

Wetterstation mit Raspberry Pi Pico

Der Bloggerkollege Andreas Wolter hat Ihnen in seinem [Beitrag](#) den Einstieg in den Raspberry Pi Pico gegeben, wie dieser beim ersten Mal benutzen eingerichtet werden muss und worauf es noch ankommt. In meinem heutigen Blogartikel greife ich mein Lieblingseinstiegsthema Wetterstation auf. Mit dem Raspberry Pi Pico, nachfolgend einfach nur noch Pico genannt, wollen wir uns die Temperatur und später auch die Luftfeuchtigkeit und den Umgebungsdruck ausgeben lassen.

Das Projekt wird, da der kleine MicroController ja ein Produkt der Raspberry Pi Foundation ist, auf dem Raspberry Pi 4 mit der Thonny Python IDE (also MicroPython) entwickelt. Damit Sie auch den Pico ein bisschen näher kennenlernen, wird das Projekt in diesem Blog in drei Teile unterteilt:

1. Temperaturausgabe vom intern verbauten Temperatursensor auf Konsole
2. Temperaturausgabe vom intern verbauten Temperatursensor auf i2c-OLED-Display
3. Umgebungsbedingungen mit BME/BMP280 ermitteln und auf i2c-OLED-Display ausgeben

Dies sind auch bei mir immer die ersten Gehversuche, wenn ich neue MicroController bekomme.

Benötigte Hard- und Software

Die Hardware für diesen Versuchsaufbau ist relativ einfach, siehe Tabelle 1.

Pos	Anzahl	Bauteil	Link
1	1	Raspberry Pi Pico	https://az-delivery.de
2	1	0,96 Zoll OLED SSD1306 Display I2C	https://az-delivery.de
3	1	Breadboard und Jump Wire	https://az-delivery.de
4	1	GY-BME280 Barometrischer Sensor	https://az-delivery.de

Tabelle 1: Hardwareteile

Bei der Software nutzen Sie, da es ein Programm mit Python3 wird, [Thonny Python IDE](#), welches bei dem Raspbian Image schon zur Verfügung steht, für alle gängigen Betriebssysteme installiert werden kann. Ich empfehle an dieser Stelle, sollten sie z.B. mit Ubuntu arbeiten, über die in Code 1 beschriebene Methode Thonny Python IDE zu installieren.

```
sudo pip3 install thonny
sudo usermod -a -G dialout $USER
```

Code 1: Install Thonny Python IDE via pip3

Danach können Sie die Thonny Python IDE zwar „nur“ noch über die Konsole starten, aber das ist so weit vertretbar.

Die Temperatur vom Pico auslesen

Im ersten Teilaufbau soll der intern verbaute Temperatursensor vom Pico ausgelesen werden. Sie werden sich jetzt wahrscheinlich wundern, da es beim ersten Betrachten nicht danach aussieht, dass der Pico einen Temperatursensor besitzt. Daher sollten Sie sich erst einmal das Pinout vom Pico ansehen, siehe Abbildung 1.

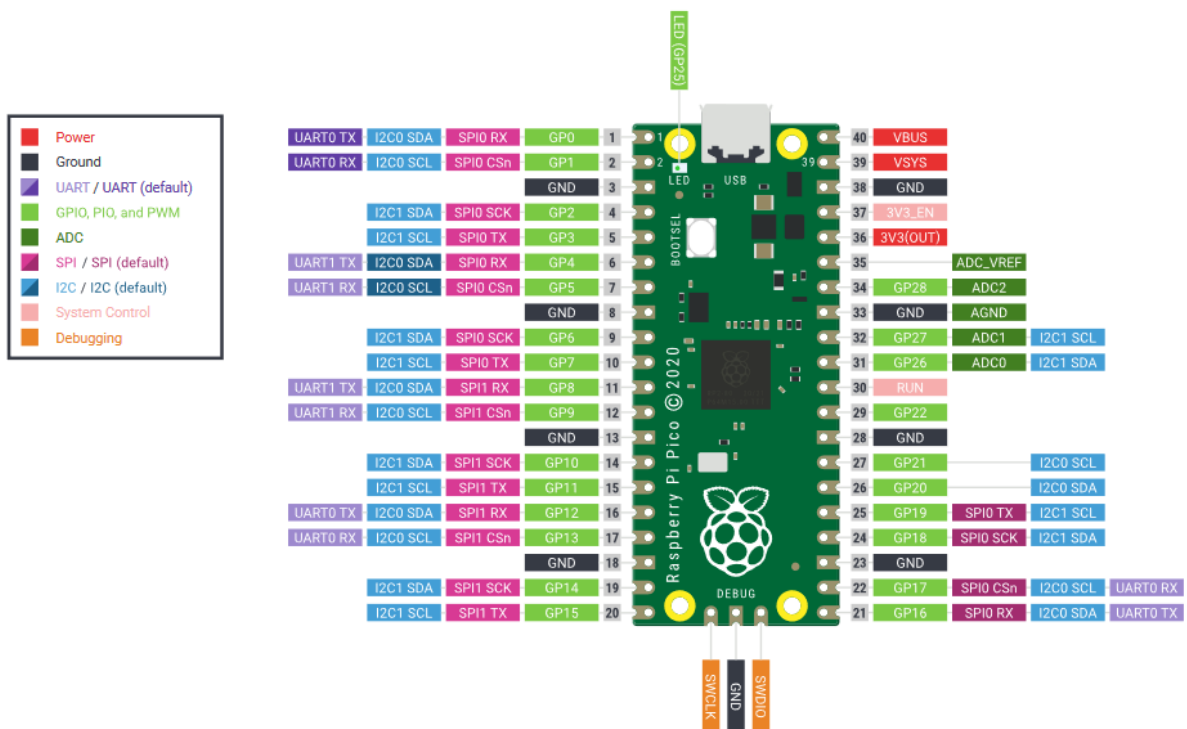


Abbildung 1: Pinout vom Raspberry Pi Pico, Quelle: <https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf>

Hier sehen Sie deutlich die Anschlüsse ADC0 – ADC2, wobei ADC_VREF hier auch als ADC3 betrachtet werden kann, die für analoge Signale genutzt werden können. Was Sie auf dem Pinout aus Abbildung 1 nicht sehen, aber im dem [RP2040 DataSheet](#) unter Kapitel 4.9.1 nachlesen können, ist ein fünfter analoger Pin (ADC4), der direkt an einen intern verbauten Temperatursensor angeschlossen ist, siehe Abbildung 2.

