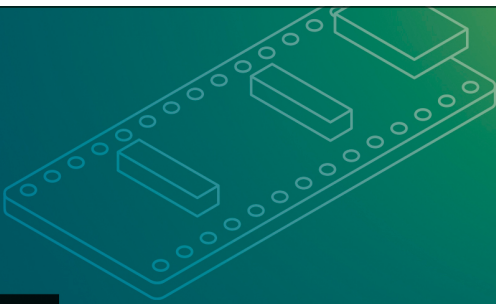




Claus Kühnel

MicroPython

Das Handbuch für Maker
und IoT-Development

</>

○○○

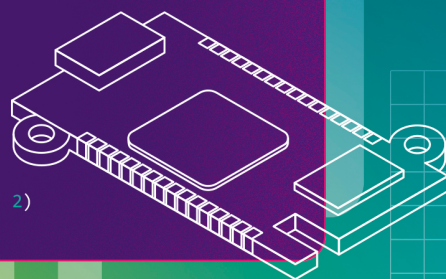
```
from machine import Pin, I2C

# create an I2C bus
i2c = I2C(scl=Pin('X1'), sda=Pin('X2'))

# scan for list of attached devices
dev_list = i2c.scan()

# write to and read from a device
i2c.writeto(0x42, b'4')
data = i2c.readfrom(0x42, 4)

# memory transactions
i2c.writeto_mem(0x42, 0x12, b'')
data = i2c.readfrom_mem(0x42, 0x12, 2)
```



- ▶ Mikrocontroller mit Python programmieren
- ▶ Multitasking, Timer, Interrupts und Netzwerke
- ▶ Projekte vom LoRa-Netz bis zur eigenen Wetterstation



Mit Codebeispielen und Praxisprojekten



Rheinwerk
Computing

Kapitel 4

Daten speichern, teilen und anzeigen

Sie haben nun die grundlegenden Elemente von MicroPython kennengelernt und wissen, wie Sie Logik in Kontrollstrukturen aufbauen und in Funktionen kapseln. Das ist wichtig, reicht aber nicht aus, denn Sie wollen die Daten ja auch speichern, über ein Netzwerk teilen und anzeigen. Wie das geht, schauen wir uns nun an.

4.1 Dateien und Dateisystem

Das Modul `os` in MicroPython stellt grundlegende Funktionen für den Umgang mit dem Dateisystem und betriebssystemnahe Informationen bereit. In Abschnitt 3.5.2 hatte ich Ihnen unter `os` das Modul bereits vorgestellt. Hier möchte ich das aus Anwendungssicht vertiefen.

4.1.1 Skript-Struktur: `main.py`, `boot.py`

Im Zusammenhang mit dem WDT haben Sie bereits die Startsequenz eines MicroPython-Programms kennengelernt.

Das Programm `boot.py` wird stets zuerst ausgeführt und dient dem Low-Level-Setup. Neben der boardspezifischen Initialisierung kann beispielsweise das Netzwerk vorbereitet oder der Garbage Collector konfiguriert werden. Den Quelltext in `boot.py` sollten Sie kurzhalten; Endlosschleifen und blockierende Tasks sind nicht zulässig.

Das nach `boot.py` gestartete Programm `main.py` ist das eigentliche Anwendungsprogramm. Im Gegensatz zu `boot.py` darf `main.py` Endlosschleifen enthalten. Das Verhalten von MicroPython ist abhängig vom Programm `main.py`, wie Ihnen Tabelle 4.1 zeigt.

Endlosschleife in <code>main.py</code>	Verhalten
nein	Programm endet. REPL erscheint (<code>>>></code>).
ja	Programm läuft dauerhaft. REPL ist blockiert (außer per Interrupt).

Tabelle 4.1: Verhalten von MicroPython gesteuert durch `main.py`

4.1.2 Dateisystem-Funktionen

Der guten Übersicht halber liste ich in Tabelle 4.2 die Dateisystem-Funktionen auf.

Funktion	Anweisung	Beispiel
Listet Verzeichnisse und Dateien.	<code>os.listdir([path])</code>	<code>os.listdir('/lib')</code>
Erstellt neues Verzeichnis.	<code>os.mkdir(path)</code>	<code>os.mkdir('data')</code>
Löscht (leeres) Verzeichnis.	<code>os.rmdir(path)</code>	<code>os.rmdir('data')</code>
Löscht eine Datei.	<code>os.remove(file)</code>	<code>os.remove('data.csv')</code>
Benennt Datei oder Verzeichnis um.	<code>os.rename(old, new)</code>	<code>os.rename('old.txt', 'new.txt')</code>
Wechselt in neues Verzeichnis.	<code>os.chdir(path)</code>	<code>os.chdir('/lib')</code>
Gibt das aktuelle Verzeichnis zurück.	<code>os.getcwd()</code>	<code>cwd = os.getcwd()</code>
Gibt Statusinformationen zu einer Datei oder einem Verzeichnis zurück. (Typ, Größe, Zeitstempel ...)	<code>os.stat(path)</code>	<code>info = os.stat('test.txt')</code>
Gibt Informationen zum Dateisystem zurück (Speicherplatz).	<code>os.statvfs(path)</code>	<code>fs = os.statvfs('/')</code>

Tabelle 4.2: Funktionen Modul *os*

Mit den in Tabelle 4.2 gelisteten Funktionen lassen sich die erforderlichen Aktivitäten im Dateisystem durchführen und Informationen zum Dateisystem abfragen. Abbildung 4.1 zeigt Ihnen die in Listenform ausgegebenen Informationen zu einem ESP32-C3.

```
>>> os.stat('/')
(16384, 0, 0, 0, 0, 0, 0, 0, 0, 0)
>>> os.statvfs("/")
(4096, 4096, 512, 498, 498, 0, 0, 0, 0, 255)
```

Abbildung 4.1: Ausgaben *os.stat('/')* und *os.statvfs('/')*

Die Bedeutung der Indizes in beiden Informationen zeigen Tabelle 4.3 und Tabelle 4.4.

Index	Bedeutung
<code>st[0]</code>	Dateityp und Rechte (Bitmaske)
<code>st[1]</code>	Inode (meist 0 oder nicht genutzt)

Tabelle 4.3: Statusinformationen

Index	Bedeutung
st[2]	Gerät (meist 0)
st[3]	Anzahl Links
st[4]	User-ID (meist 0)
st[5]	Group-ID (meist 0)
st[6]	Dateigröße in Bytes
st[7]	Zugriffszeit (Epoch)
st[8]	Änderungszeit
st[9]	Erstellungszeit

Tabelle 4.3: Statusinformationen (Forts.)

In der Praxis sind st[0] und st[6] relevant.

Index	Bedeutung
fs[0]	f_bsize – Blockgröße in Bytes
fs[1]	f_frsize – Fragmentgröße
fs[2]	f_blocks – Gesamtanzahl Blöcke
fs[3]	f_bfree – freie Blöcke
fs[4]	f_bavail – für Programme verfügbare freie Blöcke
fs[5]	f_files – maximale Inodes (meist nicht genutzt)
fs[6]	f_ffree – freie Inodes
fs[7]	f_favail – verfügbare Inodes
fs[8]	f_flag – Mount-Flags
fs[9]	f_namemax – maximale Dateinamenslänge

Tabelle 4.4: Speicherinformation

Hier sind in der Praxis auch nur fs[0], fs[2], fs[3] und fs[9] relevant.

Etwas aussagefähiger kann die Ausgabe in dem kleinen Tool *fs_info.py* gestaltet werden (Listing 4.1). Das Programm selbst weist keine Besonderheiten auf. Die Rückgabewerte

der Funktion `os.statvfs(path)` werden miteinander verknüpft und ausgegeben. Abbildung 4.2 zeigt die deutlich lesbarere Ausgabe der Informationen zum Dateisystem.

```
import os
def fs_info(path="/"):
    fs = os.statvfs(path)
    block = fs[0]
    total = fs[2] * block
    free = fs[3] * block
    used = total - free
    print("File System Info:", path)
    print("-----")
    print("Total   :", total, "Bytes")
    print("Used    :", used, "Bytes")
    print("Free    :", free, "Bytes")
    print("Blocks  :", fs[2])
    print("Max Name:", fs[9], "Characters")
fs_info()
```

Listing 4.1: Programmbeispiel `fs_info.py`

```
File System Info: /
-----
Total    : 2097152 Bytes
Used     : 57344 Bytes
Free     : 2039808 Bytes
Blocks   : 512
Max Name : 255 Characters
```

Abbildung 4.2: Ausgabe Programmbeispiel `fs_info.py`

Abbildung 4.2 zeigt, dass 57.344 der insgesamt zur Verfügung stehenden 2.097.152 Bytes aktuell verwendet werden, wodurch noch 2.039.808 Bytes frei sind. Diese Ausgaben sind sicher wichtig, um den noch zur Verfügung stehenden Speicherplatz zu kennen, aber interessant ist auch, wer wie viel des Speicherplatzes belegt. In der Thonny IDE sind die Verzeichnisse und Dateien im **Files**-Window sichtbar (siehe Abbildung 4.3).

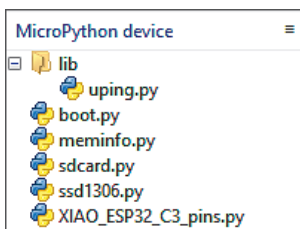


Abbildung 4.3: Dateien im Files-Window in Thonny

Aus MicroPython heraus liefert `os.listdir()` zwar die Namen der Verzeichnisse und Dateien, aber nicht deren Größe. Hier kann wieder ein kleines Tool wie *ls.py* Abhilfe schaffen (Listing 4.2).

```
import os
def ls(path="/"):
    for name in os.listdir(path):
        if path == "/":
            full = "/" + name
        else:
            full = path + "/" + name
        st = os.stat(full)
        mode = st[0]
        if mode & 0x4000: # Verzeichnis
            print("📁", name)
        else:             # Datei
            print("📄", name, "-", st[6], "Bytes")
ls()
```

Listing 4.2: Programmbeispiel *ls.py*

Bevor der Aufruf `os.stat(full)` erfolgt, wird der Pfad zur einzelnen Datei systematisch erstellt. Handelt es sich um eine Datei, dann wird über `st[6]` ihre Größe abgefragt und zusätzlich zum Dateinamen ausgegeben. Abbildung 4.4 zeigt die Ausgaben von `os.listdir()` und `ls()` im Vergleich. Letztere zeigt zur in Abbildung 4.3 gezeigten Struktur auch noch die Dateigrößen an.

```
>>> os.listdir()
['XIAO_ESP32_C3_pins.py', 'boot.py', 'lib', 'meminfo.py', 'sdcard.py', 'ssd1306.py']
>>> ls()
📄 XIAO_ESP32_C3_pins.py - 381 Bytes
📄 boot.py - 139 Bytes
📁 lib
📄 meminfo.py - 697 Bytes
📄 sdcard.py - 7939 Bytes
📄 ssd1306.py - 4461 Bytes
>>>
```

Abbildung 4.4: Ausgaben `os.listdir()` und `ls()` im Vergleich

Die Arbeit mit den Dateisystem-Funktionen können Sie sehr gut im Programmbeispiel *rot_log.py*, einem rotierenden Datenlogger, nachvollziehen. Das Programmbeispiel finden Sie im Repository, hier möchte ich es aus Platzgründen nicht listen.

