Wetterticker - 182 x 8 - Matrix

Dieser Beitrag ist auch als PDF-Dokument verfügbar.

Der HTU21 kann Temperatur und Luftfeuchte messen. Für den Luftdruck bräuchten wir zum Beispiel zusätzlich einen BMP280, oder gleich einen BME280 für alles zusammen. Da erhebt sich natürlich die Frage, wie es richtig heißen muss: Lass mir messen oder lass mich messen? In diesem Fall ist beides falsch, denn heute heißt es richtig, lass andere messen.

Diese "anderen" betreiben eine Website mit der URL openweathermap.org. Von dort kann man Wetterdaten aus der ganzen Welt bekommen. Voraussetzung ist nur eine kostenlose Anmeldung. Dann bekommt man einen 32-Zeichen langen Key. Damit werden die Anfragen gefüttert, welche darüber hinaus auch noch den Ort, die Landeskennung und die Sprache enthalten müssen. Wie das alles funktioniert und welche Rolle das Mammut-Matrix-Display dabei spielt, das erfahren Sie in diesem Blogpost aus der Reihe

MicroPython auf dem ESP32 und ESP8266

heute

Das Mammut-Matrix-Display als Wetterticker

Zum Mammut-Matrix-Display selbst und dem zugehörigen Controller erfahren Sie mehr im ersten Teil dieser Reihe. Dort finden Sie den Schaltplan und eine Beschreibung der Arbeitsweise des Displays. Ferner wurde ein MicroPython-Modul für die Ansteuerung des Displays entwickelt, das in weiteren Projekten Anwendung finden kann und wird, zum Beispiel hier.

Im zweiten Teil ging es um den Temperatur- und Feuchtesensor HTU21 alias SHT21. Für dessen Ansteuerung wurde ebenfalls ein MicroPython-Modul gebaut. Daneben erfahren Sie dort mehr über die Arbeitsweise des I2C-Busses und es wird gezeigt, wie man mit preiswerten Mitteln die Bussignale sichtbar machen kann. Der eingesetzte 8-Kanal-Logicanalyzer ist dafür bestens geeignet. Außerdem gibt es Informationen zur CRC-Prüfsummenberechnung.

Um die Funkübermittlung der Daten an das Display ging es im dritten Teil. Die Abbindung erfolgte via WLAN und zwar wahlweise über den Accesspoint des Routers oder über den Accesspoint des ESP32, der auch das Display ansteuert.

Außerdem können Sie hier nachlesen, wie man aus Windows-TTF-Zeichensätzen Pixelzeichensätze für OLED-Displays und natürlich auch für Mammut-Matrix-Display clonen kann.

An Hardware verwende ich in dieser Episode nur einen ESP32(S) und das Matrix-Display. Natürlich kann auch ein HTU21 zusätzlich die hauseigenen Klimadaten liefern.

Die anzuzeigenden Texte sind alle deutlich länger als 16 Zeichen, deshalb wird die Methode **roll**() verwendet. Das Display sollte wegen der Lesbarkeit wenigstens aus 8 bis 12 Elementen bestehen.

Hardware

1	ESP32 Dev Kit C unverlötet oder ESP32 NodeMCU Module WLAN WiFi Development Board oder NodeMCU-ESP-32S-Kit
nach Bedarf	MAX7219 8x8 1 Dot Matrix MCU LED Anzeigemodul oder MAX7219 8x32 4 in 1 Dot Matrix LED Anzeigemodul
1	LM2596S DC-DC Netzteil Adapter Step down Modul
diverse	Jumperkabel
1	Minibreadboard oder Breadboard Kit - 3 x 65Stk. Jumper Wire Kabel M2M und 3 x Mini Breadboard 400 Pins
1	Sperrholzstreifen 5x 25cm … zur Display-Montage

Die Schaltung ist sehr einfach und daher gut für Anfänger geeignet. Im Schaltbild ist optional ein HTU21 alias SHT21 mit eingezeichnet. Über dessen Verwendung erfahren Sie im 2.Teil alles Notwendige.

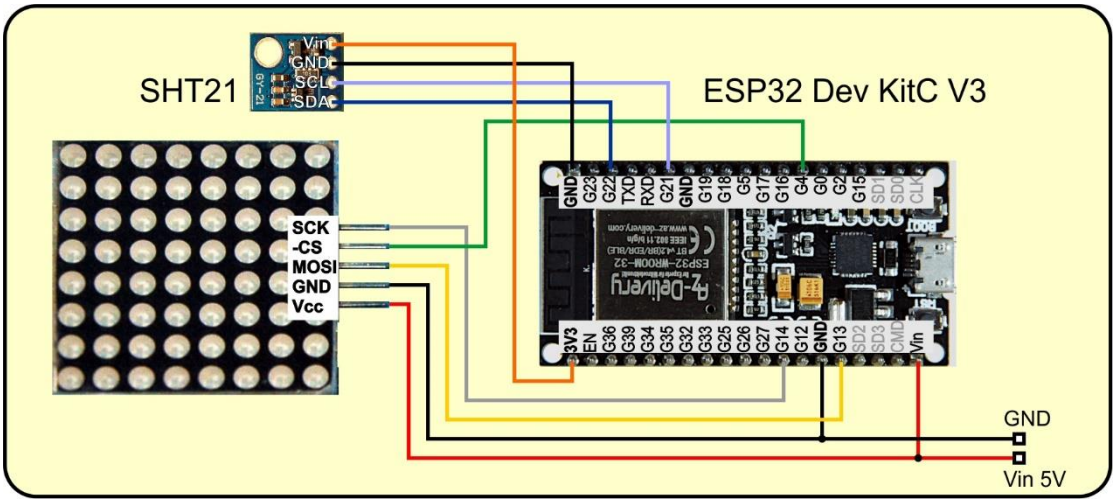

Abbildung 1: SHT21-Thermo+Hyrdometer mit ESP32

Als 5V-Versorgung für mehrere 8x8-Elemente sollte eine Spannungsversorgung mit ausreichend Stromstärkereserven verwendet werden. Ich verwende dafür ein Modul mit Step-Down-Converter, das bis zu 2A (3A kurzzeitig) liefern kann. Bei variabler Eingangsspannung von 6 bis ca. 20V wird der Ausgang auf 5V mit dem Trimm-Poti eingestellt.