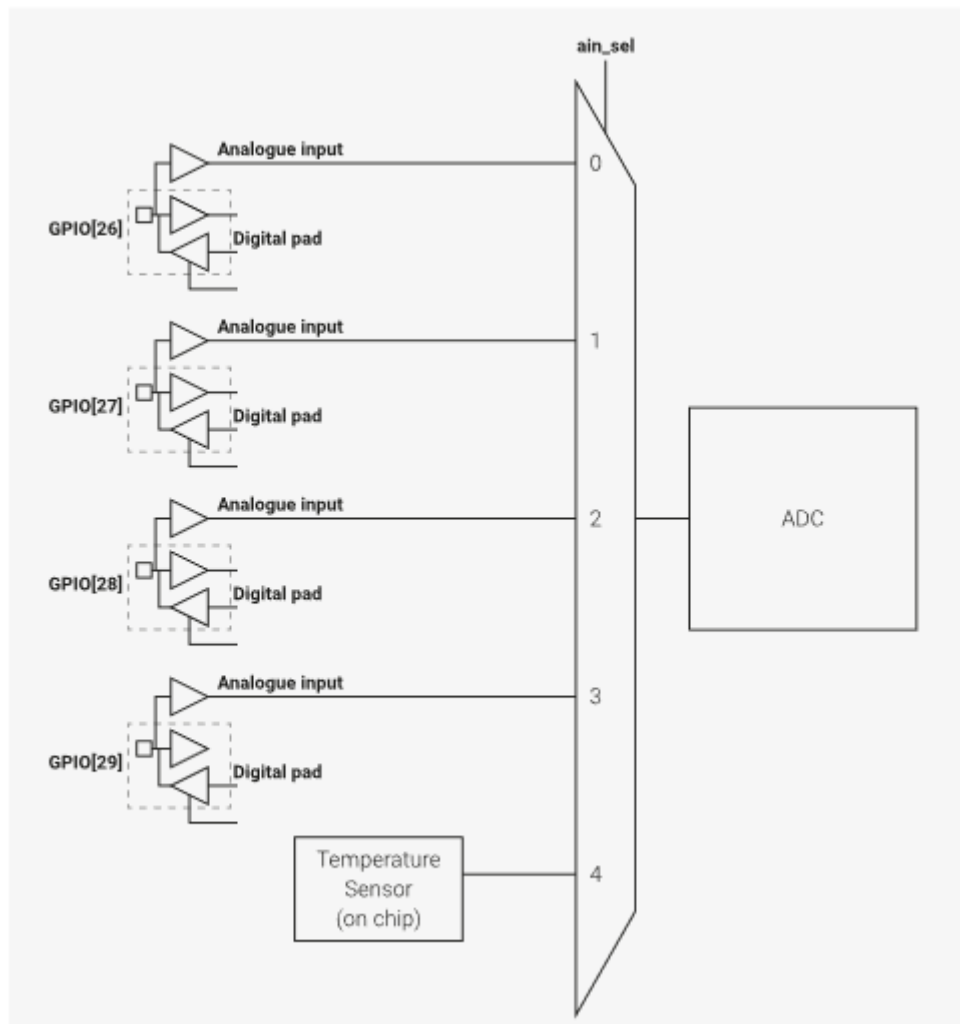


Abbildung 2: ADC Verbindungsdiagramm, Quelle: <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf>

Vom Grundprinzip haben wir damit die benötigte Hardware schon direkt auf dem Pico verbaut, um die Umgebungstemperatur zu ermitteln. Code 2 liest die Temperatur aus und gibt diese auf der Konsole aus. Das Programm ist recht einfach, wobei ich drei Stellen direkt erklären möchte.

```
"""
```

```
// Read internal temp-sensor
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 23. Oct 2021
// Update: 23. Oct 2021
"""
```

```
#First import needed libs
import machine #Important to get Pins and config them
import utime #For a little timer
```

```
sensor = machine.ADC(4) # Create object sensor and init as pin ADC(4)
conversion_factor = 3.3 / 65535 # 3.3V are 16bit (65535)
```

```
while True:
    valueTemp = sensor.read_u16() * conversion_factor
    temp = 27 - (valueTemp - 0.706) / 0.001721 # See 4.9.5 of rp2040 datasheet
```

```
print('Temperatur value: ' + "{:.2f}".format(temp)) # Print in terminal with two decimal
utime.sleep(2) #Sleep for two seconds
```

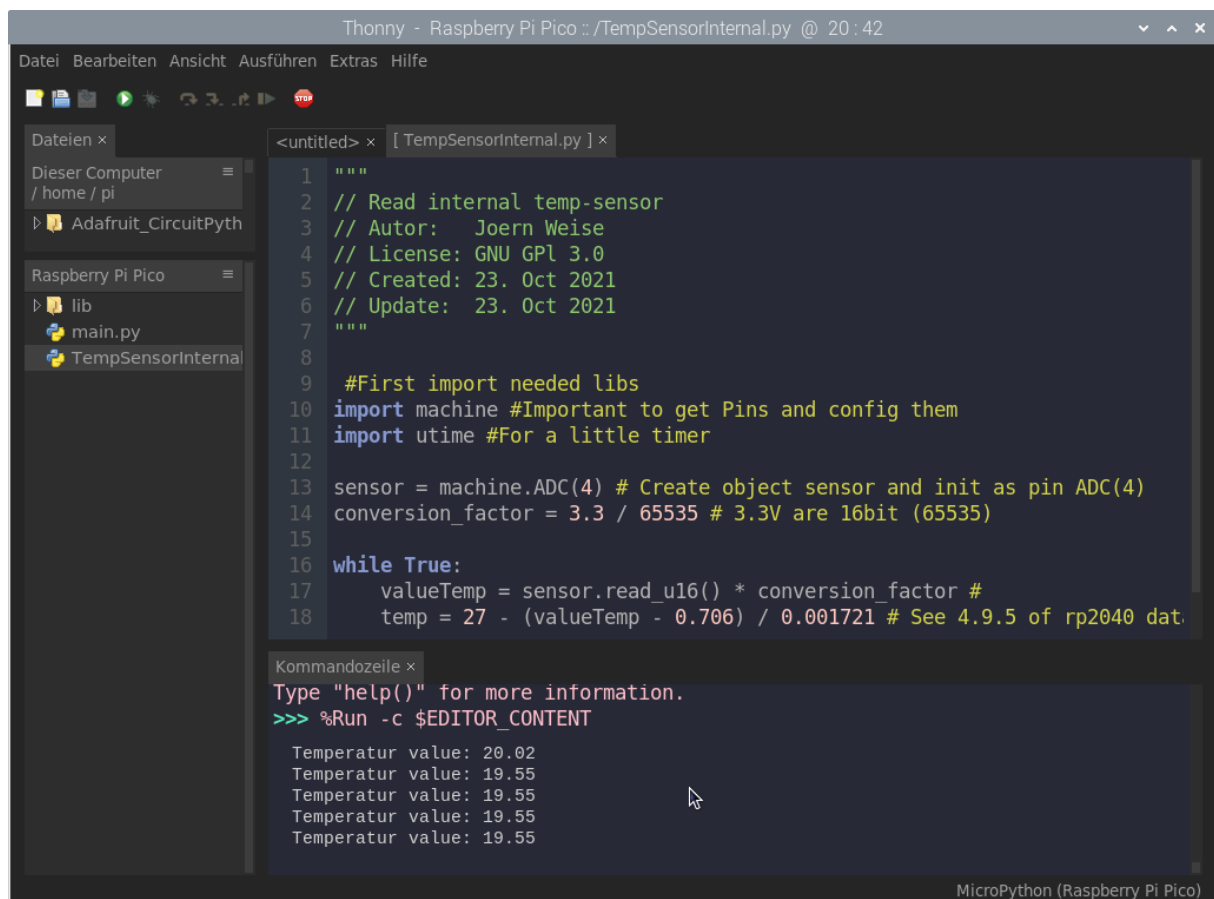
Code 2: Simples Programm um internen Temperatursensor auszulesen

Zunächst die Zeile „conversion_factor = 3.3 / 65535“, bei der ein Korrekturfaktor definiert wird. Diesen brauchen wir, da der Pico uns später einen analogen Wert zwischen 0 und 65535 bei der Sensorausgabe übergeben wird. Der Wert ist insoweit interessant, da es sich um die Größe 2 Byte bzw. 16bit handelt. Da 3,3 Volt dann 65535 entspricht, brauchen wir diesen, um die 2 Byte in eine Spannung umzurechnen.

Dann kommen wir zu der Zeile „valueTemp = sensor.read_u16() * conversion_factor“ bei der zunächst der Sensorwert ausgelesen und dann direkt mit dem Konvertierungsfaktor multipliziert wird. Interessant ist weniger die Multiplikation, mehr in diesem Fall das **read_u16()**, welches genutzt wird, um einen 16bit-Wert zu erhalten. Anders als **read()**, wo nur ein binärer Wert zurückgeliefert wird, weisen wir MicroPython mit **read_u16()** an, den Wert in 2 Byte auszugeben, was für einen analogen Pin durchaus Sinn macht.