Dieser »FlashLogger« zeichnet sich durch rotierende Logdateien (*log0*, *log1* usw.) aus. Die Rotation erfolgt automatisch, wenn die maximale Dateigröße von *log0* erreicht ist. Vor je-

dem Schreiben erfolgt deshalb eine Speicherprüfung. Jeder Eintrag wird mit einem Zeitstempel versehen. Über das Kommando `ls_logs()` bekommen Sie eine Übersicht über die existierenden Logdateien.

Die Bedienung des FlashLoggers in der REPL zeigt Abbildung 4.5. Die resultierende Logdatei sehen Sie in Abbildung 4.6.

```
>>> log = FlashLogger(base_name="log", max_files=5, max_size=2048, warn_free_bytes=1024)
>>> log.write("Sensor started")
>>> log.write("Temperature: 23.5°C")
>>> log.write("Temperature: 24.1°C")
>>> log.ls_logs()

log0 - 110 Bytes

>>> log.write("Temperature: 24.1°C")
>>> log.write("Temperature: 23.5°C")
>>> log.ls_logs()

log0 - 174 Bytes

>>>
```

Abbildung 4.5: Ein- und Ausgaben Programmbeispiel `rot_log.py` in der REPL

```
=== Log started ===
[18:03:13] Sensor started
[18:03:41] Temperature: 23.5°C
[18:03:44] Temperature: 24.1°C
[18:04:09] Temperature: 24.1°C
[18:04:12] Temperature: 23.5°C
```

Abbildung 4.6: Dateiinhalt `log0`

4.1.3 Erweiterung des Speichers durch SD-Karten

Obwohl moderne Mikrocontroller mit immer mehr Speicher ausgestattet werden, kann bedingt durch die jeweilige Anwendung eine externe Speichererweiterung erforderlich werden.

Typische Anwendungsaspekte können

- Offlinebetrieb ohne Internetverbindung,
- Datenlogging von Sensoren, Audiodaten, IoT-Events u. a. m.,
- Konfiguration und Backups,
- große Dateien (Firmware, ML-Modelle, Bilder, Audio)

sein.

Selbst im Fall einer bestehenden Onlineverbindung kann ein Offline-Logging aus Backup- bzw. Sicherheitsgründen sinnvoll sein.

Es gibt eine Reihe von Mikrocontroller-Boards, die bereits einen SD-Card-Slot mit einem SDIO- oder SPI-Interface aufweisen. Die Unterschiede dieser beiden Interfaces zeigt Tabelle 4.5.

	SDIO / SDMMC	SPI Mode
Busbreite	1 Bit oder 4 Bit	1 Bit
Geschwindigkeit	sehr hoch	deutlich langsamer
Hardware	spezieller SD-Controller	normales SPI
Software	komplexer	sehr einfach
MCU-Support	nur manche MCUs	praktisch alle
Pins	viele	wenige

Tabelle 4.5: SDIO vs. SPI

Abbildung 4.7 zeigt mit dem LilyGo TTGO T8 v1.8 und dem Arduino MKR Zero zwei Boards, die einen SD-Card-Slot aufweisen und auch von MicroPython unterstützt werden.

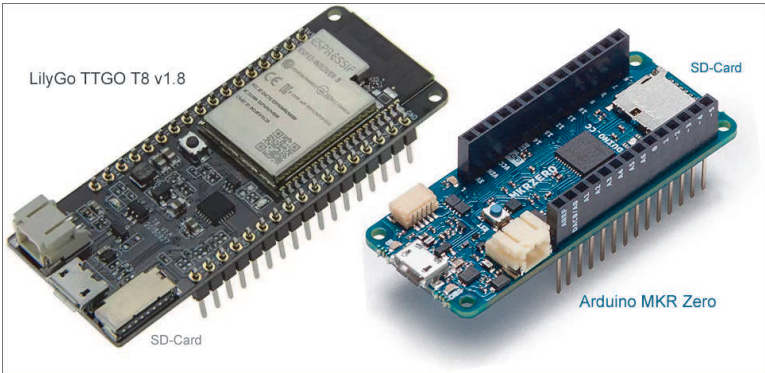


Abbildung 4.7: Mikrocontroller-Boards mit SD-Card (Auswahl)

Da das LilyGo TTGO T8 v1.8 mit einem ESP32-WROVER ausgestattet ist, ist es für zahlreiche Anwendungen das interessantere Board von den zweien.

Der ESP32 hat zwei SD Memory Card Controller (SDMMC-Hosts), die die Datenübertragung zur SD-Card optimieren. Host 0 führt an Slot 1, Host 1 an Slot 2. Damit MicroPython SDMMC nutzen kann (`machine.SDCard(slot=...)`), müssen die Pins des SD-Card-Slots hardware-seitig korrekt verdrahtet sein (Tabelle 4.6):

Signal	Slot 1 (Host 0) GPIO (typ.)	Slot 2 (Host 1) GPIO (typ.)
CLK	14	18
CMD	15	5

Tabelle 4.6: ESP32 SDMMC – Pins

Signal	Slot 1 (Host 0) GPIO (typ.)	Slot 2 (Host 1) GPIO (typ.)
D0	2	19
D1	4	optional
D2	12	optional
D3	13	optional
CS	abhängig / frei wählbar	abhängig / frei wählbar

Tabelle 4.6: ESP32 SDMMC – Pins (Forts.)

Die Pins können je nach ESP32-Variante abweichen, müssen aber mit dem SDMMC-Host verbunden sein, sonst schlägt `slot=...` fehl.

ESP32-WROVER-Kits führen die Pins nach außen, haben aber keinen SD-Card-Slot auf dem Board. Beim LilyGo TTGO T8 steht ein SD-Card-Slot auf dem Board zur Verfügung, doch leider folgt die Verdrahtung nicht den Vorgaben in Tabelle 4.6, und es bleibt nur das SPI-Interface für den Anschluss der SD-Card übrig.

SDIO- vs. SPI-Interface

Bieten Mikrocontroller SD-Card-Support durch SDMMC-Hosts, dann müssen die betreffenden Pins auch mit dem SD-Card-Slot verbunden sein, und der MicroPython-Port muss diese Kommunikation auch unterstützen. Sind diese Voraussetzungen nicht gegeben, dann bleibt das SPI-Interface die gängige Variante zum Datenaustausch zwischen Mikrocontroller und SD-Card.

Zum Test der SD-Card verwende ich das Programmbeispiel *ESP32_SD_test.py*, das ich hier aber nur ausschnittsweise erläutern möchte. Das verwendete Modul `sdcard` kann von <https://github.com/micropython/micropython-lib/blob/master/micropython/drivers/storage/sdcard/sdcard.py> heruntergeladen und auf dem Board installiert werden. Zu importieren sind außerdem die folgenden Module:

```
from machine import SPI, Pin
import os, sdcard, time
```

Die Initialisierung des SPI-Interface mit den Angaben aus dem Datenblatt erfolgt im nächsten Schritt:

```
cs = Pin(13, Pin.OUT, value=1)
spi = SPI(1, baudrate=10_000_000, polarity=0, phase=0,
         sck=Pin(14), mosi=Pin(15), miso=Pin(2))
```

Mit diesen Vorgaben kann nun das SD-Card-Objekt erstellt und als */SD* gemountet werden. Die Anweisungen `os.listdir("/sd")` und `ls("/sd")` zeigen nun den Inhalt des neuen Verzeichnisses an (siehe Abbildung 4.8):

```
try:
    sd = sdcard.SDCard(spi, cs)
    os.mount(sd, "/sd")
    print("SD mounted")
    print(os.listdir("/sd"))
    ls("/sd")
except OSError as e:
    print("SD init failed:", e)
```

Das Schreiben von Daten auf die SD-Card erfolgt nach dem Öffnen der Datei durch `open(filename, "w")` zum Schreiben durch die Anweisung `f.write()`.

```
# Write test file to sd card
print('\nWrite data to sd card...')
filename = "/sd/test.txt"
with open(filename, "w") as f:
    f.write("Hallo MicroPython!\n")
    f.write("Data written to SD card.\n")
```

Das (Zurück-)Lesen der Datei erfolgt nach dem Öffnen der Datei `open(filename, "r")` zum Lesen durch die Anweisung `f.read()`.

```
# Read file back
print('\nRead file content back from sd card...')
with open(filename, "r") as f:
    print("File content:")
    print(f.read())
```

Zum Schluss wird die auf der SD-Card erzeugte Datei wieder von der SD-Card entfernt.

```
# Remove file from sd card
print('\nRemove file from sd card...')
os.remove(filename)
```

Abbildung 4.8 zeigt die gesamten Ausgaben des Programmbeispiels. Zwischen den einzelnen Schreib- und Lesevorgängen wird jeweils der Inhalt des Verzeichnisses zur Kontrolle ausgegeben.

Im Repository können Sie zwei weitere Tools zum Test von SD-Cards finden. Das Programm *sd_benchmark.py* testet die Geschwindigkeit des Datenaustauschs für größere Dateien. Dem Einsatz in Sensorknoten näher kommt das Programm *logger_stress_test.py*.

```
ESP32 SD-Card Test via SPI
SD mounted
['System Volume Information', 'logger_test.bin']

Type  Size      Name
-----
d      0 System Volume Information
- 192000 logger_test.bin

Write data to sd card...

Read file content back from sd card...
File content:
Hallo MicroPython!
Data written to SD card.

File Size: 44 Bytes

Type  Size      Name
-----
d      0 System Volume Information
- 192000 logger_test.bin
-      44 test.txt
Free memory (Bytes): 15922823168

Remove file from sd card...

Type  Size      Name
-----
d      0 System Volume Information
- 192000 logger_test.bin
Free memory (Bytes): 15922855936
```

Abbildung 4.8: Ausgaben Programmbeispiel ESP32-SD_test.py

4.2 Kommunikation

Gerade für IoT-Devices ist die Kommunikation ein essenzieller Aspekt, den ich in diesem Abschnitt eingehend behandeln möchte. Tabelle 4.7 zeigt die für die hier betrachteten Mikrocontroller vorhandenen Kommunikationstechnologien.