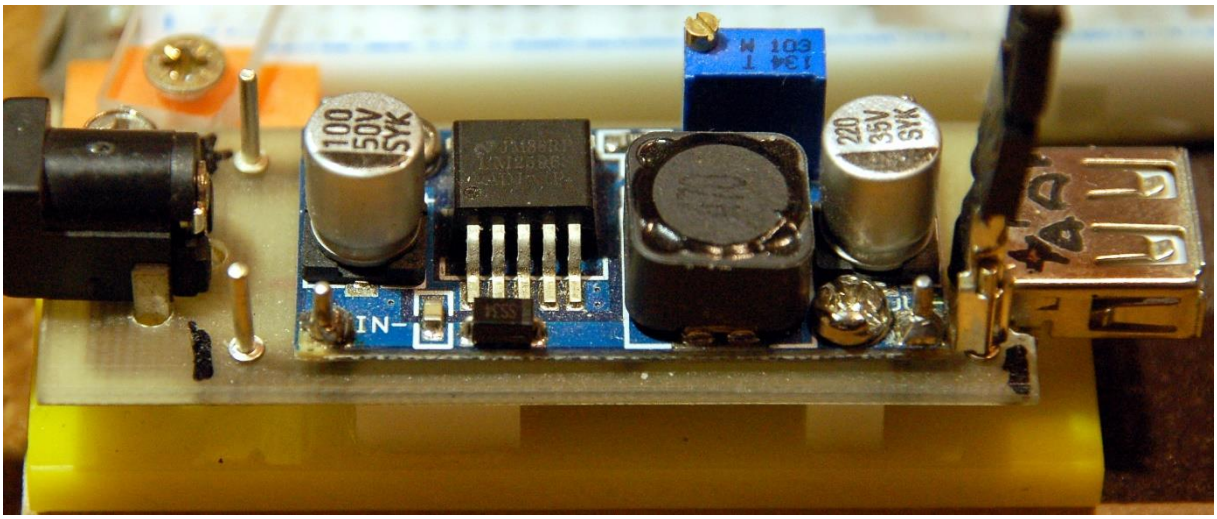

Abbildung 2: Spannungsversorgung mit Buck-Konvertrer auf meiner Trägerplatine

Für die Entwicklung von Projekten habe ich mir dazu eine Basisplatine entworfen, mit Rohrbuchse am Eingang und USB-A-Buchse am Ausgang versehen und das Ganze auf einer Plexiglas-Grundplatte montiert.

Ein Account bei Open Weather

Für einen Account bei Open Weather gehen Sie auf deren Webseite [How to Start].

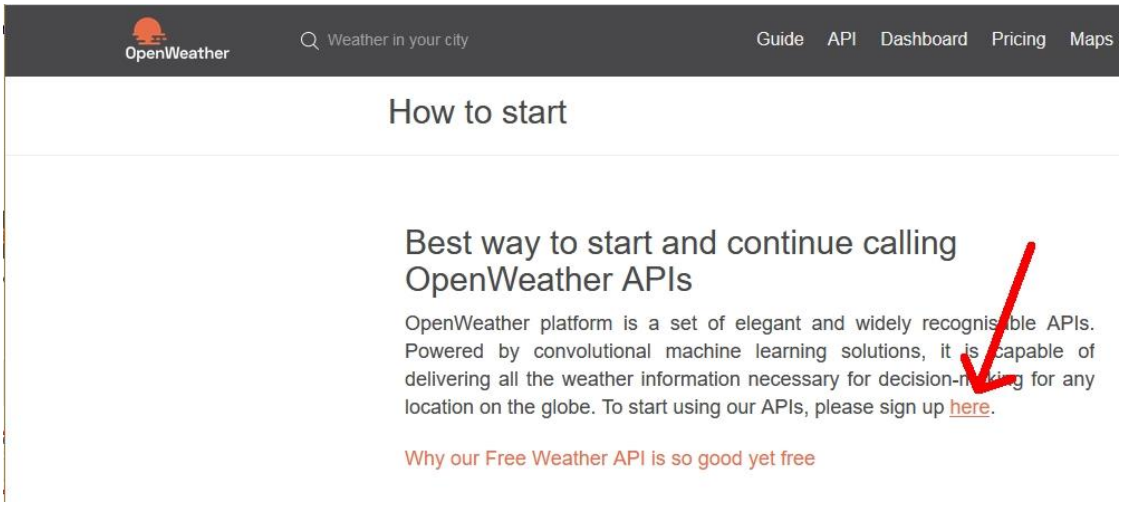

Abbildung 3: How to start

Klicken Sie auf **here** (roter Pfeil) und geben Sie im nachfolgenden Formular Ihre Daten ein.

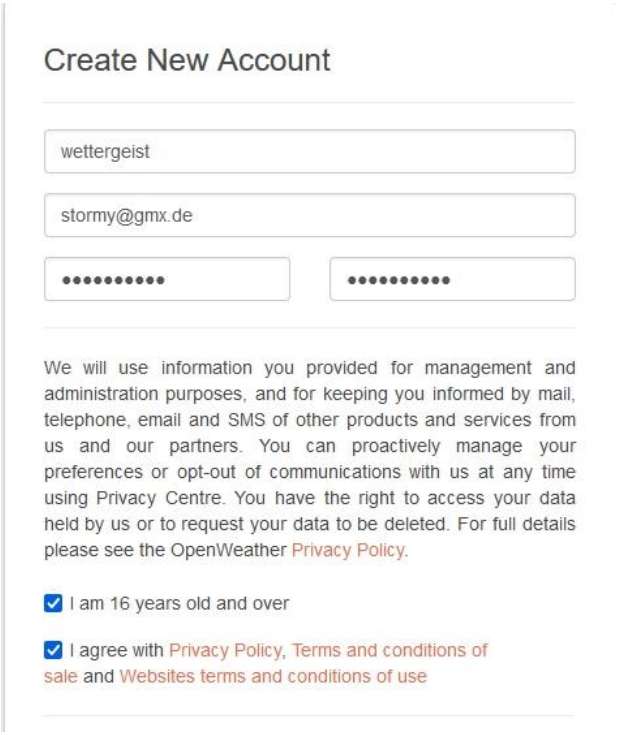

Abbildung 4: create new account

Erklären sie, dass Sie kein Roboter sind und schicken Sie die Anmeldung ab.