Zuletzt die Zeile „temp = 27 - (valueTemp - 0.706) / 0.001721“, die ja schlussendlich die Temperatur vom Pico-Temperatursensor errechnen soll. Diese Formel habe ich mir nicht ausgedacht, sondern ist so in dem DataSheet vom Pico unter Kapitel 4.9.5 genauso wiederzufinden.

Die Ausgabe in der Konsole, mehr kann unser Programm aktuell noch nicht, ist auch recht unspektakulär, siehe Abbildung 3.



```
Thonny - Raspberry Pi Pico :: /TempSensorInternal.py @ 20:42
Datei Bearbeiten Ansicht Ausführen Extras Hilfe
Dateien x
Dieser Computer
/home / pi
  ↳ Adafruit_CircuitPyth
Raspberry Pi Pico
  ↳ lib
    ↳ main.py
    ↳ TempSensorInternal
<untitled> x [ TempSensorInternal.py ] x
1 """
2 // Read internal temp-sensor
3 // Autor: Joern Weise
4 // License: GNU GPL 3.0
5 // Created: 23. Oct 2021
6 // Update: 23. Oct 2021
7 """
8
9 #First import needed libs
10 import machine #Important to get Pins and config them
11 import utime #For a little timer
12
13 sensor = machine.ADC(4) # Create object sensor and init as pin ADC(4)
14 conversion_factor = 3.3 / 65535 # 3.3V are 16bit (65535)
15
16 while True:
17     valueTemp = sensor.read_u16() * conversion_factor #
18     temp = 27 - (valueTemp - 0.706) / 0.001721 # See 4.9.5 of rp2040 dat.
Kommandozeile x
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
Temperatur value: 20.02
Temperatur value: 19.55
Temperatur value: 19.55
Temperatur value: 19.55
Temperatur value: 19.55
MicroPython (Raspberry Pi Pico)
```

Abbildung 3: Ausgabe der Temperatur in der Konsole

Damit das bei Ihnen auch gelingt, reicht ein Druck auf **F5** oder aber den grünen **Run**-Button. Sie werden aber schnell feststellen, dass der intern verbaute Temperatursensor nicht sehr präzise ist. Das soll uns aber in diesem Beispiel erst einmal wenig stören.

Die Temperatur auf einem i2c-OLED anzeigen

Im nächsten Schritt soll die Ausgabe der Temperatur nicht mehr über die Kommandozeile stattfinden, sondern im Hinblick auf Nutzung ohne Verbindung zu einem PC, über ein OLED-Display. In meinem Fall das 0,96 Zoll OLED SSD1306 Display I2C mit 128x64 Pixeln. Das Großartige dabei ist, die Verbindung benötigt nur 4 Kabel, siehe Abbildung 4, eine Bibliothek und ein paar Zeilen mehr Code.

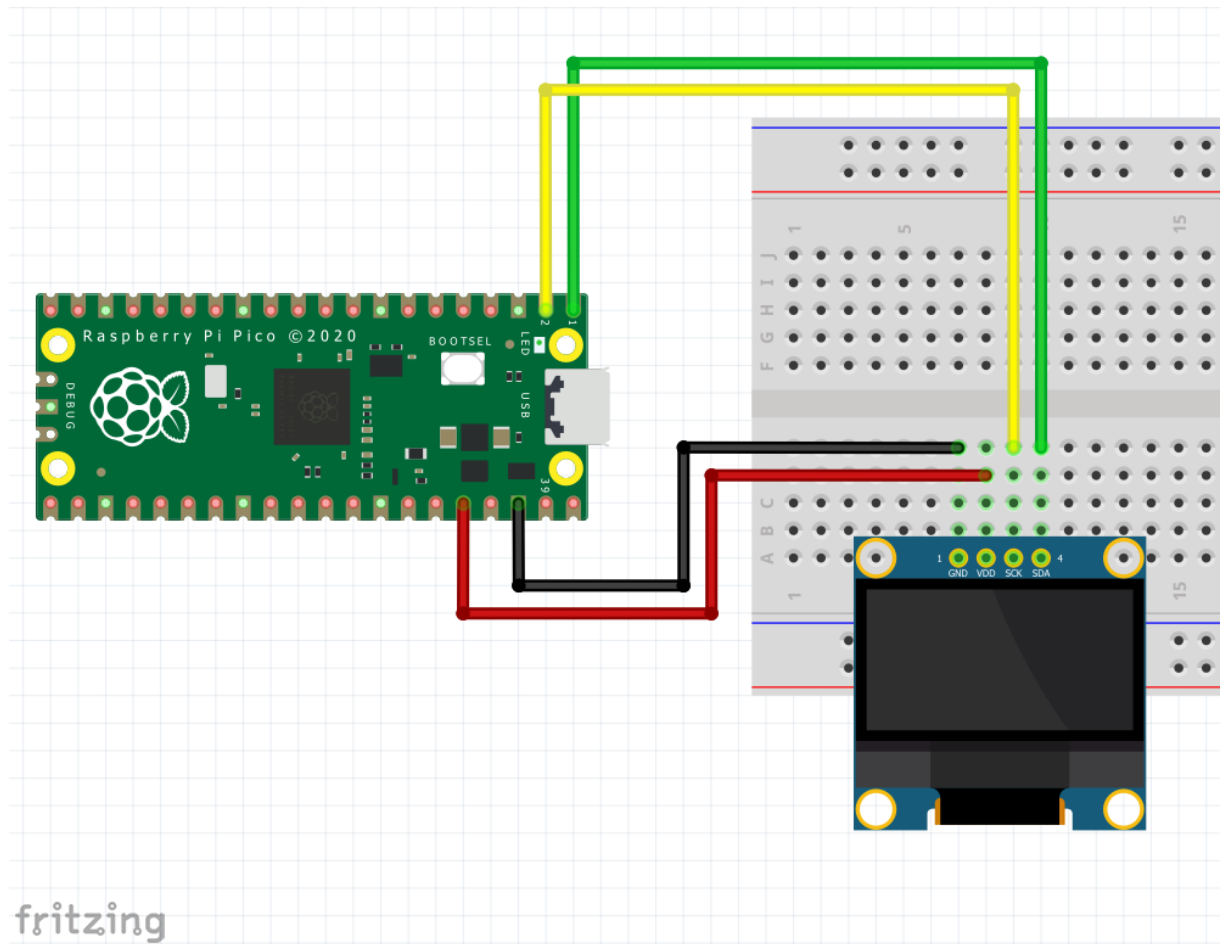


Abbildung 4: Verdrahtung Pico mit OLED-Display

Die Verdrahtung ist schon einmal recht Fix gemacht, daher soll es direkt in Thonny Python IDE mit der benötigten Bibliothek weitergehen. Öffnen Sie zunächst die Paketverwaltung, siehe Abbildung 5.