Technologie	Board-Support	Typische Reichweite	Verwendung
WiFi	ESP32, Pico W	30–100 m	Internet, MQTT, REST
LAN	ESP32, STM32	drahtgebunden	Internet, MQTT, REST
ESP-NOW	ESP32	30–100 m	Low-Latency Mesh
BLE	ESP32, nRF52	10–50 m	Sensor → App, Notifications
LoRa	ESP32 + SX127x	1–10 km	Sensornetzwerke, IoT
nRF24	ESP32, STM32	50–200 m	DIY Mesh, Remote

Tabelle 4.7: Kommunikationstechnologien

4.2.1 WiFi

Für die Konfiguration der WiFi-Verbindung weist das Modul `network` die Klasse `network.WLAN` auf. Es existieren zwei WiFi-Interfaces: eines für eine *Station* (STA) und eines für einen *Access Point* (AP). Station und Access Point sind zwei komplett unterschiedliche Rollen im Netzwerk.

WiFi vs. WLAN

WLAN (*Wireless Local Area Network*) ist ein allgemeiner technischer Begriff, während WiFi ein Markenname der WiFi Alliance ist – einer gemeinnützigen Organisation, die die gängigen und künftigen WLAN-Standards verwaltet. Im praktischen Sprachgebrauch sind beide Begriffe identisch.

Station bezeichnet einen Client oder Teilnehmer, der sich mit einem vorhandenen WLAN verbindet, um sich mit dem Internet zu verbinden, Daten zu senden (MQTT, API, Cloud) oder, allgemein gesprochen, Teil des Netzwerkes zu sein.

Ein Access Point baut ein eigenes WLAN auf, mit dem sich Clients verbinden können. Ein solcher AP kann zum Beispiel ein Konfigurations-Portal bereitstellen, ein Setup ohne Internet ermöglichen oder Webserver hosten.

Ich zeige Ihnen im Folgenden die Konfiguration von Station und AP, um anschließend beide Rollen auf einem Mikrocontroller zu vereinen.

Station (STA)

Um eine Verbindung zu einem WLAN-Router im Netzwerk aufzubauen, sind die einzelnen Schritte der Funktion `connect_wifi()` auszuführen (Listing 4.3).

```
import network, time
from credentials import WIFI_SSID, WIFI_PASSWORD
# Configuration Data in credentials.py
# Connect WiFi
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print("Connect with WiFi...")
        wlan.connect(WIFI_SSID, WIFI_PASSWORD)
        while not wlan.isconnected():
            print('.', end=" ")
            time.sleep(1)
        print("\nWiFi connected:", wlan.ifconfig()[0])

connect_wifi()
```

Listing 4.3: Programmbeispiel `connect_wifi.py`

Die Anweisung `wlan = network.WLAN(network.STA_IF)` erstellt das Netzwerk-Interface im STA-Mode zum Router. Nach Aktivierung des Interfaces kann durch `wlan.connect(WIFI_SSID, WIFI_PASSWORD)` die Verbindung aufgebaut werden.

Die erforderlichen Einwahldaten SSID und Passwort habe ich in der Datei *credentials.py* versteckt. Im Abschnitt 3.5.3 habe ich diese Auslagerung bereits beschrieben. Zu Beginn des Programms wurden diese beiden Daten importiert.

Nehmen Sie das in Listing 4.4 gezeigte Template als Ausgangspunkt für die diversen Zugangsdaten, und benennen Sie die aktualisierte Version in *credentials.py* um, bevor diese auf dem MicroPython-Device gespeichert wird. Durch die Anweisung `from credentials import WIFI_SSID, WIFI_PASSWORD` stehen Ihnen dann SSID und Passwort im Programm zur Verfügung.

```
# credentials_template.py
WIFI_SSID = 'Your SSID'
WIFI_PASSWORD = 'Your Password'
PUSHOVER_USER_KEY = 'Your Pushover User Key'
PUSHOVER_API_TOKEN = 'Your pushover API Token'
PUSHOVER_URL = 'Pushover URL'
```

Listing 4.4: Quelltext *credential_template.py*

Solange noch keine Verbindung zum WLAN aufgebaut ist, signalisiert im Programm *connect_wifi.py* eine Reihe von Punkten das Warten auf die Verbindungsaufnahme. Ist diese erfolgt, dann wird die zugewiesene IP-Adresse ausgegeben (siehe Abbildung 4.9).

```
Connect with WiFi...
....
WiFi connected: 192.168.1.230
```

Abbildung 4.9: WiFi-Verbindungsaufnahme

Access Point (AP)

Um einen Access Point im Netzwerk bereitzustellen, sind die einzelnen Schritte der Funktion `start_ap()` auszuführen (Listing 4.5).

```
import network
def start_ap():
    ap = network.WLAN(network.AP_IF)
    ap.active(True)
    ap.config(
        essid="ESP32_AP",
        password="12345678", # mind. 8 Zeichen
        authmode=network.AUTH_WPA_WPA2_PSK
    )
```

```

print("AP gestartet")
print("IP:", ap.ifconfig()[0])
start_ap()

```

Listing 4.5: Programmbeispiel connect_ap.py

Die Anweisung `ap = network.WLAN(network.AP_IF)` erstellt das Netzwerk-Interface im AP-Mode. Nach Aktivierung des Interfaces können durch die Funktion `ap.config()` und die Werte `ESSID = "ESP32_AP"` das Passwort und der Authentifizierungs-Mode eingestellt werden. Der so erzeugte AP meldet sich dann im Netzwerk als ESP32_AP und erwartet das Passwort 12345678. Sie sind vollkommen frei in der Wahl der Zugangsdaten.

Ist die Konfiguration erfolgt, wird die zugewiesene IP-Adresse ausgegeben (siehe Abbildung 4.10), und der ESP32_AP ist im Netzwerk sichtbar (siehe Abbildung 4.11).

```

AP gestartet
IP: 192.168.4.1

```

Abbildung 4.10: AP-Verbindungsaufnahme

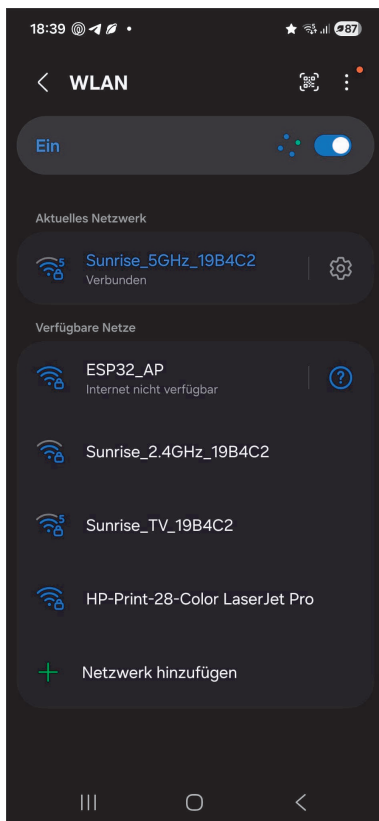


Abbildung 4.11: ESP32_AP im Netzwerk

STA und AP kombiniert

Durch die gleichzeitige Implementierung von STA und AP erhalten Sie beide Welten. Vielleicht ist Ihnen das schon mal bei der Konfiguration eines neuen IoT-Geräts begegnet.

Das noch nicht ins WLAN eingebundene Gerät startet und baut ein eigenes WLAN (hier **ESP32_AP**) auf. Mit dem Mobilphone kann eine Verbindung zum ESP32_AP aufgebaut werden. Die Zugangsdaten für das zukünftig zu verwendende Router-WLAN werden eingegeben und abgespeichert, und künftig verbindet sich das Gerät direkt als STA.

Das Programmbeispiel *STA_AP.py* zeigt den Aufbau der beiden Verbindungen, wobei hier die Zugangsdaten für den STA-Mode bekannt und wieder in *credentials.py* abgespeichert sind (Listing 4.6).

```
import network
from credentials import WIFI_SSID, WIFI_PASSWORD
ap = network.WLAN(network.AP_IF)
sta = network.WLAN(network.STA_IF)
ap.active(True)
ap.config(essid="ESP32_AP", password="12345678")
sta.active(True)
sta.connect(WIFI_SSID, WIFI_PASSWORD)
print("AP IP:", ap.ifconfig()[0])
print("STA IP:", sta.ifconfig()[0])
```

Listing 4.6: Programmbeispiel *STA_AP.py*

Abbildung 4.12 zeigt die IP-Adressen für STA im Heimnetzwerk und den AP, der über die Verbindung zum ESP32_AP erreicht werden kann.

```
AP IP: 192.168.4.1
STA IP: 192.168.1.229
```

Abbildung 4.12: IP-Adressen für AP und STA

Mit der Einwahl selbst ist es im geschilderten Fall aber noch nicht getan. Die folgenden Schritte sind erforderlich:

- WiFi Reset: `wifi_reset()`
- Laden vorhandener Zugangsdaten: `cfg = load_config()`
- Start STA: `sta = start_sta(cfg)`
- Start AP: `ap = start_ap()`
- Start Webserver zur Dateneingabe: `start_server()`

Im Programmbeispiel *web_config_pro.py* sind diese Aktivitäten vereint. Das Programm finden Sie im Repository unter https://github.com/ckuehnel/MicroPython2026/blob/main/General/web_config_pro.py.

Neu im Programm sind die HTML-Seite zur Eingabe der Zugangsdaten und der Webserver. Die HTML-Seite wird in der Funktion `html_page()` (Listing 4.7) aufgebaut, zum Webserver komme ich in Abschnitt 8.5.

```
def html_page():
    nets = scan_networks()
    options = ""
    for n in nets:
        options += f"<option value='{n[0]}'>{n[0]} ({n[1]}dBm)</option>"
    return f"""HTTP/1.0 200 OK

<html>
<h2>TTGO Pro Config Portal</h2>
<form action="/save">
SSID:<br>
<select name="ssid">{options}</select><br>
Password:<br>
<input name="password" type="password"><br><br>
<input type="submit">
</form>
</html>
"""
```

Listing 4.7: Quelltext HTML-Seite

Den eigentlichen HTML-Code habe ich im Quelltext hervorgehoben. Sie können diesen Teil in einem HTML-Editor betrachten und den Aufbau der späteren Website begutachten.