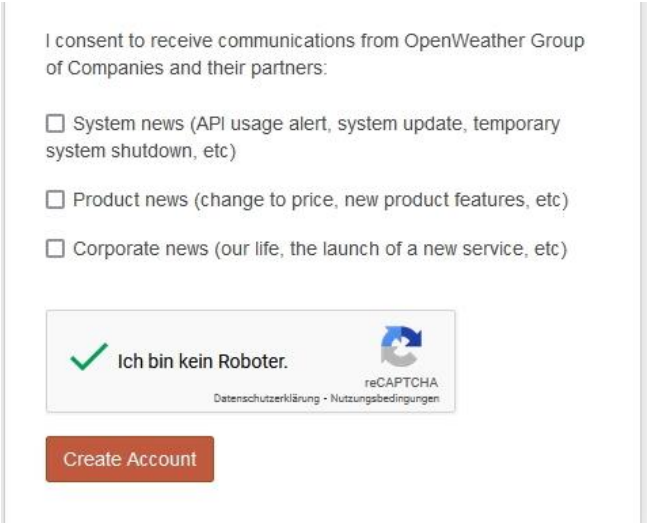

Abbildung 5: create account

Man will wissen, wofür Sie Open Weather nutzen möchten. Wenn Sie sich nicht festlegen wollen, wählen Sie **other**. Zum Abschluss klicken Sie auf **Save**.

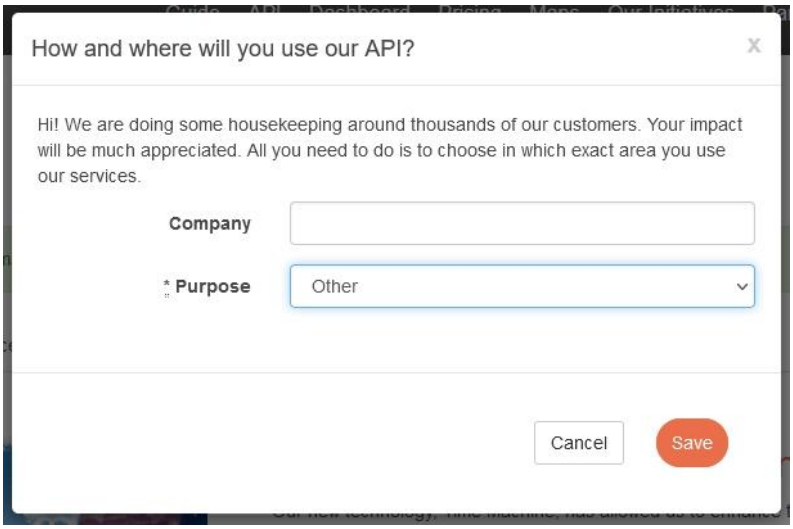

Abbildung 6: How to use

Danach gibt es eine Begrüßungsmeldung. stormy@gmx.de ist eine Fake-Adresse. Anfragen an diesen Mail-Account werden sicher nicht beantwortet. Wenn Sie den Confirmation Link quittieren wollen, sollten Sie natürlich Ihre Mailadresse angegeben haben.

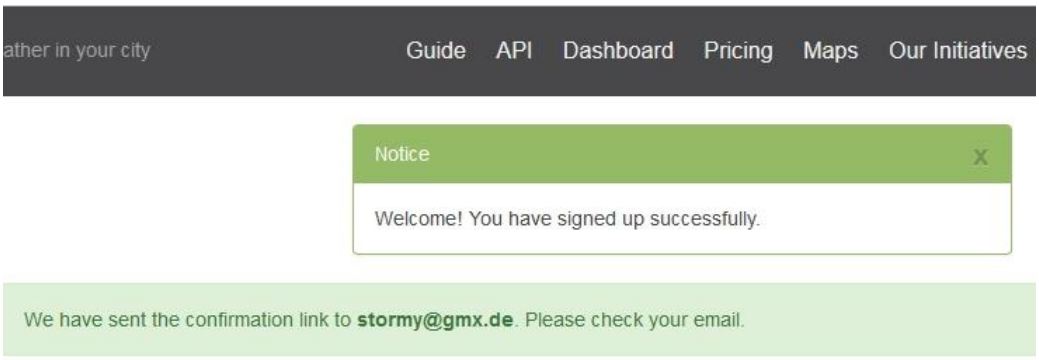

Abbildung 7: confirmation mail

Ein Klick auf **API keys** zeigt Ihnen Ihre APPID. Kopieren Sie diese an einen sicheren Ort und halten Sie den Code bereit, wir brauchen ihn in Kürze.