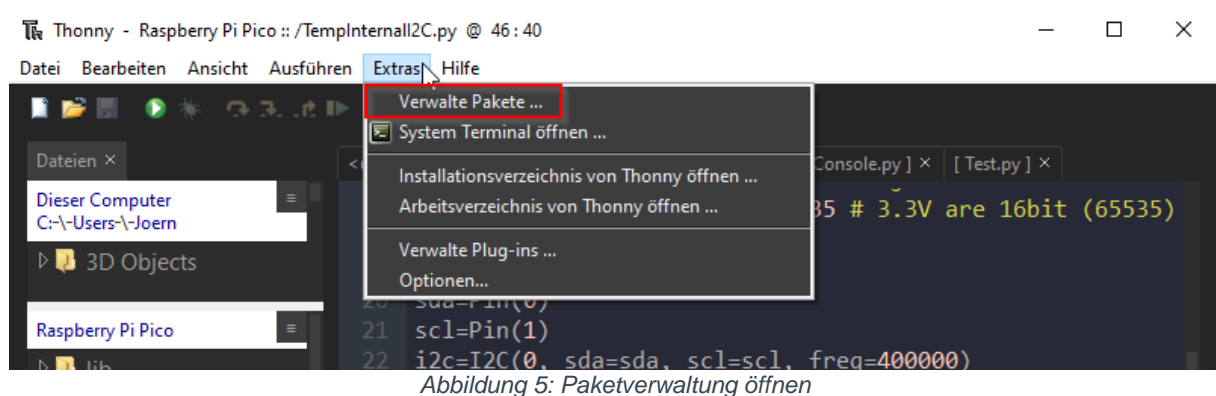


Abbildung 5: Paketverwaltung öffnen

Es öffnet sich danach ein neues Fenster, indem Sie nach Belieben nach Paketen, also Bibliotheken für weitere Hardware, suchen können. In unserem Fall benötigen wir eine MicroPython-Bibliothek für den SSD1306-Controller. Geben Sie dazu einfach in das

Suchfenster **ssd1306** ein und drücken Sie **Enter** oder auf **Suche auf PyPi** und wählen das Paket **micropython-ssd1306** aus, siehe Abbildung 6.

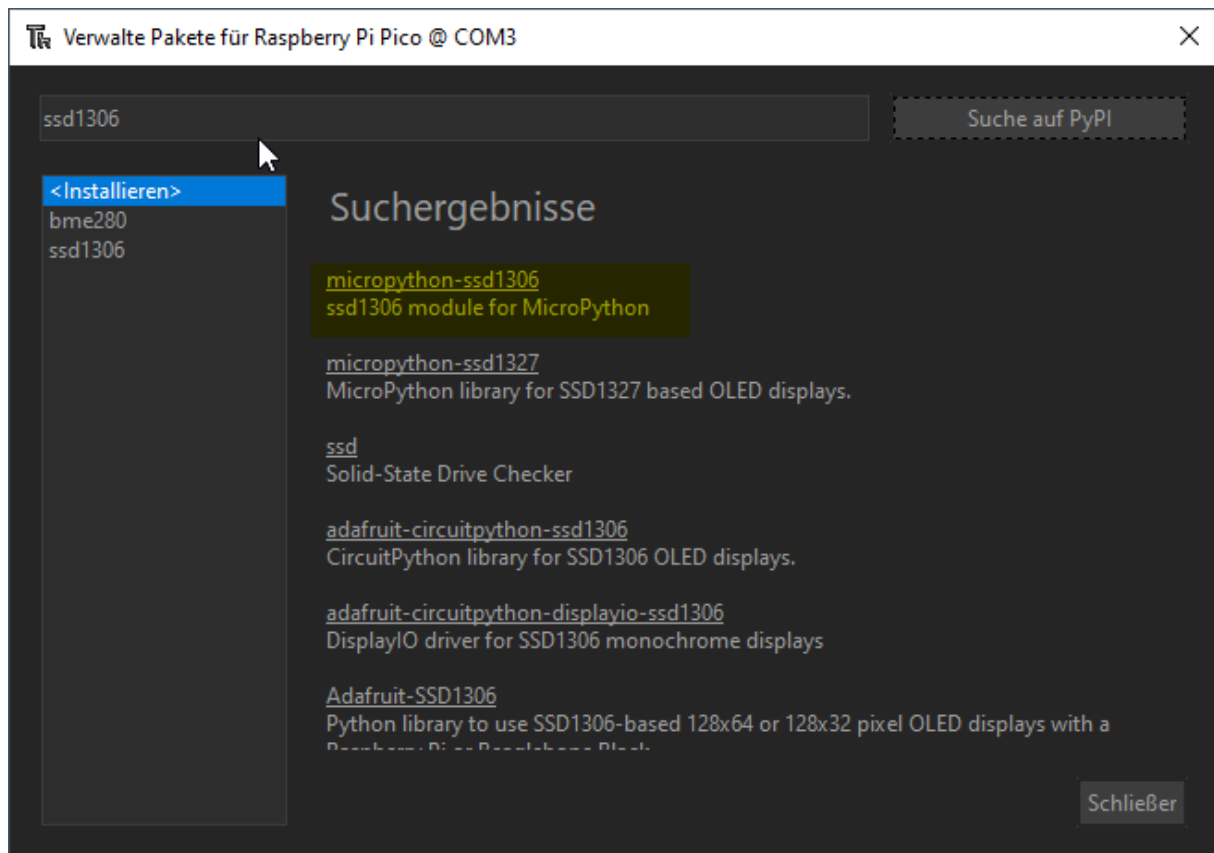


Abbildung 6: Nach MicroPython-Bibliothek für SSD1306 suchen

Direkt im nächsten Schritt drücken Sie auf den Button **Installieren**, wobei auf dem Pico ein Ordner **lib** angelegt wird und auch die MicroPython-Bibliothek für den SSD1306 genau in diesen Ordner **lib** hineinkopiert wird, siehe Abbildung 7.