Damit im SSID-Feld die vorhandenen Netzwerke gelistet werden können, wird das Netzwerk zuerst gescannt, und SSID und Feldstärke in dBm werden in `options` gespeichert. Abbildung 4.13 zeigt die APs in meinem Netzwerk.

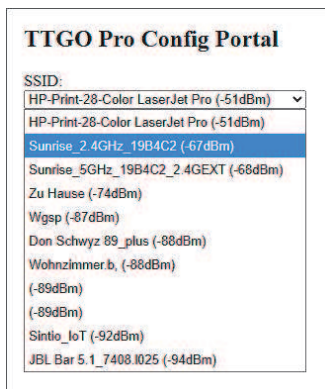
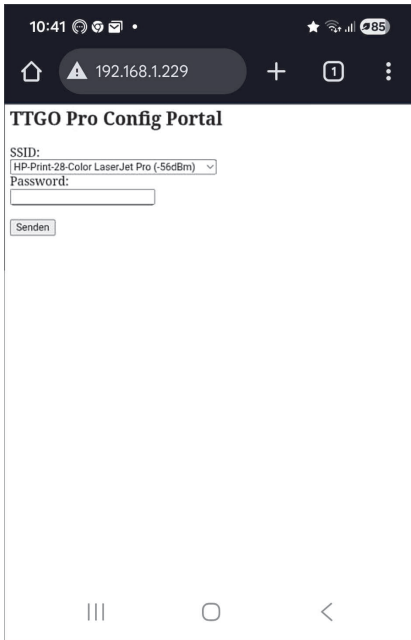


Abbildung 4.13: Aktuelle APs im Netzwerk

Rufe ich die STA-IP 192.168.1.229 vom Mobilphone aus auf, dann erscheint die wenig befriedigende Website aus Abbildung 4.14.


Abbildung 4.14: Website `web_config_pro.py`

Eine Anpassung an die Displaygröße des Mobilphones muss unbedingt vorgenommen werden.

Durch den Einbau von Viewport und größeren Schriften lässt sich das Erscheinungsbild deutlich verbessern. Den veränderten HTML-Code zeigt die aktualisierte Funktion `html_page()` (Listing 4.8). Das angepasste Programm `web_config_pro+.py` finden Sie ebenfalls im Repository.

```
def html_page():
    nets = scan_networks()
    options = ""
    for n in nets:
        options += f"<option value='{n[0]}'>{n[0]} ({n[1]}dBm)</option>"
    return f"""HTTP/1.0 200 OK

<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {{
    font-size: 20px;
    font-family: Arial;
    padding: 15px;
}}
```

```

input, select, button {{
    font-size: 20px;
    width: 100%;
    margin-top: 10px;
    padding: 10px;
}}
</style>
</head>
<body>
<h2>TTGO Pro Config Portal</h2>
<form action="/save">
SSID:<br>
<select name="ssid">{options}</select><br>
Password:<br>
<input name="password" type="password"><br><br>
<input type="submit">
</form>
</body>
</html>
" " "

```

Listing 4.8: Quelltext erweiterte HTML-Seite

Beim Aufruf der STA-IP 192.168.1.229 vom Mobilphone aus wird nun die angepasste und deutlich besser lesbare Website erscheinen (siehe Abbildung 4.15).

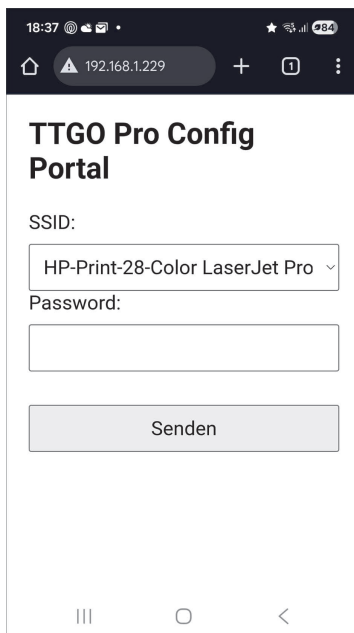


Abbildung 4.15: Website web_config_pro+.py

4.2.2 Ethernet

In den vorhergehenden Abschnitten haben Sie mit WiFi eine drahtlose Netzwerkverbindung kennengelernt. Bei kabelgebundenen LANs (*Local Area Networks*) ist Ethernet die meistbenutzte Technologie und kann als Pendant zu WiFi betrachtet werden.

Ethernet ist WiFi dann überlegen, wenn Sie ein Gateway oder einen Server betreiben, dauerhaft kommunizieren, 24/7-Onlineverfügbarkeit sicherstellen müssen und eine feste Installation haben.

Interessant ist außerdem die Möglichkeit, die Spannungsversorgung gleichzeitig über das Ethernet-Kabel vorzunehmen. Die PoE-Technik (*Power Over Ethernet*) ist gerade für stationäre IoT-Knoten eine interessante Variante.

Den grundsätzlichen Aufbau einer Ethernet-Schnittstelle zeigt Abbildung 4.16. Es gibt Mikrocontroller, die den MAC bereits enthalten und extern nur noch den PHY und die Magnetics benötigen. Mikrocontroller, die keinen MAC aufweisen, können über SPI einen externen TCP/IP-Controller, wie beispielsweise einen WIZnet W5500 oder Infineon CYW43439, ansteuern, der MAC und PHY beinhaltet.

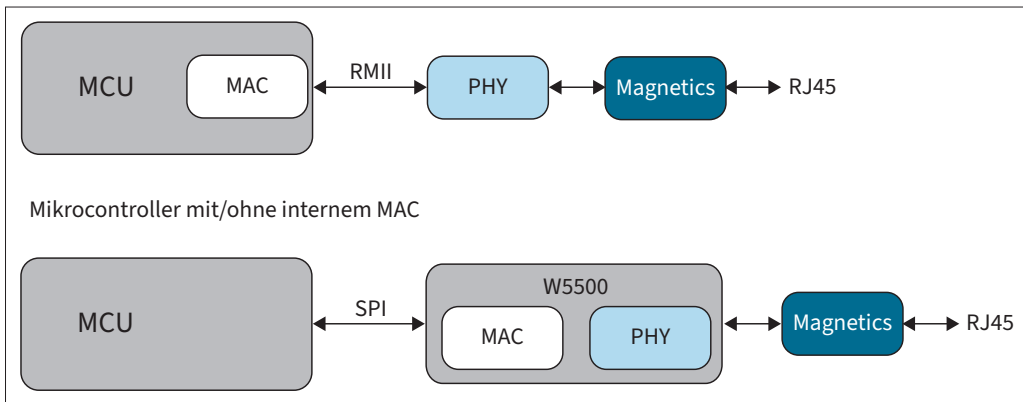


Abbildung 4.16: Aufbau einer Ethernet-Schnittstelle

Der Media Access Controller (MAC) ist die digitale Logik, die das Ethernet-Protokollhandling übernimmt, während der Physical Layer (PHY) die analoge, elektrische Schnittstelle zwischen MAC und den Magnetics darstellt. Beispiele für PHY-Chips sind LAN8720, DP83848, KSZ8081, RTL8211 (Gigabit). Die Magnetics sind bei Ethernet die Übertrager und Drosseln zwischen PHY und RJ45-Buchse. Sie sorgen dafür, dass die Signale galvanisch getrennt, sauber gefiltert und normgerecht aufs Kabel gehen. Oft sind alle diese Teile in der RJ45-Buchse integriert (»MagJack«). Abbildung 4.17 zeigt einen solchen Mag-Jack.



Abbildung 4.17: MagJack

Ob der betreffende Port von MicroPython Ethernet unterstützt, finden Sie durch die folgenden Kommandos schnell heraus. Das verwendende Mikrocontroller-Board muss das dann natürlich hardware-seitig auch tun.

```
>>> import network
>>> if "LAN" in dir(network):
    print("LAN supported")
LAN supported
```

Ich habe gute Erfahrungen mit dem Olimex ESP32-EVB-EA (<https://www.olimex.com/Products/IoT/ESP32/ESP32-EVB/>) gemacht, bei dem neben dem LAN- auch ein WLAN-Interface zur Verfügung steht, wodurch es für Bridge-Anwendungen sehr geeignet ist. Außerdem stehen microSD Card, CAN-Interface und zwei Relais (10 A/250 V AC Wechselspannung) mit externen Anschlüssen und Status-LED zur Verfügung.



Abbildung 4.18: Olimex ESP32-EVB-EA

Den Bootprozess für das Ethernet-Interface zeigt das Programmbeispiel *ETH_boot.py* (Listing 4.9). Während des Bootprozesses signalisieren das Relais und seine Status-LED den jeweiligen Status. Wichtig ist die LAN-Initialisierung. Sehen Sie sicherheitshalber im Schaltbild des Boards nach den Verbindungen von Mikrocontroller und PHY nach. Abbildung 4.19 zeigt Ihnen die Ausgaben des Programms während des Bootprozesses.

```
# ETH_boot.py - ESP32-EVB-EA LAN Check
import machine, network, time, _thread, socket
# --- Relay setup ---
rel1 = machine.Pin(32, machine.Pin.OUT)
rel1.off()
# --- PHY Power / Reset ---
pwr = machine.Pin(12, machine.Pin.OUT)
pwr.off()
time.sleep_ms(200)
pwr.on()
time.sleep_ms(200)
# --- LAN Initialization ---
lan = network.LAN(
    mdc=machine.Pin(23),
    mdio=machine.Pin(18),
    power=pwr,
    phy_type=network.PHY_LAN8720,
    phy_addr=0
)
lan.active(True)
# --- Wait for Ethernet link ---
print("Wait for Ethernet-Link...")
timeout = 20
link_ok = False
for _ in range(timeout):
    rel1.on()
    time.sleep(0.5)
    if lan.isconnected():
        rel1.off()
        link_ok = True
        break
    print(".", end="")
    time.sleep(0.5)
if link_ok:
    print("\nEthernet connected!")
    ip = lan.ifconfig()[0]
    print("IP-Configuration:", ip)
    rel1.off() # steady off = LAN OK
```

```

else:
    print("\nNo Link! Check cable / switch / power-pin.")
    for _ in range(3):
        rel1.on()
        time.sleep(0.3)
        rel1.off()
        time.sleep(0.3)
print("Boot complete. LAN status:", "OK" if link_ok else "FAIL")

```

Listing 4.9: Programmbeispiel *ETH_boot.py*

```

Wait for Ethernet-Link...

Ethernet connected!
IP-Configuration: 192.168.1.220
Boot complete. LAN status: OK

```

Abbildung 4.19: Ausgaben Programmbeispiel *ETH_boot.py*

Im Repository ist noch das Programmbeispiel *ETH_EVB_Relay_Control.py* enthalten, das die Steuerung der Relais auf dem ESP32-EVB-EA über das Netzwerk zulässt.