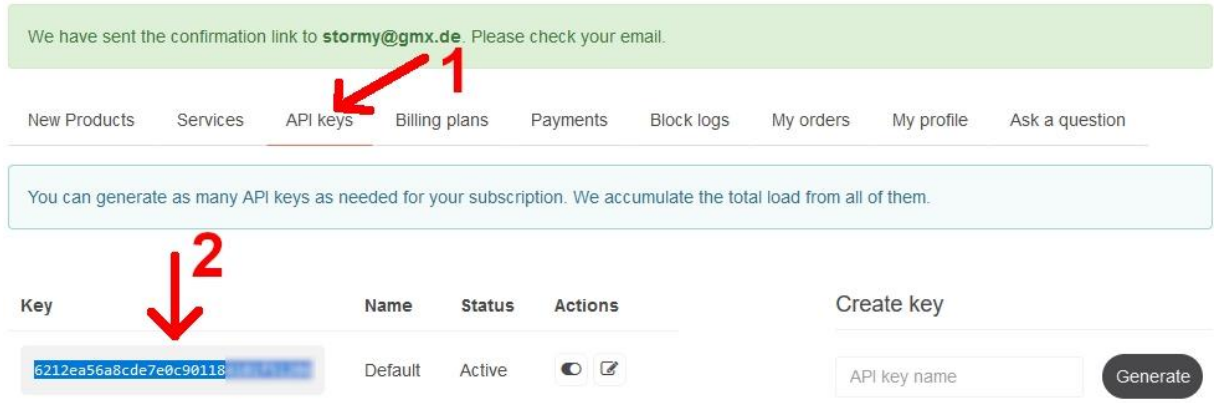

Abbildung 8: API-Key kopieren und sichern

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft

Verwendete Firmware für den ESP32:

MicropythonFirmware
Bitte eine Stable-Version aussuchen
ESP32 mit 4MB Version 1.18 Stand 25.03.2022

Die MicroPython-Programme zum Projekt:

matrix8x8.py Treibermodul für den MAX7219
evtl. sht21.py Treibermodul für das GY-21-Modul
evtl. shtdisplay.py Thermo- Hygrometer Software mit Writer
evtl. shtdisplay+.py Thermo- Hygrometer Software mit Writer
writer.py Der Treiber für die Anzeige mit den neuen Zeichensätzen.
ocr8.py schmaler Zeichensatz
openweathermap.py Dienstsoftware zum Blog

Sonstige Software:

Browser

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 MicropythonFirmware (Stand 05.02.2022) 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.

Die Verbindung zu Open Weather

Bevor wir mit dem Programm starten, testen wir gleich einmal die Verbindung zu Open Weather. Haben Sie Ihre APPID griffbereit? Ja, dann Starten Sie bitte einen Browser, wenn nicht eh schon einer läuft und geben Sie folgende URL ein. Bevor sie die Anfrage abschicken, müssen Sie die xxxxx.. durch Ihre APPID ersetzen.

http://api.openweathermap.org/data/2.5/weather?q=Berlin,DE&APPID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&lang=de

Und ab die Post. Das Ergebnis sollte so ähnlich aussehen, wenn Sie den Reiter JSON gewählt haben.

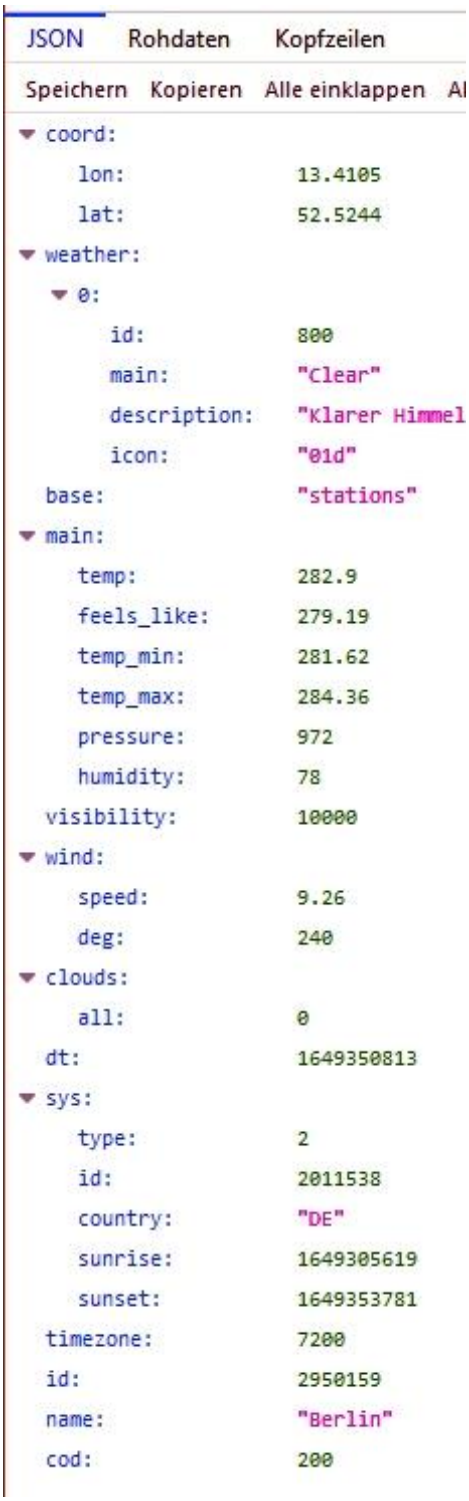

Abbildung 9: Wetter Berlin

Wenn Sie auf Rohdaten klicken schaut das so aus.

Abbildung 10: Rohdaten


```
{"coord":{"lon":13.4105,"lat":52.5244},"weather":[{"id":800,"main":"Clear",
"description":"Klarer Himmel","icon":"01d"}],"base":"stations","main":{"tem
p":282.9,"feels_like":279.19,"temp_min":281.62,"temp_max":284.36,"pressure"
:972,"humidity":78},"visibility":10000,"wind":{"speed":9.26,"deg":240},"clo
uds":{"all":0},"dt":1649350813,"sys":{"type":2,"id":2011538,"country":"DE",
"sunrise":1649305619,"sunset":1649353781},"timezone":7200,"id":2950159,"nam
e":"Berlin","cod":200}
```


Was Sie vor sich haben ist **JSON**-Code. JSON ist das Akronym für **JavaScript Object Notation**. Neben dem Klartext in Form von Strings und Zahlen finden Sie in dem Text eine Reihe von Symbolen wie { }, [ ], :, " " und Kommas. Strings sind in doppelte Hochkommas eingefasst wie in MicroPython. Daten sind als Paare mit einem Namen und einem Wert, getrennt durch einen Doppelpunkt angegeben wie in einem Dict in MicroPython. Geschweifte Klammern kennzeichnen Dict-Objekte und eckige Klammern definieren Listen.