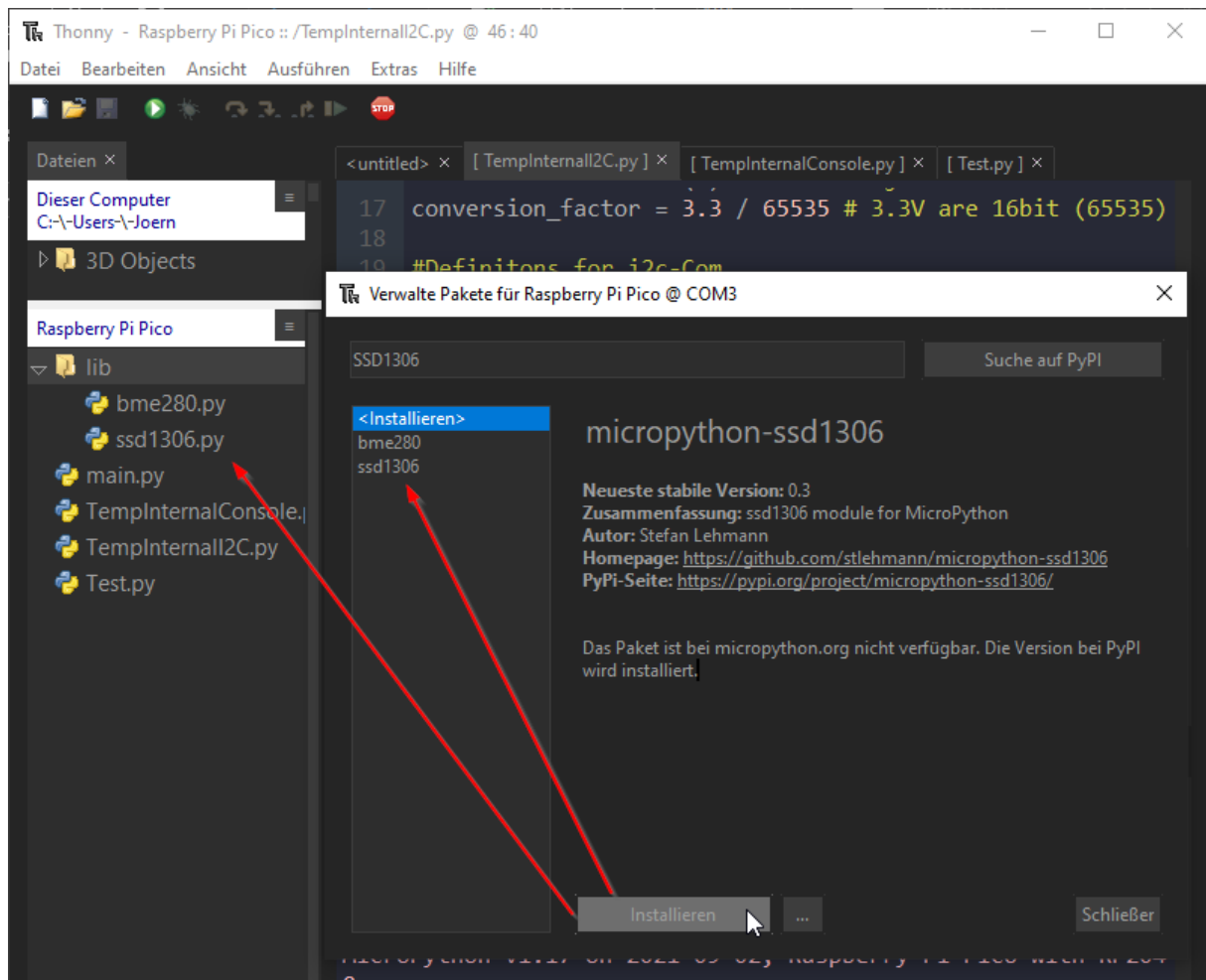


Abbildung 7: MicroPython-Bibliothek für SSD1306 installieren

Damit haben wir nun erst einmal die Bibliothek, nun muss noch das Display korrekt programmiert werden, siehe Code 3.

```
"""
// Read internal temp-sensor and
// show on i2c-OLED
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 23. Oct 2021
// Update: 24. Oct 2021
"""

#First import needed libs
from ssd1306 import SSD1306_I2C #Import from Lib the needed subpart
from machine import Pin, I2C
import machine #Important to get Pins and config them
import utime #For a little timer

#Definitions for internal temp-sensor
sensor = machine.ADC(4) # Create object sensor and init as pin ADC(4)
conversion_factor = 3.3 / 65535 # 3.3V are 16bit (65535)

#Definitons for i2c-Com
sda=Pin(0)
scl=Pin(1)
```

```

i2c=I2C(0, sda=sda, scl=scl, freq=400000)

#Write in cmd found addresses
i2cScan = i2c.scan()
counter = 0
for i in i2cScan:
    print("I2C Address " + str(counter) + "      : "+hex(i).upper())
    counter+=1

#Definitions for OLED-Display
WIDTH = 128
HIGHT = 64
oled = SSD1306_I2C(WIDTH, HIGHT, i2c)

i = 0
while True:
    valueTemp = sensor.read_u16() * conversion_factor #
    temp = 27 - (valueTemp - 0.706) / 0.001721 # See 4.9.5 of rp2040 datasheet
    print('Temperatur value: '+ '{:.2f}'.format(temp)) # Print in terminal with two decimal
    print('Counter: ' +str(i))
    #Write data to display
    oled.fill(0)
    oled.text('Internal Temp',6,8)
    oled.text('Temp: ',6,22)
    oled.text(str(round(temp,2)),50,22)
    oled.text('*C',95,22)
    oled.text('Counter: ' + str(i),6,36)
    oled.show()
    i+=1
    utime.sleep(2) #Sleep for two seconds

```

Code 3: Code aus ersten Beispiel um OLED-Display erweitert

In den Kommentaren wird genau erklärt, was ich mache, aber es gibt einige Sektionen, die ich kurz näher erläutern möchte. Angefangen mit „from ssd1306 import SSD1306_I2C“, bei dem wir aus der Bibliothek ssd1306 den Teil SSD1306_I2C laden bzw. bereitstellen. Damit muss nicht die gesamte Bibliothek geladen werden, sondern wir laden genau den Teilbereich, der für uns relevant ist.

Nach dem Kommentar „#Definitons for i2c-Com“ passiert auch erst einmal wenig Spannendes. Zunächst werden die Pins für SDA und SCL deklariert und danach die i2c-Schnittstelle **0** mit den bereits deklarierten Pins und der Frequenz 400kHz erstellt. Im Anschluss, was einfach nur für mich eine Interessante Ausgabe ist, fragt mein Programm alle verfügbaren Hex-Adressen ab, ob ein Teilnehmer vorhanden ist und gibt über die **for-Schleife** die gefundenen Teilnehmeradressen in der Kommandozeile aus.

Direkt danach, hinter dem Kommentar „#Definitions for OLED-Display“, deklariere ich die Größe des Displays und erzeuge ein Objekt mit allen nötigen Parametern für das Display. Im Anschluss wurde die **while-Schleife** so modifiziert, dass sowohl in der Kommandozeile als auch auf dem Display die Temperatur und ein kleiner Counter angezeigt wird, siehe Abbildung 8.

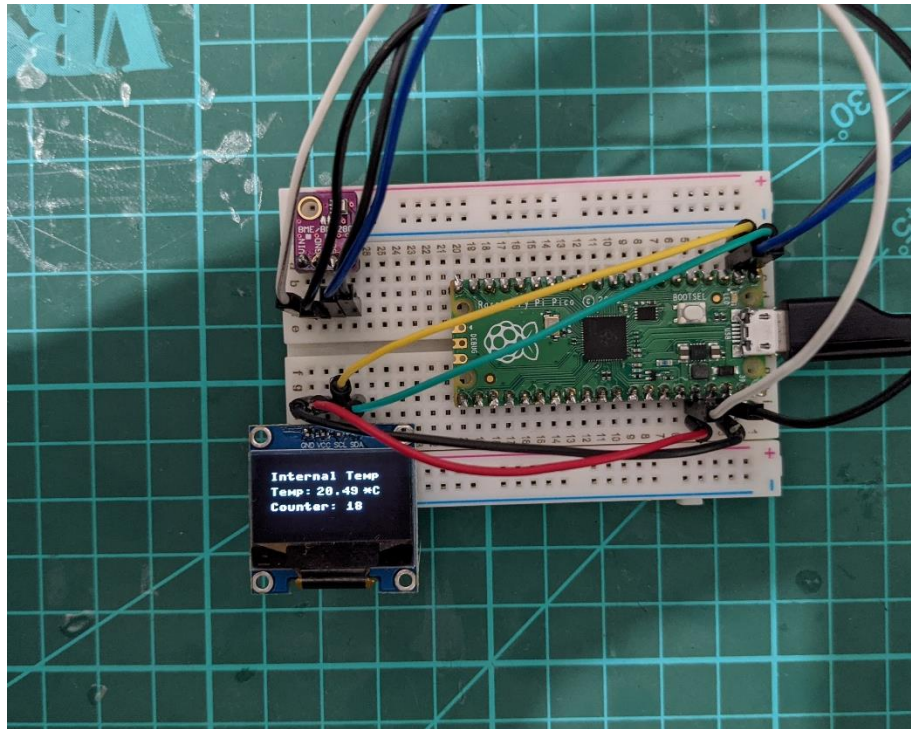


Abbildung 8: OLED-Ausgabe

Den Counter habe ich eingefügt, damit ich sehe, dass mein Pico noch arbeitet und nicht unbemerkt hängen geblieben ist.

Die Temperatur von einem GY-BME280 auslesen

Wie Sie wahrscheinlich schon bemerkt oder in der Doku vom Pico gelesen habe, ist die Temperaturangabe des intern verbauten Sensors nicht wirklich genau. An dieser Stelle nutze ich immer gerne den GY-BME280 Barometrischen Sensor für Temperatur, Luftfeuchtigkeit und Luftdruck, da er mir alle relevanten Daten direkt liefert. Schon in früheren Blogbeiträgen ist dieser Sensor von mir genutzt worden. Da der GY-BME280 ebenfalls über i2c kommuniziert, braucht es zunächst auch erst einmal nur vier weitere Drähte, zu den schon genutzten für das OLED-Display, siehe Abbildung 9.