4.2.3 Ethernet-WiFi-Kombination

Die Verbindung über Ethernet und WiFi ist im Programm *ESP32_automnet.py* kombiniert. AutoNet erkennt automatisch

- Ethernet (LAN),
- WiFi STA,
- AP Config Mode (Fallback)

und läuft auf allen ESP32.

Das Programm finden Sie im Repository. Hier möchte ich nur die Anwendung aufzeigen.

Kopieren Sie das Programm *ESP32_automnet.py* in das MicroPython-Device. Ersetzen Sie die Angaben für SSID und Passwort der Einfachheit halber im folgenden Quelltext, und kopieren Sie das Programm als *main.py* ebenfalls in das MicroPython-Device. Das Programm *boot.py* kann ohne Inhalt bleiben.

```

import automnet
net = automnet.connect(
    ssid="Your SSID",
    password="Your Password"
)

```

Listing 4.10: Verbinde *automnet* in *main.py*

4.2.4 ESP-NOW

ESP-NOW ist ein vom chinesischen Hersteller Espressif entwickeltes, drahtloses Kommunikationsprotokoll, das speziell für ESP8266- und ESP32-Mikrocontroller optimiert wurde. Es ermöglicht eine direkte Peer-to-Peer-Kommunikation zwischen Geräten – ganz ohne WLAN-Access-Point, Router oder Internetverbindung. Dadurch ist ESP-NOW besonders schnell, energieeffizient und zuverlässig.

Im Gegensatz zu klassischem WLAN werden bei ESP-NOW kleine Datenpakete direkt zwischen den Knoten ausgetauscht. Die Latenz ist sehr gering, der Overhead minimal, und mehrere Geräte können gleichzeitig miteinander kommunizieren (*one to one*, *one to many* oder *many to many*). Optional unterstützt ESP-NOW auch eine AES-Verschlüsselung, was es für sichere IoT-Anwendungen attraktiv macht.

Typische Einsatzbereiche sind Sensornetzwerke, Fernbedienungen, Smart-Home-Komponenten, mesh-ähnliche Strukturen oder batteriebetriebene Geräte, bei denen Stromverbrauch und Reaktionszeit entscheidend sind.

Auf dem ESP32 stellt MicroPython für ESP-NOW auch AES-Verschlüsselung zur Verfügung, die Teil des ESP-NOW-Protokolls ist. AES-128 wird automatisch vom WiFi-Stack genutzt.

In MicroPython wird AES-128 aktiviert, indem Peers mit einem PMK/LMK (Primary/Local Master Key) hinzugefügt werden. Die eigentliche Ver- und Entschlüsselung passiert transparent im Hintergrund. Sie können also verschlüsselt senden und empfangen, ohne selbst AES implementieren zu müssen. ESP-NOW ist ein schlanker MAC-Frame mit Espressif-Payload. Details MicroPython für ESP-NOW finden Sie unter <https://docs.micropython.org/en/latest/library/espnw.html>.

Damit eine zielgerichtete Kommunikation aufgebaut werden kann, muss der Sender die Adresse des jeweiligen Empfängers kennen. Verwendet wird hier die MAC-Adresse (Media Access Control Address) des betreffenden Controllers – eine eindeutige Hardware-Kennung, die jedes Gerät in einem Netzwerk identifiziert.

Um die MAC-Adressen der beteiligten ESP8266 oder ESP32 zu ermitteln, bietet sich das Programm `ESP32_MAC_Addr.py` an (Listing 4.11).

```
import network
sta = network.WLAN(network.STA_IF)
sta.active(True)
peer = sta.config('mac')
print('\nMAC Address:')
print(peer)
print('.'.join(f'{b:02X}' for b in peer))
```

Listing 4.11: Programmbeispiel `ESP32_MAC_Addr.py`

Der spätere Netzwerkknoten wird in den STA-Mode versetzt und die MAC-Adresse abgefragt. Es erfolgt eine Ausgabe als Bytearray und in der üblichen, durch : separierten

Form. Es empfiehlt sich, den Netzwerkknoten mit einem Label, der diese Adresse ausweist, zu versehen.

Ein einfaches Beispiel soll die Initialisierung und den Datenaustausch verdeutlichen. Zuerst ist die MAC-Adresse der Empfängerseite abzufragen, da sie vom Sender benötigt wird (siehe Abbildung 4.20).

```
MAC Address:
b'\xb4\x8a\nb\x9eh'
B4:8A:0A:62:9E:68
```

Abbildung 4.20: MAC-Adresse des Empfängers

Danach können Sie das Programm *ESP32_ESPNow_Receive.py* auf dem Empfänger installieren. Hierfür habe ich die Arduino MicroPython IDE verwendet. Das in Listing 4.12 gezeigte Programm bietet wenig Neues. Ich habe die wichtigen Stellen fett markiert. Die Initialisierung von ESP-NOW beschränkt sich auf zwei Zeilen Quelltext, und die empfangenen Daten enthalten die MAC-Adresse des Senders sowie die Message an sich (*Payload*).

```
import network, espnow, time
# WLAN im STA-Modus starten
sta = network.WLAN(network.STA_IF)
sta.active(True)
# ESP-NOW initialisieren
e = espnow.ESPNow()
e.active(True)
print("ESP-NOW Receiver started...")
while True:
    host, msg = e.recv()
    if msg:
        print("From MAC:",
              ':'.join(f'{b:02X}' for b in host),
              "\tMessage:",
              msg.decode())
    time.sleep(0.1)
```

Listing 4.12: Programmbeispiel *ESP32_ESPNow_Receive.py*

Anschließend installieren Sie auf der Senderseite das Programm *ESP32_ESPNow_Transmit.py* (Listing 4.13). Hierfür habe ich die Thonny IDE verwendet. Der Programmaufbau selbst ist vergleichbar mit dem Empfänger, nur dass hier die Adresse des Empfängers mitgeteilt werden muss. Kennt der Sender bereits die Empfängeradresse, dann wird der resultierende `OSError` abgefangen, anderenfalls wird die neue Adresse in die Liste der

Empfänger eingetragen. Schließlich wird hier im Takt von fünf Sekunden eine Message mit einer Countervariablen versendet, deren Empfang protokolliert wird.

```
import network, espnow, time
# MAC-Adresse des Empfängers (ANPASSEN!)
peer = b'\xb4\x8a\nb\x9eh'
# WLAN im STA-Modus starten
sta = network.WLAN(network.STA_IF)
sta.active(True)
# ESP-NOW initialisieren
e = espnow.ESPNow()
e.active(True)
# Peer hinzufügen
try:
    e.add_peer(peer)
except OSError:
    print('Peer already exists.')
print("ESP-NOW Transmit started.")
counter = 0
while True:
    msg = f"Hallo {counter}"
    e.send(peer, msg)
    print("Gesendet:", msg)
    counter += 1
    time.sleep(5)
```

Listing 4.13: Programmbeispiel ESP32_ESPNow_Transmit.py

Abbildung 4.21 zeigt die Datenübertragung der beiden ESP-NOW-Knoten in den beiden IDEs. Im Shell Window der Thonny IDE sehen Sie die Ausgaben der Senderseite, während in der Arduino MicroPython IDE die Protokollierung der empfangenen Daten erfolgt. Hier erkennen Sie an der MAC-Adresse, wer Daten gesendet hat und sehen den Inhalt der Message.

Wollen Sie die ESP-NOW-Verschlüsselung (AES-128) nutzen, dann ist Folgendes zu berücksichtigen:

Es gibt zwei jeweils 16 Byte umfassende Schlüssel:

- PMK (Primary Master Key) ist ein globaler Netzwerkschlüssel.
- LMK (Local Master Key) ist ein peer-spezifischer Schlüssel.

Beide Schlüssel müssen exakt 16 Bytes lang sein. Die Verschlüsselung funktioniert nur bei Unicast – Broadcast kann nicht verschlüsselt werden.

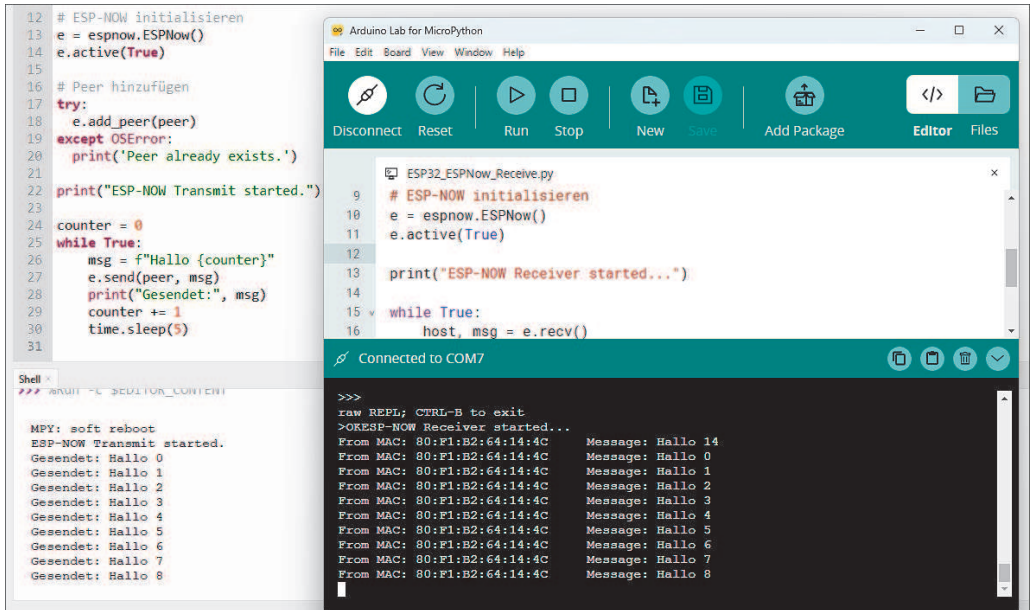


Abbildung 4.21: ESP-NOW-Datenübertragung

Die Schlüssel sollten nicht einfach vorgegeben, sondern möglichst generiert werden. Hierzu können Sie z. B. Pseudo-Zufallszahlen oder spezielle Algorithmen heranziehen. Mit dem Programmbeispiel `ESP32_ESPNow_PMK_LMK.py` steht ein kleines Tool für die Generierung der Schlüssel bereit (Listing 4.14).

```
import os, ubinascii
from hmac_sha256 import hmac_sha256
# Peer MAC-Adresse
Peer_MAC = b'\xb4\x8a\nb\x9eh'
print('\nGeneration of PMK & LMK...')
PMK = os.urandom(16)
LMK = os.urandom(16)
print('\nRandom generated:')
print("PMK:", ubinascii.hexlify(PMK))
print("LMK:", ubinascii.hexlify(LMK))
print('\nKey derivation using HMAC-SHA256:')
LMK = hmac_sha256(PMK, Peer_MAC)[:16]
print("LMK:", ubinascii.hexlify(LMK))
```