Die übersichtliche JSON-Notation im Browser nutzen wir jetzt, um die Daten aus der Antwort von Open Weather für unsere Anzeige aufzubereiten. Weitere Informationen zu den einzelnen Feldern finden übrigens unter dieser URL.

https://openweathermap.org/current

Wir starten im Programm mit diversen Importen. Programmzeilen, die sich auf den HTU21 beziehen kommentiere ich aus. Importzeilen, die sich speziell auf das Projekt beziehen, habe ich fett formatiert.

```
# openweathermap.py
import os,sys       # System- und Dateianweisungen
from machine import Pin,SoftI2C, SPI
from time import sleep, sleep_ms, ticks_ms
#from sht21 import SHT21
from matrix8x8 import MATRIX
import ocr8 as charset
from writer import Writer

#from i2cbus import I2CBus

import esp
esp.osdebug(None)
import gc
gc.collect()
```

Es folgen Importe zur Netzwerkanbindung und das Arbeiten mit Serververbindungen.

```
import network
import socket
try:
  import urequests as requests
except:
  import requests
try:
  import ujson as json
except:
  import json
```

Die Klasse **network** enthält die Methoden zur Anbindung ans lokale Netzwerk, **socket** stellt Methoden für die TCP-Verbindung zum Open Weather Server zur Verfügung. Mit **requests** bauen wir die Verbindung auf und empfangen die Antwort. **json** bietet Methoden zum bequemen Zerpflücken der Antwort, wie wir sie oben gesehen haben. Ohne dieses Modul müssten wir einen eigenen Parser schreiben, was weitaus aufwendiger wäre.

Die folgenden vier Zeilen erklären sich selbst. Natürlich müssen Sie hier Ihre eigenen Zugangsdaten eintragen und die IP-Adresse für den ESP32 an Ihre Netzwerk-Infrastruktur anpassen.

```
mySSID = "Here goes your SSID"
myPass = 'Here goes your password'
myIP="10.0.1.96" # unsere STA-IP
myPort=9009 # unsere Portnummer
```

Die Flash-Taste des ESP32-Moduls erklären ich als Notbremse. Ich benutze sie, wenn sonst nichts mehr hilft, um aus der Mainloop auszusteigen, vor allem, wenn das Programm im Produktionssystem als **boot**.py autonom läuft. Wie das mit dem Autostart läuft, ist im Kapitel Software erklärt.

```
taste=Pin(0,Pin.IN,Pin.PULL_UP)
city = 'Nuremberg'
country_code = 'DE'
api_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'

wr=["N","NNO","NO","ONO",
    "O","OSO","SO","SSO",
    "S","SSW","SW","WSW",
    "W","WNW","NW","NNW",
    ]

messDelay=5*60*1000 # 5 Minuten
```

Wir definieren einen Ortsnamen, einen Ländercode und legen die APPID fest. Für die x-e setzen Sie bitte wieder Ihre APPID ein. Die Liste der Windrichtungen hilft uns, die

Richtung statt in Grad, mit Hilfe der Richtungsakronyme auszugeben. Und es reicht, wenn wir alle 5 Minuten eine Abfrage beim Server starten.

Dann bestimmen wir den Typ des verwendeten Controllers und stellen danach die GPIO-Pins für den SPI-Bus ein, den wir zu Ausgabe ans Display brauchen.

```
chip=sys.platform
if chip == 'esp8266':
    # Pintranslator fuer ESP8266-Boards
    # LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
    # ESP8266 Pins 16  5  4  0  2 14 12 13 15
    #                 SC SD
    bus = 1
    MISOp = Pin(12)
    MOSIp = Pin(13)
    SCKp  = Pin(14)
    spi=SPI(1,baudrate=4000000)   #ESP8266
    # # alternativ virtuell mit bitbanging
    # spi=SPI(-1,baudrate=4000000,sck=SCK,mosi=MOSI,\
    #         miso=MISO,polarity=0,phase=0)  #ESP8266
    CSp = Pin(16, mode=Pin.OUT, value=1)
#     SCL=Pin(5) # S01: 0
#     SDA=Pin(4) # S01: 2
#     i2c=SoftI2C(SCL,SDA)
elif chip == 'esp32':
    bus = 1
    MISOp= Pin(15)
    MOSIp= Pin(13)
    SCKp = Pin(14)
    spi=SPI(1,baudrate=10000000,sck=Pin(14),mosi=Pin(13),\
            miso=Pin(15),polarity=0,phase=0)  # ESP32
    CSp = Pin(4, mode=Pin.OUT, value=1)
#     SCL=Pin(21)
#     SDA=Pin(22)
#     i2c=SoftI2C(SCL,SDA)
else:
    raise OSError ("Unbekannter Port")
```

```
numOfDisplays=16
d=MATRIX(spi,CSp,numOfDisplays)
d.setIntensity(7)
w = Writer(d, charset)
```

Ich teile dem Programm mit, dass ich 16 Display-Elemente benutze und instanziiere mit dem SPI-Bus-Objekt, dem GPIO-Pin für die Chipselect-Leitung und der Anzahl von Einheiten das Mammut-Matrix-Display-Objekt d. Das Writer-Objekt w dient der Darstellung eigener Zeichensätze. Wie diese aus Windows-Zeichensätzen abgeleitet und verwendet werden, das ist in Teil 3 dieser Reihe erklärt.