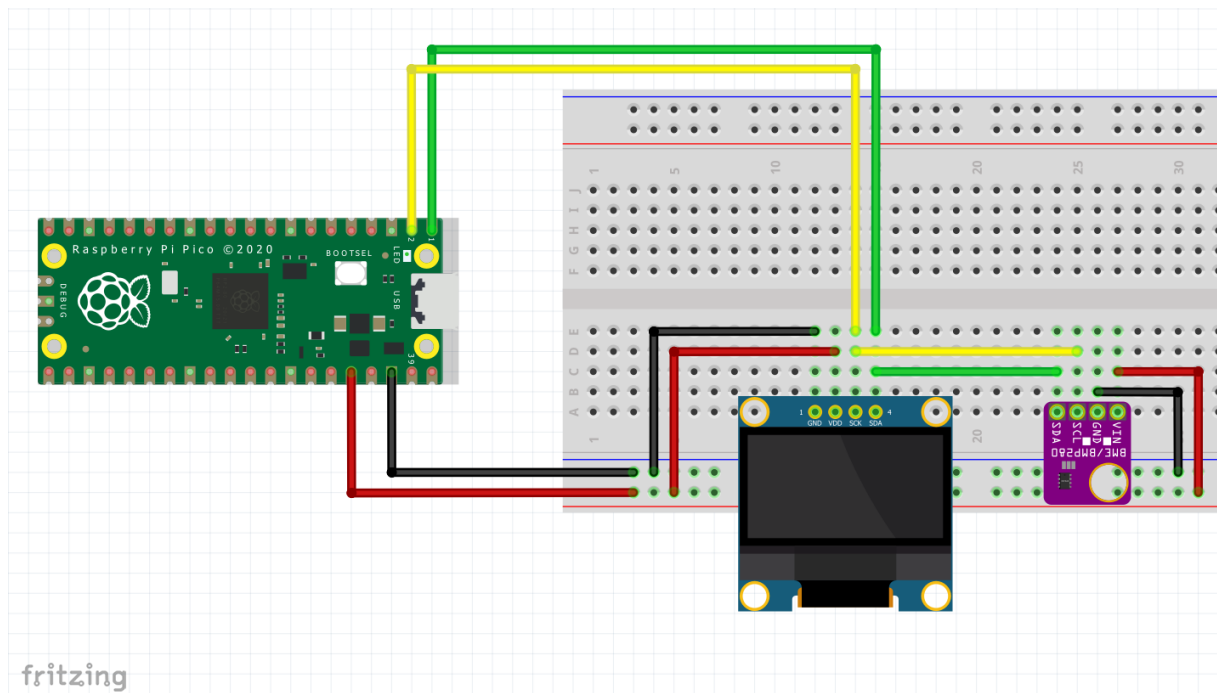


Abbildung 9: Das GY-BME280 anschließen

Wenn Sie die Datenblätter des OLED und GY-BME280 durchgelesen haben, sollte Ihnen aufgefallen sein, dass beide Unterschiedliche Adressen nutzen, daher können wir beide Komponenten in Reihe schalten. Wären die Adressen gleich, so müssten wir vom Pico eine weitere i2c-Schnittstelle erzeugen. Suchen Sie nun in der Paketverwaltung nach **bme280** und wählen Sie zur Installation die Bibliothek **micropython-bme280** aus, auch wenn in der Beschreibung die Rede vom ESP8266/ESP32 ist, siehe Abbildung 10.