Listing 4.14: Programmbeispiel `ESP32_ESPNow_PMK_LMK.py`

PMK und LMK werden zuerst durch eine Pseudo-Zufallszahl gebildet. Diese beiden Schlüssel lassen sich im Flash speichern. Durch eine Schlüsselableitung (*Key Derivation Function* – KDF) kann der LMK im Peer automatisch erstellt werden. MicroPython hat kein

Inhalt

Danke	10
1 Python 3 vs. MicroPython	11
1.1 Popularität von Programmiersprachen	12
1.2 Gemeinsamkeiten und Unterschiede	14
2 Entwicklungsboards und -umgebungen	19
2.1 Eingesetzte Hardware	20
2.1.1 Bildungssysteme	20
2.1.2 Entwicklungsboards	24
2.2 Entwicklungsumgebungen	43
2.2.1 Texteditor und mpremote reichen	44
2.2.2 Thonny	46
2.2.3 Arduino Lab for MicroPython	48
2.2.4 MAiXPy IDE	49
2.2.5 OpenMV IDE	50
2.3 Firmware installieren	51
2.3.1 Pyboard	52
2.3.2 Nucleo-F446RE	54
2.3.3 BBC micro:bit	55
2.3.4 ESP32	56
2.3.5 Teensy 4.0	59
2.3.6 Raspberry Pi Pico	59
2.3.7 Arduino Nano und Portenta	61
2.3.8 MAiX BiT	63
2.3.9 OpenMV Cam H7 R2	64
2.4 MicroPython-Benchmarks	66
2.4.1 micropython-pystone-lowmem.py	67
2.4.2 micropython_mega_benchmark.py	68
2.4.3 Interpretation der Resultate insgesamt	70
3 Die Sprachelemente von MicroPython	73
3.1 Grundlegende Syntax	73
3.1.1 Interaktive Eingaben in der Shell	73

3.1.2	Einrückungen und Codeblöcke	75
3.1.3	Kommentare	76
3.1.4	Konstanten, Variablen und Namenskonventionen	76
3.1.5	Datentypen (int, float, str, bool)	78
3.1.6	Tupel	79
3.1.7	Listen	79
3.1.8	Dictionaries	81
3.1.9	Sets	83
3.1.10	Strings und ihre Methoden	85
3.2	Operatoren	87
3.2.1	Arithmetische Operatoren	87
3.2.2	Vergleichsoperatoren	88
3.2.3	Logische Operatoren (and, or, not)	89
3.2.4	Zuweisungsoperatoren (+, -=, ...)	91
3.3	Kontrollstrukturen	92
3.3.1	if ... elif ... else	92
3.3.2	for-Schleifen	93
3.3.3	while-Schleifen	95
3.3.4	break, continue, pass	97
3.3.5	try ... except	98
3.4	Funktionen	102
3.4.1	Built-in-Funktionen	102
3.4.2	Anwenderspezifische Funktionen	106
3.4.3	Lambda-Funktionen	110
3.5	Module in MicroPython	111
3.5.1	Importmechanismus	111
3.5.2	Wichtige eingebaute Module	113
3.5.3	Benutzerdefinierte Module	128
3.6	Besonderheiten von MicroPython	131
3.6.1	Digitale Ein- und Ausgabe (I/O)	132
3.6.2	Analoge Ein- und Ausgabe (I/O)	135
3.6.3	I ² C-Bus	139
3.6.4	Serial Peripheral Interface (SPI)	142
3.6.5	UART	146
3.6.6	1-Wire	149
3.6.7	Interrupts	154
3.6.8	Timer	158
3.6.9	Watchdog	161

4	Daten speichern, teilen und anzeigen	167
4.1	Dateien und Dateisystem	167
4.1.1	Skript-Struktur: main.py, boot.py	167
4.1.2	Dateisystem-Funktionen	168
4.1.3	Erweiterung des Speichers durch SD-Karten	172
4.2	Kommunikation	176
4.2.1	WiFi	177
4.2.2	Ethernet	184
4.2.3	Ethernet-WiFi-Kombination	187
4.2.4	ESP-NOW	188
4.2.5	Bluetooth Low Energy (BLE)	192
4.2.6	Kommunikationsmodule mit AT-Kommando-Interface	205
4.3	Anzeigeeinheiten	210
4.3.1	Konventionelle LEDs	210
4.3.2	NeoPixel	211
4.3.3	OLED	214
4.3.4	LCD	215
4.3.5	E-Paper	224
5	Fortgeschrittene Programmierung	229
5.1	Multitasking mit Threads und AsyncIO	229
5.1.1	Threads	229
5.1.2	AsyncIO	233
5.1.3	Threads vs. AsyncIO	234
5.2	State Machines	235
5.3	Raspberry Pi Pico: PIO-Programmierung	239
5.4	Bildverarbeitung	248
5.4.1	OpenMV Cam H7 R2	248
5.4.2	MAiX BiT	252
6	Debugging	257
6.1	Statische Codeanalyse	257
6.2	Debugging über REPL	259
6.3	print()-Debugging	261
6.4	Thonny-Debugger	262

7	Best Practices	265
7.1	Auswahlkriterien für den Mikrocontroller	265
7.1.1	RAM und Flash	266
7.1.2	Performance der CPU	266
7.1.3	Peripherie und Hardware-Funktionen	267
7.2	Auswahl der Firmware	267
7.2.1	Offizielle MicroPython-Firmware	267
7.2.2	Erweiterte oder angepasste Firmware	268
7.2.3	Firmware-Features prüfen	269
7.3	MicroPython und das CPU Diagnostic Tool	270
7.4	Fazit	280
8	MicroPython-Anwendungen	283
8.1	Anwesenheitsdetektion zur Raumüberwachung	283
8.1.1	Ultraschallsensoren	283
8.1.2	ToF-Sensoren	289
8.1.3	Raumüberwachung mit PIR-Sensor	294
8.1.4	Sensorkombination zur Raumüberwachung	298
8.2	BLE-Sensor-Netzwerk mit BLE-MQTT-Gateway	299
8.2.1	Verwendete BLE-Sensorik	299
8.2.2	Eingesetzte Hardware	301
8.2.3	Anforderungen an das BLE-Sensor-Netzwerk	302
8.2.4	BLE-MQTT-Gateway-Anwendungsprogramm	302
8.3	Monitoring des Raumklimas	310
8.3.1	Sensoren für die Messung der Luftqualität	311
8.3.2	Eingesetzte Hardware	313
8.3.3	Anforderungen an die Messeinrichtung	315
8.3.4	Anwendungsprogramm zum Raum-Monitoring	316
8.3.5	Anwendungsprogramm zum Schlafrum-Monitoring	323
8.4	Intelligenter Regenalarm	330
8.4.1	Datenbasis und API-Aggregation via Open-Meteo	331
8.4.2	Algorithmische Trendanalyse mittels KI	331
8.4.3	Eingesetzte Hardware	331
8.4.4	Anforderungen an die Messeinrichtung	332
8.4.5	Anwendungsprogramm zum Regenalarm	335

8.5	Messwerterfassung und Visualisierung im Webserver	345
8.5.1	Verwendete Sensoren	346
8.5.2	Eingesetzte Hardware	346
8.5.3	Anforderungen an die Messeinrichtung	346
8.5.4	Anwendungsprogramm zur Messwerterfassung und Visualisierung	347
8.6	Visualisierung von Bilddaten im Webserver	355
8.6.1	Verwendete Sensoren	355
8.6.2	Eingesetzte Hardware	355
8.6.3	Anforderungen an die Messeinrichtung	357
8.6.4	Anwendungsprogramm zur Visualisierung im Webserver	357
8.7	Echtzeit-Audiovisualisierung	363
8.7.1	Verwendete Sensorik	363
8.7.2	Eingesetzte Hardware	363
8.7.3	Anforderungen an die Visualisierungseinrichtung	364
8.7.4	Anwendungsprogramm zur Visualisierung	365
8.8	Internetzugriff auf IoT-Geräte	372
8.8.1	Blynk im Überblick	372
8.8.2	NeoPixel-Steuerung	374
8.8.3	Klimasteuerung	378
8.8.4	Fazit zu Blynk	383
	Index	385

Index

@rp2.asm_pio(set_init=rp2.PIO. OUT_LOW)	243
/CS	143
0.96"-OLED-Display	214
12-bit-ADC ADS1015	139
1-Wire	149
<i>Bus</i>	150
<i>Bus Timing</i>	150
<i>Devices</i>	150
24-Bit-I ² S-Schnittstelle	363
32-Bit-Single-Precision-Floating-Point- Zahlen	78

A

Abklingverhalten durch envelope	368
Access Point (AP)	178
ACK	140
ADC	19, 135
add	83
Addition	87
Adressierungsschema (I2C)	139
Advertisement Data	306
Advertising-Datentypen	197
Advertising-Nachrichten	195
AES-128	188, 190
Alias (Module)	112
Analoge Ein- und Ausgabe	135
Anschlussleiste	21
Anwenderspezifische Funktionen	106
Anwesenheitsdetektion	283, 298
Anzeigeeinheiten	210
APDS9960	29
API-Aggregator	331
append	80
Arduino Lab for MicroPython	48
Arduino MicroPython IDE	73
Arduino MKR Zero	173
Arduino Nano 33 BLE Sense	28
<i>Anschlussbelegung</i>	29
Arduino Nano ESP32	30
<i>Anschlussbelegung</i>	31
Arduino Nano RP2040 Connect	29
<i>Anschlussbelegung</i>	30
Arduino Portenta H7	39
<i>Anschlussbelegung</i>	40
Arduino-Pro	39
AsyncIO	229, 233
asyncio	233

AT-Kommando	205
Atmel WINC1500 WiFi-Firmware-Update ...	356
AttributeError	101
Audiosignal	363
Audiovisualisierung	363
Ausgabereihenfolge	82
Auto-Gain	365
AutoNet	187