Wir definieren zwei Funktionen. **hexMac**() nimmt die Bytefolge von **nic.config**('mac') und macht daraus einen normalen String für die Rückgabe. Diese Information wird gebraucht, um im Router den Zugriff für unsere Client-Station freizuschalten.
Nehmen Sie für diese Einstellung bitte Rücksprache mit dem Handbuch Ihres WLAN-Routers.

```
def hexMac(byteMac):
    macString =""
    for i in range(0,len(byteMac)):
        macString += hex(byteMac[i])[2:]
        if i <len(byteMac)-1 :
            macString +="-"
    return macString+" "
```

TimeOut() ist eine Funktion, die in ihrem Funktionskörper eine weitere Funktion definiert. Diese Funktion **compare**() nutzt den Wert des Übergabeparameters t und die Variable **start**, die außerhalb von compare() declariert ist, aber innerhalb der Funktion benutzt wird. TimeOut() gibt eine Referenz auf compare() zurück, nicht etwa einen Wert. compare() ist eine sogenannte Closure und das hat den Vorteil, dass Variablen zwischen den Aufrufen vom compare() ihren Wert behalten, während lokale Variablen normalerweise beim Verlassen der Funktion eingestampft werden. Ich benutze die Closure gerne als nicht blockierenden Softwaretimer.

```
def TimeOut(t):
    start=ticks_ms()
    def compare():
        return int(ticks_ms()-start) >= t
    return compare
```

Das Dictionary connectStatus erlaubt die Ausgabe des WLAN-Verbindungsstaus im Klartext.

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

Das AP-Interface wird nicht gebraucht, deshalb schalten wir es vorsichtshalber ab, bevor es Unfug anrichtet. Das kommt bisweilen vor.

```
nac=network.WLAN(network.AP_IF)
nac.active(False)
nac=None
```

Das Station Interface wird instanziiert und erst einmal deaktiviert. Dann holen wir die MAC und geben sie aus.

```
nic = network.WLAN(network.STA_IF)
nic.active(False)

MAC = nic.config('mac')
myID=hexMac(MAC)
print("Client-ID",myID)
```

Nun erfolgt die Aktivierung des STA-Interfaces, wir setzen eine statische IP-Adresse. Das deaktiviert gleichzeitig den DHCP-Client, der die IP-Adresse vom WLAN-Router beziehen würde. Bei einem Client ist eine statische IP nicht unbedingt erforderlich, erleichtert aber die Fehlersuche, wenn irgendetwas nicht wunschgemäß läuft.

Weiter oben haben Sie Ihre Credentials angegeben, mit denen jetzt eine Verbindung zum Router aufgebaut werden soll.

```
# Wir aktivieren das Netzwerk-Interface
nic.active(True)

# Aufbau der Verbindung
# Wir setzen eine statische IP-Adresse
nic.ifconfig(("10.0.1.96","255.255.255.0","10.0.1.20","10.0.1.
100"))

# Anmelden am WLAN-Router
nic.connect(mySSID, myPass)
```

Normalerweise besteht jetzt noch keine Verbindung zum Accesspoint. Wir sind aber vorsichtige Menschen und deshalb fragen wir den Zustand erst einmal ab, nachdem der Laufindex n mit 1 initialisiert wurde.

```
n=1
if not nic.isconnected():
    # warten bis die Verbindung zum Accesspoint steht
    while not nic.isconnected():
        print("{}.".format(nic.status()),end='')
        d.text("."*n,0,0,1)
        d.show()
        n+=1
        sleep(1)
```

Im Normalfall landen wir also in der **while**-Schleife, die so lange durchlaufen wird, bis die Verbindung steht. Solang das nicht der Fall ist, wird im Sekundenabstand ein Punkt mehr an das Display geschickt. Üblich sind 2 bis 3 Punkte. Bei mehr als zehn Punkten können Sie davon ausgehen, dass ein Problem vorliegt. Im Terminal wird der Verbindungsstatus als Zahl ausgegeben. Das Dict **connectStatus** verrät Ihnen die Ursache.

Den Laufindex kann man auch dazu verwenden, nach einer gewissen Zeit den Kontaktversuch zu diesem Router abzubrechen und entweder die Fehlermeldung im Klartext anzuzeigen oder die Verbindung zu einem alternativen Accesspoint zu versuchen.

```
n=1
if not nic.isconnected() and n<=10:
    # warten bis die Verbindung zum Accesspoint steht
    while not nic.isconnected():
        print("{}.".format(nic.status()),end='')
        d.text("."*n,0,0,1)
        d.show()
        n+=1
        sleep(1)
```

Ich verfolge den ersten Ansatz.

```
print("\nVerbindungsstatus: ",connectStatus[nic.status()])
if nic.isconnected():
    # War die Konfiguration erfolgreich? Kontrolle
    STAconf = nic.ifconfig()
    print("STA-IP:\t\t",STAconf[0],"\nSTA-NETMASK:\t",\
          STAconf[1],"\nSTA-GATEWAY:\t",STAconf[2] ,sep='')
else:
    print("No AP found")
```

Dann wird das Socket-Objekt **s** instanziiert. Die Parameter **socket.SOCK_STREAM** und **socket.AF_INET** bauen ein TCP-Interface auf der Basis der **IPv4-Adress**-Familie auf. Beim Aufruf der Methode **setsockopt**() sorgt der Parameter **socket.SO_REUSEADDR** dafür, dass dieselben Socketdaten, IP-Adresse und Portnummer, nach einem Neustart ohne Reset wiederverwendet werden können, ohne einen Fehler zu verursachen.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
s.bind(('', myPort))
print("I am on port",myPort)
s.settimeout(5)
```

Die Methode **bind**() bindet die Portnummer in **myPort** an die im WLAN-Teil eingestellte IP-Adresse. Das Timeout von 5 ms sorgt schließlich dafür, dass die Empfangsschleife des Sockets den Ablauf der Mainloop nicht blockiert. Wäre das der

Fall, dann könnte zum Beispiel das **roll**()-Kommando nicht vordergründig durchgeführt werden. Das Senden von Nachrichten an den Open Weather Server und der Empfang vom diesem sind also eher Nebensache und werden vom oben angesprochenen Timer geregelt. Dazu kommen wir gleich.

Die nächsten drei Zeilen setzen die URL für die API des Open Weather Servers, senden die Anfrage, holen im gleichen Zug die Antwort ein und starten den Timer für den nächsten derartigen Event.