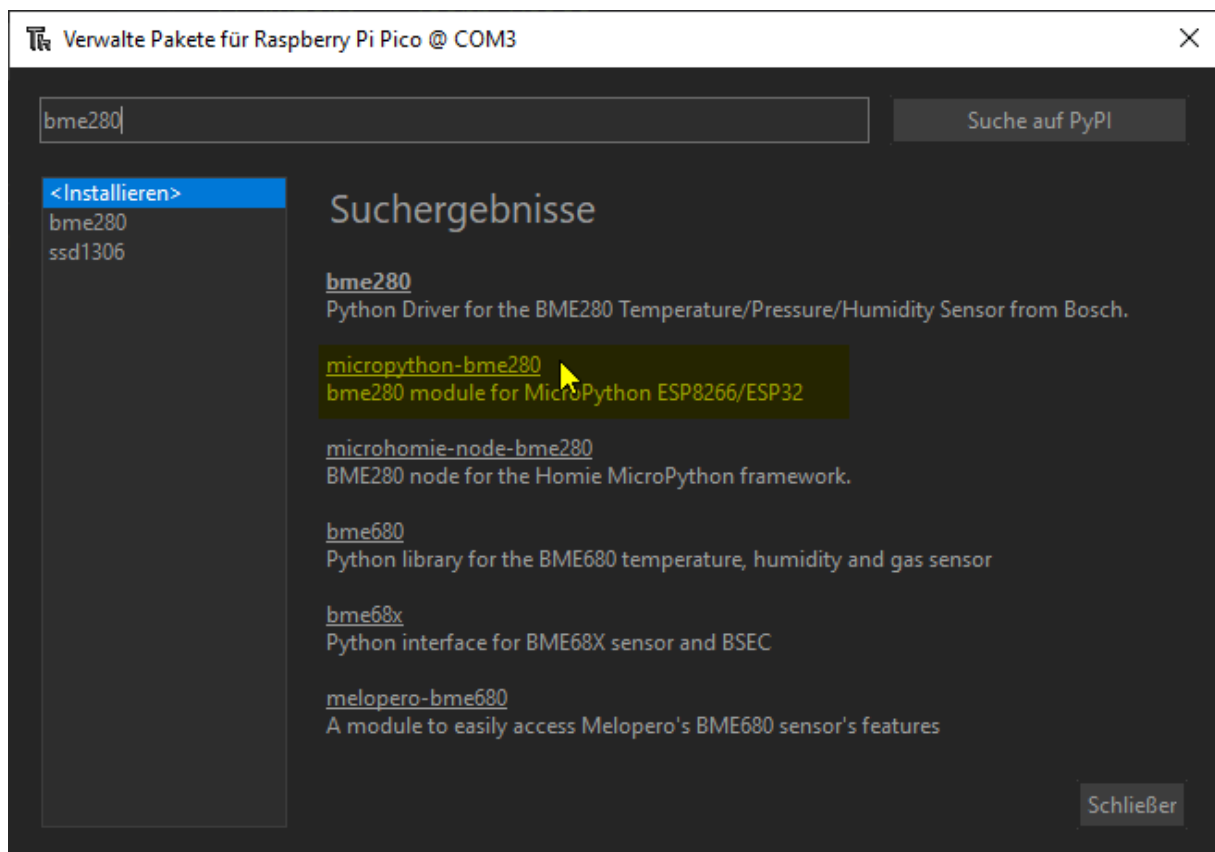


Abbildung 10: Bibliothek für BME280 installieren

Damit ist eine Bibliothek für unseren Sensor nun vorhanden und unser Beispielcode kann um ein paar Zeilen erweitert werden, siehe ""

```
// Read BME280 and
```

```
// show on i2c-OLED
```

```
// Autor: Joern Weise
```

```
// License: GNU GPI 3.0
```

```
// Created: 23. Oct 2021
```

```
// Update: 24. Oct 2021
```

```
""
```

```
#First import needed libs
```

```
from ssd1306 import SSD1306_I2C #Import from Lib the needed subpart
```

```
from bme280 import BME280 #Import BME280-lib
```

```
from machine import Pin, I2C
```

```
import machine #Important to get Pins and config them
```

```
import utime #For a little timer
```

```
#Definitions for internal temp-sensor
```

```
sensor = machine.ADC(4) # Create object sensor and init as pin ADC(4)
```

```
conversion_factor = 3.3 / 65535 # 3.3V are 16bit (65535)
```

```
#Definitons for i2c-Com
```

```
sda=Pin(0)
```

```
scl=Pin(1)
```

```
i2c=I2C(0, sda=sda, scl=scl, freq=400000)
```

```
#Write in cmd found addresses
```

```
i2cScan = i2c.scan()
```

```
counter = 0
```

```
for i in i2cScan:
```

```
    print('I2C Address ' + str(counter) + ' : '+hex(i).upper())
```

```
    counter+=1
```

```
print('-----')
```

```
#Definitions for OLED-Display
```

```
WIDTH = 128
```

```
HIGHT = 64
```

```
oled = SSD1306_I2C(WIDTH, HIGHT, i2c)
```

```
#Definition for BME280
```

```
sensorBME = BME280(i2c=i2c)
```

```
i = 0
```

```
while True:
```

```
    valueTemp = sensor.read_u16() * conversion_factor #
```

```
    temp = 27 - (valueTemp - 0.706) / 0.001721 # See 4.9.5 of rp2040 datasheet
```

```
    tempC, preshPa, humRH = sensorBME.values #Receive current values from GY-BME280
```

```
as tuple
```

```
    tempC = tempC.replace('C', '*C')
```

```
    print('Temperatur value: ' + '{:.2f}'.format(temp) + '*C') # Print in terminal with two decima
```

```
    print('Temperatur BME: ' + tempC)
```

```
    print('Pressure BME: ' + preshPa)
```

```
    print('Humidity BME: ' + humRH)
```

```
    print('Counter: ' + str(i))
```

```
    print('>-----<')
```

```
    #Write data to display
```

```
    oled.fill(0)
```

```

oled.text('GY-BME280 ',6,0)
oled.text('Temp:' + tempC,6,14)
oled.text('Pres:' + preshPa,6,28)
oled.text('Humi:' + humRH,6,42)
oled.text('Counter:' + str(i),6,56)
oled.show()
i+=1
utime.sleep(2) #Sleep for two seconds
Code 4.

```

```

'''

```

```

// Read BME280 and
// show on i2c-OLED
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 23. Oct 2021
// Update: 24. Oct 2021
'''

```

```

#First import needed libs
from ssd1306 import SSD1306_I2C #Import from Lib the needed subpart
from bme280 import BME280 #Import BME280-lib
from machine import Pin, I2C
import machine #Important to get Pins and config them
import utime #For a little timer

```

```

#Definitions for internal temp-sensor
sensor = machine.ADC(4) # Create object sensor and init as pin ADC(4)
conversion_factor = 3.3 / 65535 # 3.3V are 16bit (65535)

```

```

#Definitons for i2c-Com
sda=Pin(0)
scl=Pin(1)
i2c=I2C(0, sda=sda, scl=scl, freq=400000)

```

```

#Write in cmd found addresses
i2cScan = i2c.scan()
counter = 0
for i in i2cScan:
    print('I2C Address ' + str(counter) + ' : '+hex(i).upper())
    counter+=1

```

```

print('-----')
#Definitions for OLED-Display
WIDTH = 128
HIGHT = 64
oled = SSD1306_I2C(WIDTH, HIGHT, i2c)

```

```

#Definition for BME280
sensorBME = BME280(i2c=i2c)

```

```

i = 0
while True:
    valueTemp = sensor.read_u16() * conversion_factor #
    temp = 27 - (valueTemp - 0.706) / 0.001721 # See 4.9.5 of rp2040 datasheet
    tempC, preshPa, humRH = sensorBME.values #Receive current values from GY-BME280
    as tuple

```

```

tempC = tempC.replace('C', '*C')
print('Temperatur value: ' + '{:.2f}'.format(temp) + '*C') # Print in terminal with two decima
print('Temperatur BME: ' + tempC)
print('Pressure BME: ' + preshPa)
print('Humidity BME: ' + humRH)
print('Counter: ' + str(i))
print('>-----<')
#Write data to display
oled.fill(0)
oled.text('GY-BME280 ',6,0)
oled.text('Temp:' + tempC,6,14)
oled.text('Pres:' + preshPa,6,28)
oled.text('Humi:' + humRH,6,42)
oled.text('Counter:' + str(i),6,56)
oled.show()
i+=1
utime.sleep(2) #Sleep for two seconds

```

Code 4: Messung mit GY-BME280

Wie schon im Vorgängercode sind die Änderungen überschaubar. Zunächst wird **BME280** aus der Bibliothek **bme280** geladen. Hinter dem Kommentar „#Definition for BME280“ wird ein Objekt **sensorBME** angelegt und die i2c-Schnittstelle übergeben. In der **while-Schleife** passiert dann der interessante Teil. Hier wird mittels „sensorBME.values“ die für Menschen lesbaren Umgebungsdaten in einem [Tuple](#) an die Variablen tempC, preshPa, humRH übergeben.

Im Anschluss werden diese Daten auf dem Display, siehe Abbildung 11, wiedergegeben.

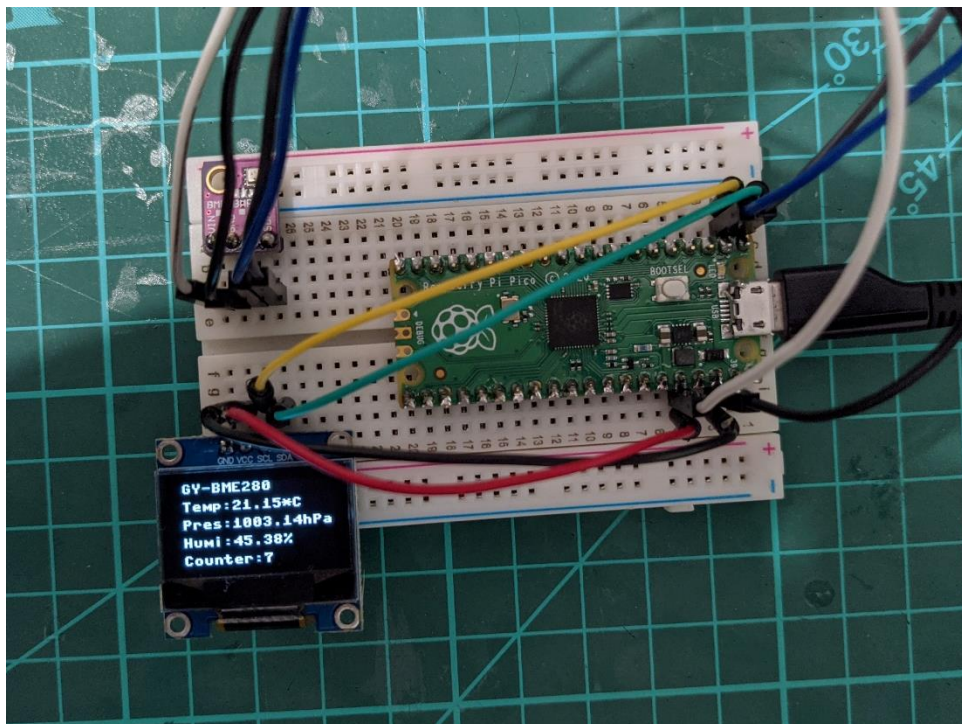
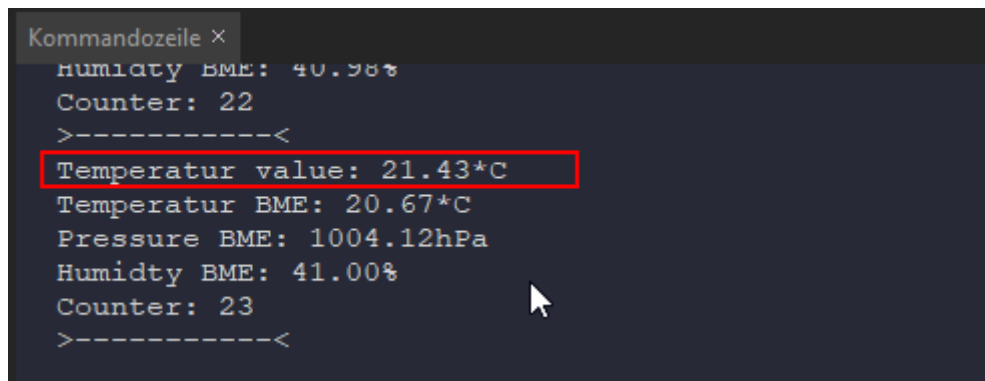