B

Badger2350	224
Banana Pi bpi:bit	212
BBC micro:bit	21
<i>Python Editor</i>	22
Beat Detection	365
Benchmark	
<i>micropython_mega_benchmark.py</i>	68
<i>micropython-pystone-lowmem.py</i>	67
<i>Resultate MicroPython Mega Benchmark</i>	68
<i>Resultate Pystone</i>	67
Benchmarks	66
Benutzerdefinierte Module	128
Beschleunigungssensor	29
Bilddaten	355
Bilderfassung	357
Bildsensor MT9M114	355
Bildungssysteme	20
Bildverarbeitung	248
Binäräquivalente	88
BLE	192
<i>Broadcast</i>	195
<i>Characteristic</i>	194
<i>Generic Access Profile (GAP)</i>	193
<i>Peripheral</i>	198
<i>Rollen im GAP</i>	193
<i>Services</i>	194
<i>UART</i>	201
BLE-MQTT-Gateway	299, 302
BLE-Sensor-Netzwerk	299
<i>Anforderungen</i>	302
<i>BLE-MQTT-Gateway</i>	301
Bluetooth Low Energy	192
Bluetooth SIG	194
Blynk	
<i>API</i>	373
<i>Architektur</i>	373
<i>GUI auf dem Smartphone</i>	379
<i>Klimasteuerung</i>	378

<i>Library</i>	373
<i>NeoPixel Control GUI</i>	375
<i>NeoPixel-Steuerung</i>	374
<i>Überblick</i>	372
BME680	312
BMI270	29
BMM150	29
BMP280	350, 355
bool	78
boot.py	167
BotFather	323
Bot-Token	323
break	97
Broadcasting Device	193
Brute-Force-Methode	272
Built-in-Funktionen	102

C

Callback-Funktion	157
Calliope mini	22
<i>Python Editor</i>	23
CAN-Bus	42
capitalize	85
Castellated-Modul	32
Central Device	193
chdir	168
Cheap Yellow Display	215
clear	80–81, 83
Cloud-Broker	372
C-Module	268
Codeanalyse	257
Codierungstechnik	123
Coding Rules	123
complex	78
const	123
Consumer-Task	232
continue	97
copy	83
count	80
CPHA	143
CPOL	143
CPU-Performance	266
CPython	257
credentials.py	128
CYD	215
<i>Custom Fonts</i>	216
<i>Grafikcontroller</i>	216
<i>Hardware-Revisionen</i>	220
<i>Touchcontroller</i>	216
CYW43439	32

D

DAC	19, 136
DAC-ADC-Charakteristik	136
Damien P. George	11
Dateisystem	167, 271
<i>Funktionen</i>	168
<i>Speicherinformation</i>	169
<i>Statusinformationen</i>	168
<i>Struktur des Flash-Speichers</i>	270
Datenlogger	171
Datentypen	78
DDNS	372
Debugging	257
<i>print()</i>	261
<i>REPL</i>	259
<i>Ruff</i>	257
<i>Thonny-Debugger</i>	262
<i>Tipps</i>	257
Deep Sleep	331
Dekorator	242
Deterministische Vorfilterung	344
DFU-Bootloader	26
dht	380
DHT11	378
Diagnostic-Resultate	
<i>Arduino Portenta</i>	277
<i>Pyboard v1.1</i>	280
<i>Raspberry Pi Pico 2 W</i>	278
<i>Speed-M1-Mikrocontroller</i>	277
<i>XIAO nRF52840 Sense</i>	279
dict	82
Dictionaries	81
difference	83
Digitale Ein- und Ausgabe	132
discard	83
Distance Measuring Transducer Sensor	284
Division	87
DMA	19
DS18B20	150–151
ds18x20	151
DS28E18	150
DS-Lite	372
DTMF	366
Duty	210
DX-CP35	300, 305
Dynamic DNS	372

E

Echtzeit-Audio-Visualisierung	363
Efento	307

Einfügereihenfolge	82
Eingaben in der Shell	73
E-Ink-Display	224
Einrückungen	76
Einsatz in der Schule	21
else	98
Emoji	79
endswith	86
Energiebedarf	265
<i>Peripherie</i>	265
Entwicklungsboards	24
Entwicklungsprozess	281
Entwicklungsumgebungen	43
Envelope	365
E-Paper	224
esp	273
esp32	273
ESP32 Atom Matrix	
<i>Anschlussbelegung</i>	38
ESP32-S3	30
ESP8266 NodeMCU	36
ESP-NOW	188
<i>Datenübertragung</i>	190
<i>LMK</i>	190
<i>PMK</i>	190
Espressif	188
Espressif ESP8266	36
esptool	57
Ethernet	184
Event Loop	353
except	98
Exception	98, 101
extend	80

F

Fading	213
Feldeffekt-Transistor (FET)	135
Fermion BLE Sensor Beacon	299
FFT	365
FileNotFoundError	101
finally	98
find	86
FireBeetle 2 ESP32-E	35
<i>Anschlussbelegung</i>	36
Firmware	
<i>Arduino Nano 33 BLE</i>	63
<i>Arduino Nano ESP32</i>	63
<i>Arduino Portenta H7</i>	61
<i>Features prüfen</i>	269
<i>FireBeetle</i>	58
<i>installieren</i>	51

<i>M5Stack Atom</i>	58
<i>MAiX BiT</i>	63
<i>micro:bit</i>	56
<i>mit kflash_gui</i>	63
<i>Nucleo-F446RE</i>	54
<i>OpenMV IDE</i>	65
<i>Pyboard</i>	52
<i>Raspberry Pi Pico</i>	59
<i>Seeed Studio XIAO RP2350</i>	60
<i>Teensy 4.0</i>	59
<i>vorkompilierte Builds</i>	268
FlashLogger	172
float	78
Floating-Point-Zahlen	89
for-Schleifen	93
framebuf	273
Frequenzanalyse	365
FSM	235
Funktionen	102
fw_update.py	356
fw_version.py	356

G

Ganzzahldivision	87
GAP	193
Garbage Collection	121, 124
GATT	193
gc	121, 273
gc.collect	125–126
gc.mem_alloc	126
gc.mem_free	126
Gemini	331
<i>API</i>	334
<i>Token-Verbrauch</i>	345
Generic Access Profile	193
Generic Attributes	193
get	81
getcwd	168
GIL	234
Global Interpreter Lock	234
Goertzel-Algorithmus	365
Goldfinger	21
Google AI Studio	334
Govee H5075	299, 305, 308
GPIO	19
Grayscale-Mode	355
Grove Wio-E5	206
Grove-Anschluss	23
Grove-System	23
Gyroskop	29

H

Hardware-Plattform	20
hashlib	273
HC-SR04	284
HC-SR501	294
Heap-Fragmentierung	121
Hintergrundbeleuchtung	218
HiveMQ	372
HMI-Modul	221
HS3003	29
HSV-Modell	213
HSV-zu-RGB-Konvertierung	214
HTML-Seite	181
<i>lokaler Webserver</i>	357
HTS221	28
HTTP-Fehler 503	340

I

I/O-Konfiguration	
<i>Pyboard</i>	133
I2C	19, 139
<i>Bausteinübersicht</i>	139
<i>Controller</i>	141
<i>Hardware</i>	142
<i>Master</i>	140
<i>Netzwerk</i>	139
<i>Slave</i>	140
<i>Software</i>	142
<i>Timing</i>	140
<i>Traget</i>	141
I2C-Check	141
I2C-Scan	141
IAQ	312
iBeacon DX-CP35	300
if ... elif ... else	92
Import	111
ImportError	101
IMU	29–30
in	82
IndentationError	101
index	80
IndexError	101
Indoor Air Quality Index	312
INMP441	363
InPlay Inc.	299
insert	80
int	78
Intelligenter Regenalarm	330
Internetzugriff auf IoT-Geräte	372
Interrupt-Handler	154

Interrupts	154
Interrupt-Service-Routine	154
intersection	83
IoT OnOff	372
IP-Reservierung	372
isalpha	86
isclose	89
isdigit	86
ISR	154
isspace	86
items	81

J

join	85
json	273
JST-Batterieanschluss	23

K

Kamera-Interface	41
Kamerasensor	41
KeyboardInterrupt	101
KeyError	101
keys	81
Key-Value-Paare	81
Kommentare	76
Kommunikation	176
Konstanten	77
Kontrollstrukturen	92
Konvertierung JPG in Raw	218

L

Lambda-Funktionen	110
LAN	184
LAN8720	184
Langzeitaufzeichnung	261
LCD	215
lcd160cr	273
Least Significant Bit	143
LED	210
len	80, 82, 84
Libraries	102
Light and Versatile Graphics Library	221
LilyGo TTGO T8 v1.8	173
listdir	168
Listen	79
Local Area Network	184
Lock	231, 234

Logfile	261
Logische Operatoren	89
LoRaWAN-Knoten	206
lower	85
LPS22HB	28–29
LSB	143
LSM303AGR	28
LSM6DSO	28
LSM6DSOXTR	30
Luftqualität	311, 314
LVGL	221

M

M5Stack	290
M5Stack Atom Matrix	37
M5Stack ENV II Unit	355
M5Stack OLED Unit	215
M5Stack PIR Motion Sensor	295
M5Stack ToF-Unit	290
M5Stack Unit ENV-II	313, 346
M5Stack Unit Mini TVOC/eCO2	313, 346
MAC-Adresse	188
machine	114, 273
MagJack	184–185
main.py	167
MAiX BiT	41, 252
<i>Anschlussbelegung</i>	42
<i>Firmware</i>	253
MAiXPy IDE	49
Manufacturer-Specific Data	304
Maskierung	92
max	80
Media Access Controller	184
meminfo.py	249
MEMS-Mikrofon	21, 29, 41, 363
Messwerterfassung	345
Microdot	346, 372
MicroPython	11
<i>Benchmarks</i>	66
<i>Besonderheiten</i>	131
<i>Firmware</i>	20
<i>Micropython</i>	111
<i>offizielle Firmware</i>	267
<i>Package Installer für Arduino</i>	48
<i>RAM-Bereiche</i>	121
<i>Sprachelemente</i>	73
<i>Website</i>	51
MicroPython Diagnostic Tool	270
MicroPython Installer	48
MicroPython Pyboard	24
micropython.mem_info	125–126
Mikrocontroller	
<i>Auswahlkriterien</i>	265
<i>ESP32-PICO-D4</i>	37
<i>ESP32-S3</i>	30
<i>ESP8266</i>	36
<i>ESP-WROOM-32E</i>	35
<i>Kendryte K210</i>	41
<i>Nordic nRF52840</i>	28
<i>NXP i.MX RT1062</i>	39
<i>Raspberry Pi RP2040</i>	29
<i>RP2040</i>	32
<i>RP2350</i>	33
<i>STM32F405RGT6</i>	25
<i>STM32F411RET6</i>	26
<i>STM32F446RE</i>	27
<i>STM32H743VI</i>	42
<i>STM32H747</i>	39
Mikrofon	363
min	80
MISO	143
Mittelwert	288, 292
mkdir	168
Modul	102, 111
ModuleNotFoundError	101
Modulo-Operation	88
MOSI	143
Most Significant Bit	143
MOX	311
MP34DT06	29
mpremote	44
MQTT	372
MQTT Dash	372
MQTT Terminal Free	307
MSB	143
Multiplikation	87
Multitasking	229, 233
<i>kooperativ</i>	233
Multithreading	229