```
owmurl =
'http://api.openweathermap.org/data/2.5/weather?zip=93047'+
',' + country_code + '&APPID=' + api_key + '&lang=de'
weather_data = requests.get(owmurl)
messen=TimeOut(messDelay)
```

An dieser Stelle ist eine Anmerkung zur Ortsangabe angebracht. Im Beispielcode wird eine der Postleitzahlen von Regensburg verwendet, zip=93047. Alternativ kann für größere Orte auch der Ortsname verwendet werden. Für Nürnberg würde dieselbe Stelle folgendermaßen lauten: q=Nuremberg. Sie können das austesten, wenn Sie den Inhalt von **owmurl** im Terminal ausgeben lassen und den String als URL im Browser eingeben. Anstatt der x-e sollte Ihre APPID stehen.

>>> owmurl = 'http://api.openweathermap.org/data/2.5/weather?zip=93047'+ ',' + country_code + '&APPID=' + api_key + '&lang=de'

>>> owmurl
'http://api.openweathermap.org/data/2.5/weather?zip=93047,DE&APPID=xxxxxxxxxxxxxxxxxxxxxxxxxxxx&lang=de'

>>> owmurl = 'http://api.openweathermap.org/data/2.5/weather?q=Nuremberg'+ ',' + country_code + '&APPID=' + api_key + '&lang=de'

>>> owmurl
'http://api.openweathermap.org/data/2.5/weather?zip=93047,DE&APPID=xxxxxxxxxxxxxxxxxxxxxxxxxxxx&lang=de'

Die Anfrage an den Server senden und dessen Antwort entgegennehmen steckt alles ganz kompakt in dieser Zeile.

weather_data = requests.get(owmurl)

requests.get() gibt ein reponse-Objekt zurück, das einige Datenfelder und die Methode **json**() enthält, die wir gleich zum Parsen benutzen werden.

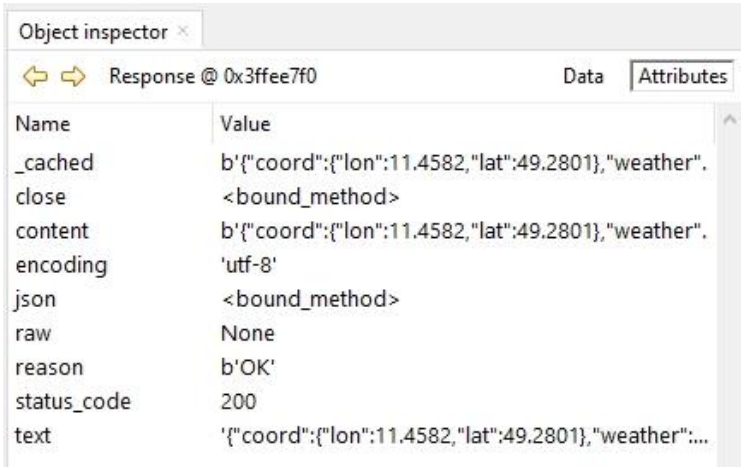

Abbildung 11: Innenleben des Response-Objekts

Ich mache grade noch den Timer fertig, dann geht es in die Mainloop. **messen** enthält eine Referenz auf die Funktion **compare**() im Inneren von **TimeOut**(). **messen** ist als Funktion also aufrufbar. Mit dem Aufruf von **messen**() rufe ich eigentlich **compare**() auf. Die Closure **compare**() gibt den Wert True zurück, wenn der Timer abgelaufen ist. Dann starten wir eine neue Anfrage und den Timer neu.

Weil wir neugierig sind, lassen wir uns den Text des response-Objekts ausgeben.

Die Reihenfolge bei der Decodierung des JSON-Codes ist beliebig und nur an dem Namen der Paare orientiert. So findet sich das Paar name:"Regensburg" ganz unten in der Reihe. **weather_data.json().get('name')** gibt den Wert "Regensburg" zurück.

Manche der Paare sind eingerückt, so zum Beispiel beim Namen **sys** (Abbildung 6). Schauen wir uns das beim Rohtext an, dann erkennen wir, dass dem Namen sys ein ganzes Dict zugewiesen ist.

'sys': {
'country': 'DE',
'sunrise': 1649392622,
'sunset': 1649440481,
'id': 1839, 'type': 1
}

Um auf die Länderkennung DE zuzugreifen, bedarf es einer gestuften Abfrage.

weather_data.json().get('sys').get('country')

Besonders interessant ist die Abfrage der Wetterbeschreibung. Dem Namen weather ist eine Liste zugewiesen (eckige Klammern), deren nulltes Element wir ansprechen müssen, um auf main und description zugreifen zu können.

```
weather_data.json().get('weather')[0].get('main')
weather_data.json().get('weather')[0].get('description')
```

Der zusammengesetzte Text wird am Terminal ausgegeben und zum Matrix-Display mit dem Befehl **roll**() geschickt.
Im Dict **main** sind die Daten für Temperatur, rel. Feuchte und Luftdruck untergebracht. Auch die sind also zweistufig abzufragen. Die Temperaturen in Kelvin müssen in Celsiuswerte umgerechnet werden. Die Gradweite bei der Kelvinskala ist dieselbe wie bei Celsius, lediglich 273,15 ist abzuziehen.