Abbildung 11: BME280-Daten auf dem OLED-Display

Da ich trotzdem Interesse daran habe, was denn der Pico für Werte liefert, habe ich diese in der Kommandozeilenausgabe gelassen, siehe Abbildung 12.



```
Kommandozeile x
Humidity BME: 40.98%
Counter: 22
>-----<
Temperatur value: 21.43*C
Temperatur BME: 20.67*C
Pressure BME: 1004.12hPa
Humidity BME: 41.00%
Counter: 23
>-----<
```

Abbildung 12: Kommandozeilenausgabe

Mein Skript läuft nicht / wird ohne PC nicht gestartet

Sie werden wahrscheinlich schon versucht haben, die Skripte in Thonny Python IDE zu verwenden. Hier müssen Sie unbedingt auf zwei Dinge achten!!!

1. Wenn Sie ein Python-Skript speichern, müssen Sie immer die Endung **.py** mit beim Speichern eintragen. Andernfalls wird im Folgenden der Code nicht mehr erkannt und Änderungen sind nicht mehr möglich.
2. Damit Ihr Code auch ohne PC und Thonny Python IDE funktioniert, muss die auszuführende Datei **main.py** auf dem Raspberry Pi Pico heißen. Wenn der Pico diese Datei findet, wird das enthaltene Skript ausgeführt.

Beides habe ich einmal in Abbildung 13 noch einmal zusammengefasst.

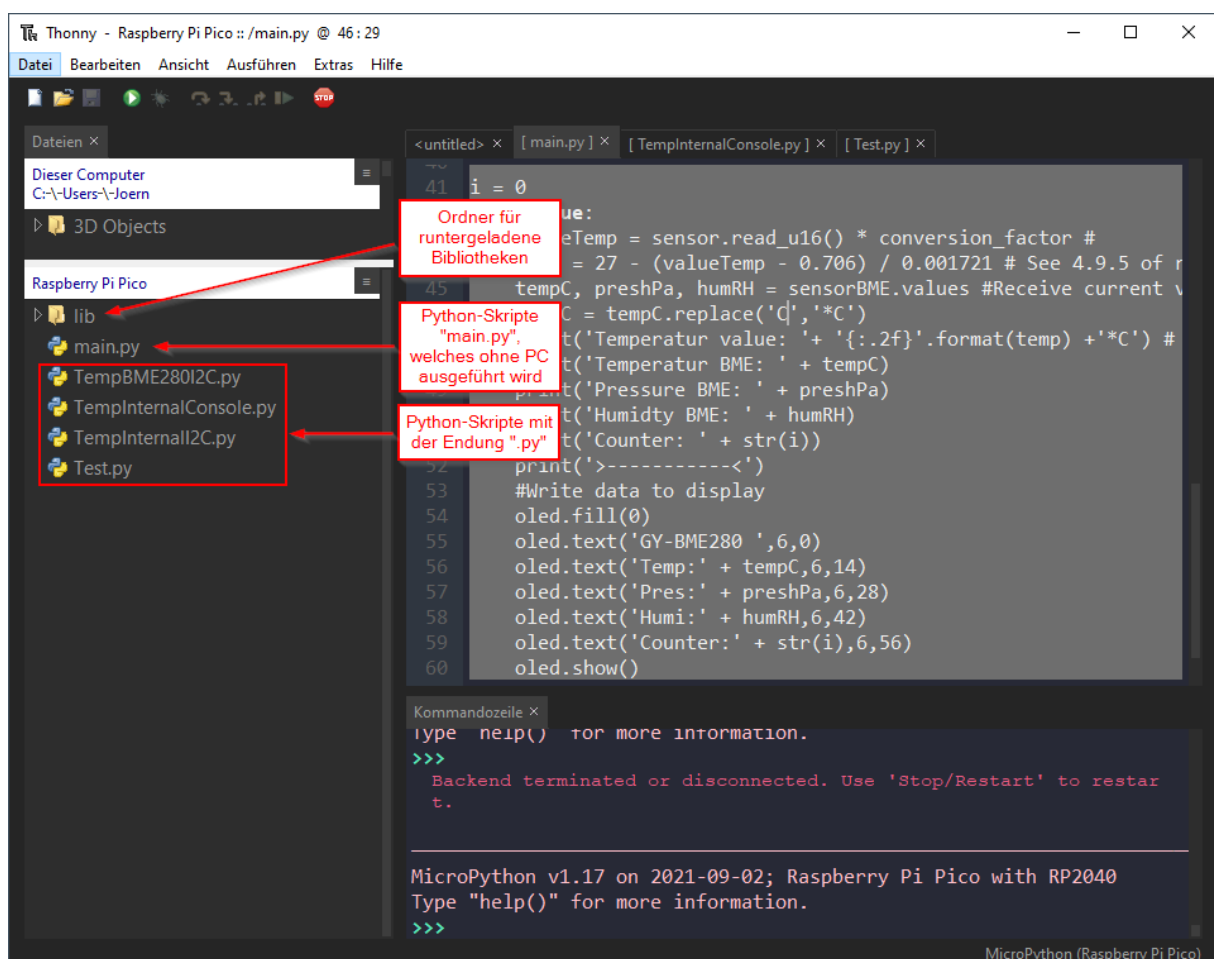


Abbildung 13: Eigenarten vom Pico

Zusammenfassung

Auch wenn Sie bisher „nur“ mit unseren MicroControllern ESP32/ESP8266 oder Nano v3.0 und der Arduino IDE gearbeitet haben, sollte dieses kleine Beispiel recht schnell umsetzbar sein. Es gibt tatsächlich die Möglichkeit die Arduino IDE zum Programmieren des Pico zu verwenden (einige Infos dazu [hier](#)), jedoch sollten Sie, was meine persönliche Meinung ist, mit der Thonny Python IDE und microPython programmieren. Letzteres hat den Vorteil, dass Sie hier eine der wohl wichtigsten Programmiersprachen, nämlich Python, lernen. Diese Sprache ist mittlerweile so beliebt, dass auch viele Hackertools mit dieser Skriptsprache umgesetzt werden.

Versuchen Sie ruhig diese drei Beispiele genauer zu verstehen und vllt. zwei separate i2c-Verbindungen zu erzeugen. Möglich wäre auch, dass Sie zwei GY-BME280 verwenden und den Mittelwert aus beiden Messungen verwenden. Die Modifikation dieser doch recht einfachen Beispiele sind fast grenzenlos, sollte aber doch den recht einfachen Einstieg zeigen. Dies hoffe ich, ist mir gelungen, sodass Sie auch mal zum Pico greifen und nicht zum doch mächtigeren ESP32.

Weitere Projekte für Az-Delivery von mir, finden Sie unter <https://github.com/M3taKn1ght/Blog-Repo>.