N

NACK	140
NameError	101
Namenskonflikt	113
NanoBeacon IN100	299
NanoPy	24
NDIR	311
near_equal	89
NeoPixel	211–212, 349, 364
<i>Kaskadierung</i>	212
<i>Strombedarf</i>	212
neopixel	213

network	119
Nextion Editor	221
Nextion HMI	221
nice-Debugger	262
NOT	90
Notepad++	44
Nowcasting	331
nRF Connect	195, 321
Nucleo-F446RE	26
Nyquist-Theorem	366

O

Observer Device	193
OLED	214
Olimex ESP32-EVB-EA	185, 302
onewire	151
Online-UUID-Generator	194
OpenAI	325
Open-Drain-Ausgang	133
Open-Meteo	331
OpenMV Cam H7 R2	42, 248
OpenMV IDE	50
OpenMV WiFi-Shield	355
Operatoren	87
OR	90
os	116, 167, 273
OV2640	41
OV5640	41
OverflowError	101
Oxocard	24

P

Parallelisierung	231
pass	97
Payload	189, 210, 322
Peer-to-Peer-Kommunikation	188
PEP	15
Peripheral Device	193
Peripherals	19
PermissionError	101
PHY	184
Physical Layer	184
Pimoroni	224
PIO	239
PIR-Sensor	294
PoE	184
pop	80, 83
popitem	81
Portierung	265

Ports	20
Portweiterleitung	372
Potenz-Operation	88
Power Over Ethernet	184
Prellzei	155
print()-Debugging	261
Probabilistische Inferenz	333, 344
Producer-Task	232
Programmable I/O	239
Programmiersprachen	
<i>Popularität</i>	12
<i>Popularitätsranking</i>	12
Prompt	325
Pull-up-Widerstand	135
Push-Benachrichtigung	333
Pushover	341
Push-Pull-Ausgang	133
PWM	19, 136, 210
Pyboard	25, 52
<i>Anschlussbelegung</i>	26
Pyboard Lite	26
Pyboard v1.1	
<i>Anschlussbelegung</i>	134
PYPL	12
Pystone	67
Python	11
Python 3	13
Python 3 und MicroPython	
<i>Gemeinsamkeiten</i>	14
<i>Unterschiede</i>	14
Python Enhancement Proposal	15
Python-Module	268

R

RAM	19
Raspberry Pi Pico	32
<i>Anschlussbelegung</i>	33
Raspberry Pi Pico 2 W	33, 374, 378
<i>Anschlussbelegung</i>	34
Raumklimas	310
Raumsensor mit Display	301
Raumüberwachung	283, 294, 298
Real-Time Clock	331
Reconnect (MQTT)	310
Regenalarm	330
Remote-Monitoring-System	355
remove	80, 83, 168
rename	168
REPL	46, 259
replace	85
Retry	239

ReturnValue 106
 reverse 80
 RGB565-Mode 355
 RISC-V-Prozessor 33, 41
 rmdir 168
 rp2 273
 RP2350 374, 378
 RP2350 Environmental Webserver 354
 RS-232 147
 RTC-Speicher 331
 ruff 257, 276
 RuntimeError 101

S

SCD30 313
 Scheduler 157
 Schlafrum-Monitor 314
 Schnittstellen-Test 270
 SCK 143
 SCL 139
 SDA 139
 SD-Card 42, 172, 174
 SDIO vs. SPI 173
 SD-Karten 172
 SD-Karten-Slot 41
 Seeed Studio 23, 206
 select 273
 Sensirion 313
 Serial Bluetooth Terminal 204
 Serial Peripheral Interface 142
 Set 83
 setdefault 81
 SGP30 313
 shared Memory 231, 234
 SHT30 350, 355
 Smart Environmental Monitoring 299
 socket 273
 Software-Timer 160
 Sonar:bit 1.4 285
 Sonderzeichen 79
 sort 80
 sorted 80
 SPI 19, 142
 Betriebsmodi 143
 Datenaustausch 143
 Datenfluss 143
 Timing 144
 SPI-Loopback-Test 146
 SPI-Master 144
 SPI-Modi 144
 SPI-Slave 146

split 85
 SSID 178
 startswith 86
 stat 168
 State-Machines 235
 States 238
 Station (STA) 177
 Station (WiFi) 177
 statvfs 168
 STMicroelectronics 290
 str 78
 String 85
 dynamischer 123
 Methoden 85
 strip 85
 struct 273
 Subtraktion 87
 sum 80
 symmetric 83
 symmetric_difference 83
 Syntaxfehler 260
 sys 118

T

Teensy 4.0 38
 Anschlussbelegung 39
 Telegram-Ausgabe 329
 Telegram-Bot 323
 Texteditor 44
 Thonny 46, 73, 163
 Debugger 262
 Threads 229
 Threads vs. AsyncIO 234
 time 114, 273
 Timer 158
 Hardware 158
 One Shot 159
 periodisch 159
 Übersicht 161
 verfügbare 158
 virtueller 158
 TIOBE 12
 ToF-Sensor 289
 ToF-Unit 290
 Tonwahlsignale 366
 Trendanalyse 331
 Trendprognose 331
 try 98
 try ... except 98
 TTN Console 209
 TTN LNS 206

TTN-LoRaWAN-Netzwerkserver	206
Tupel	79
TypeError	101

U

UART	19, 146
UART vs. RS-232	147
uasyncio	233
ulab-Projekt	268
Ultraschallsensor	283
Umweltparameter	345
UND	90
union	83
Universal Asynchronous Receiver-Transmitter 146	
Universally Unique Identifier	194
update	81
uping	121
upper	85
ure	273
usocket	273
UTF-8	88
UTF-8-Codierung	79
utime	273
UUID	194

V

ValueError	101
values	81
Variablen	76
interne	77
Namensgebung	77
Vergleichsoperatoren	88
Viewport	182
Virtuelle Pins	373
VL53L0X	290

W

Watchdog	161
Waveshare	224
WDT	161
Reset	162
Web-GUI	349
WebSockets	347
Wemos D1 mini	37
Wetter-Apps	331
while-Schleifen	95
WiFi vs. WLAN	177
WINC1500-Firmware	356
Wio-E5 mini Dev Board	206
Wio-E5 Wireless Module	206
WS2812	211
RGB-LED	211

X

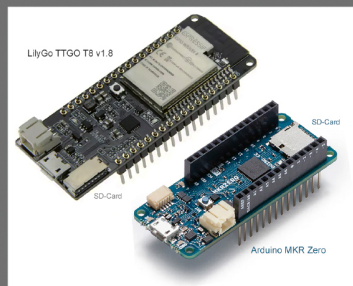
XIAO_C3	301, 304, 308
XIAO_C3 Raw Data	322
X-Nucleo-IKS01A2	26
XOR	90

Z

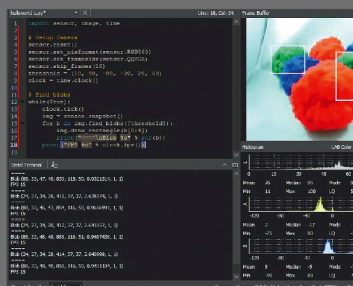
Zeichenketten	85
Zen of Python	16
Zen-Prinzipien	17
ZeroDivisionError	101
Zustandsmaschine	235
Zuweisungsoperation	91

Python für Maker

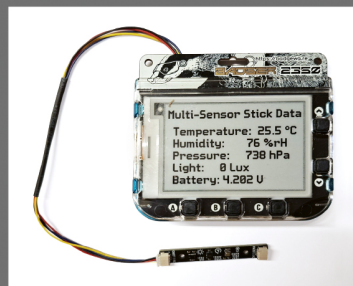
Mikrocontroller-Programmierung muss nicht schwierig und umständlich sein, denn mit MicroPython wird die Arbeit mit den Arduinos, dem ESP32 oder dem Raspberry Pi Pico einfach: Im Handumdrehen lesen Sie Sensoren aus, erstellen Webserver und versenden Daten über Bluetooth LE oder WLAN. Wie das geht, zeigt Ihnen Claus Kühnel.



Hardware auswählen



Verstehen & programmieren



Projekte umsetzen

Alles für den gründlichen Einstieg

Sie brauchen keine Vorkenntnisse, um mit MicroPython durchzustarten. Die Grundlagen sind schnell erklärt, und dann kann es schon mit den Entwicklungsboards aus der Arduino-Familie oder einem Board von ESP losgehen.

Für Mikrocontroller programmieren

Die Welt der Mikrocontroller unterscheidet sich von der klassischen Programmierung, denn hier wird um jedes Kilobyte RAM gekämpft. Lernen Sie das REPL-Schema kennen und machen Sie sich mit Best Practices und fortgeschrittenen Techniken vertraut.

Fortgeschrittene Projektideen

Von einer kleinen Wetterstation über den Aufbau eines BLE-Sensor-Netzwerks bis hin zu Experimenten mit LLMs: Beispiele zeigen Ihnen, wie Sie mit MicroPython Projekte umsetzen und selbst aktiv werden



Skripte und Projektbeispiele zum Download



Claus Kühnel ist E-Techniker und erfahrener Maker. Er hat Embedded Systems für die Labortechnik entwickelt und zahlreiche Fachbeiträge zu Mikrocontrollern geschrieben. In diesem Handbuch dreht sich alles um die Programmierung mit MicroPython.

Ihr Werkzeugkasten

Programmierung & Hardware

- Benchmarks, Hardware und IDEs für den Start
- Firmware installieren
- Sprachgrundlagen: Syntax, Operatoren, Strukturen
- Funktionen & Module
- Multitasking mit Threads und AsyncIO
- Debugging
- Best Practices für den guten Projektstart

Projekte

- Anwesenheit & Raumüberwachung
- BLE-Sensor-Netzwerk
- Raumklima messen
- Wettervorhersage mit KI
- Messwerte erfassen und visualisieren
- Echtzeit-Audiovisualisierung
- Internetzugriff über Blynk

