeAt("Temp: {:.2f}".format(temp),0,1,False)
        d.writeAt("rHum: {:.2f}".format(hum),0,2,False)
        d.writeAt("Pres: {:.2f}".format(pres),0,3,False)
        d.writeAt("Lumi: {:.2f}".format(lum),0,4)
```

Dann senden wir einen GET-Request an ThingSpeak. Alles, was wir dazu tun müssen ist, einen URL-String nach dem oben genannten Muster zu erzeugen. Die Zahlenwerte basteln wir durch die Formatierungsstrings {:.2f} in den Text. Wir senden in einem Abwasch alle vier Werte.

```
r=requests.get("https://api.ThingSpeak.com/update?api_key="+
thspkWriteKey+"&field1={:.2f}".format(temp)+"&field2={:.2f}".f
ormat(hum)+"&field3={:.2f}".format(pres)+"&field4={:.2f}".form
at(lum))
        print(r.reason.decode(),r.text)
        status=r.reason.decode()+"; "+r.text
        d.writeAt(status,0,5)
        r.close()
```

Die Antwort vom Server legen wir in der Variablen r ab. r ist eine Instanz der Klasse **Response**, die im Modul **requests** definiert ist. Den Übertragungsstatus in **reason** und den Text der Antwort geben wir im Terminal und im Display aus. Danach schließen wir die Verbindung ordnungsgemäß. Täten wir das nicht, bekämen wir beim nächsten Durchlauf eine Fehlermeldung mit Programmabbruch.

Der Rest ist schnell erledigt. Fall die Taste gedrückt ist, wird die LED gezielt ausgeschaltet und das Programm verlassen.

```
    if taste.value()==0:
        red.off()
        sys.exit()
```

Auf der Seite https://ThingSpeak.com/channels/XXXXXXX/private_show können Sie jetzt die Aufzeichnung der Messwerte verfolgen. Natürlich müssen Sie die X-e durch Ihre Kanalnummer ersetzen.

In der nächsten Folge geht es um einen Telegram-Bot. Damit werden wir in der Lage sein, nicht nur Daten zu empfangen, sondern auch Schaltbefehle zu erteilen.

Bis dann!