```
    temp = weather_data.json().get('main').get('temp')-273.15
    felt = weather_data.json().get('main').get('feels_like')-
273.15
    temperatur = 'Temperatur: {:5.1f} *C  gefuehlt: {:5.1f}
*C'.format(temp,felt)
```

Interessant sind die Formatstrings für die Ausgabe der Temperaturwerte. Da keine Namen für die Formatierung verwendet werden, gilt die automatische Zuordnung gemäß der Reihenfolge der Parameter. Die Ausgabe erfolgt als Fließkommazahl (f) mit mindestens 5 Zeichen Breite und einer Stelle nach dem Dezimalpunkt. Ähnlich sieht es beim Luftdruck und der relativen Luftfeuchte aus.

Beim Wind sind Richtung und Geschwindigkeit interessant. Die Geschwindigkeit wird als Ganzzahl in km/h umgerechnet. Die Richtung kommt in Grad von 0° (Nord) über 90° (Ost), 180° (Süd) und 270° (West). Der Index in die eingangs definierte Liste wr wird auf der Grundlage eines 22,5°-Rasters errechnet.

Ferner ist eine Warnung bei Böen (gust) von mehr als 40km/h sinnvoll. Das erledigt eine zentrierte blinkende Meldung am Schluss des while-Blocks, wenn die Geschwindigkeit der Böen 40 km/h übersteigt.

```
    speed=int(weather_data.json().get('wind').get('speed') *
3.6 + 0.5)
    direction=int(weather_data.json().get('wind').get('deg') /
22.5 + 0.5)
    gust=int(weather_data.json().get('wind').get('gust') * 3.6
+ 0.5)
    wind = 'Wind aus {} mit {:2} km/h '.format(wr[direction],
speed)
    print(wind)
    d.roll(wind)
    if gust >=40:
        w.center("SPITZEN: {:3} km/h".format(gust))
        d.blink(1500,500,6)
        d.pixelShift(8,"up",75)
```

Die Tastenabfrage schließt die while-Schleife ab.

```
    if taste.value()==0:
        d.clear()
        d.center("CANCELED")
        sys.exit()
```

Je nachdem welche API von Open Weather benutzt wird, können andere Dienste genutzt werden, sofern sie im Free Plan, also im freien, kostenlosen Zugang enthalten sind. Blättern Sie die Seite https://openweathermap.org/api durch, folgen Sie den Links, experimentieren Sie hier ruhig ein wenig. Das wäre zum Beispiel eine Anfrage für eine 7-Tage-Vorhersage.

https://api.openweathermap.org/data/2.5/onecall?lat=49.2801&lon=11.4582&exclude=hourly,minutely&appid= *xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*

Geben Sie die Zeile zusammen mit Ihrer APPID im Browser ein und sehen Sie sich erst einmal die Ausgabe im JSON-Ordner an, bevor Sie an das Zerpflücken der Meldung gehen. Die Anfrage nutzt die One-Call-API. Die Ortsangabe erfolgt hier in Breiten- und Längengraden. Wenn Sie die Werte für Ihren Ort nicht wissen, fragen einfach Google-Maps.

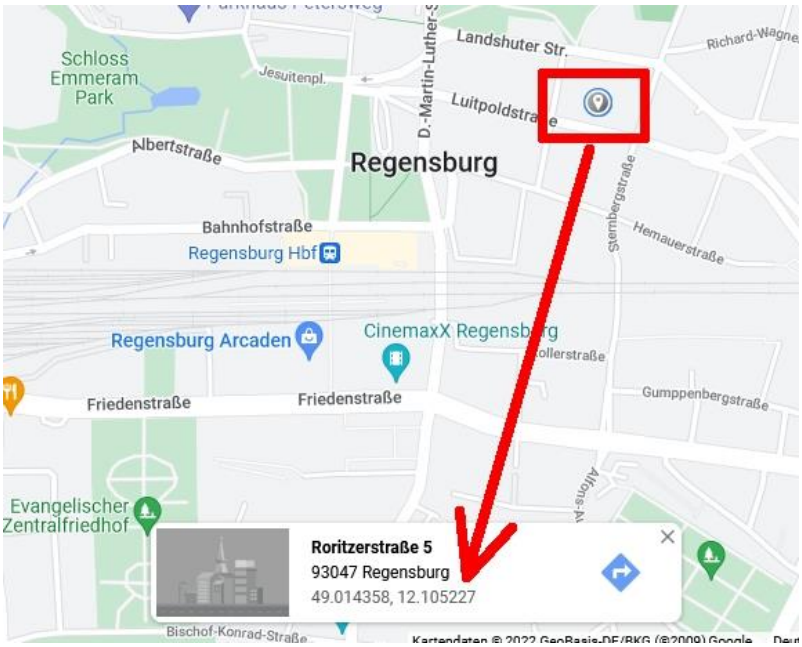

Abbildung 12: Ortskoordinaten Breite und Länge

In der Liste **daily** sind die Tage von 0 (heute) bis 7 (in einer Woche) durchnummeriert. Warnmeldungen stehen in der Liste **alerts**. Zu jeder API stehen Hilfeseiten zur Verfügung, in welchen die verschiedenen Parameter des Aufrufs und die Felder im Response-String beschrieben sind. Für die One-Call-API ist das diese Seite: https://openweathermap.org/api/one-call-api.

Alle Anfragen liefern auch verschiedene Zeiten, aktuelle Uhrzeit, Sonnenaufgang, Sonnenuntergang… Die Unix-Time-Stamps geben die Anzahl Sekunden seit dem 01.01.1970 0:0:0,0 wieder. Die Zeitepoche von MicroPython auf den ESP32-Ports beginnt aber erst am 01.01.2000. Damit die richtige Zeit-Datums-Kombi ausgegeben wird, braucht es zwei Korrekturen, 946681200 ist zu subtrahieren und während der Sommerzeit ist 3600 zu addieren. Dann kann mit **time.localtime**() die korrekte Zeit ausgegeben werden.

Das könnte dann mit der zuletzt angegebenen Anfrage so aussehen.

```
from time import localtime
...
    zeit=int(weather_data.json().get('current').get('dt') -
946681200 + 3600)
    dt= localtime(zeit)
    tag=str(dt[2])+"."+ str(dt[1])+"."+ str(dt[0])
    uhrZeit= str(dt[3])+":"+ str(dt[4])+":"+ str(dt[5])
    d.roll(tag+" *** "+uhrZeit)
```

Damit sind Sie für alle Eventualitäten gut ausgestattet und können die Dienste von Open Weather intensiv nutzen. Dazu wünsche ich viel Erfolg und Vergnügen!