

Micropython mit ESP32

Einleitung	3
Lektion 1: Was dich in diesem Kurs erwartet	4
Musst du irgendwelche Voraussetzungen erfüllen?	4
Wie weit möchtest du in die Materie eintauchen?	4
Wie gehen wir vor?	4
Brauchst du exakt die vorgeschlagene Hardware?	5
Wo bekommst du Unterstützung?	5
Lektion 2: Die Hardware	6
Spezifikationen	6
Zur Verfügung stehende Pins	7
Pinout	8
Das Schema	9
Lektion 3: Die Software	10
Die Entwicklungsumgebung	10
Installation	10
Erste Schritte	11
Lektion 4: Digitale Ein- und Ausgänge, Delay	12
Digitale Ausgänge	12
Digitale Eingänge	12
sleep (delay) und ticks (millis)	13
Lektion 5: Das Display	14
Initialisierung	14
Direkte Befehle	14
Grafikbefehle	15
Textbefehle	15
Ausgabe an Zeichenposition	16
Lektion 6: Die Stoppuhr	17
Die Verdrahtung	17
Der Ablauf	17
Initialisierung der Komponenten	18
Eine Funktion zum Abfragen der Tasten	18
Was machen wir, wenn Start gedrückt wird?	19
Was machen wir, wenn die Uhr aktiv ist und wie reagieren wir auf Stopp?	19
Anzeige auf dem Display und die format() - Anweisung	19
Deine Übungsaufgabe	20
Lektion 7: Eine mögliche Lösung	21
Die Stoppuhr soll automatisch ausgeführt werden. Wie geht das?	21
Lektion 8: Druck, Temperatur und Luftfeuchtigkeit (BME280)	22
Überblick	22
Der Versuchsaufbau	22
Die Software	22
Lektion 9: Die Thonny-Version 3.2.3	24
Installation von Thonny	24
Aktualisieren der Micropython - Version auf dem Board	25
Neue Arbeitsweise mit Thonny 3.2.3	26
Der File - Browser	27

Lektion 10: Neue Heltec-Version

28

Lektion 11: Nochmals BME280

30

Einleitung

Dieses Handbuch begleitet den Videokurs **Micropython mit ESP32**. Die dazugehörigen Videos sind auf Youtube frei zugänglich. Ich gehe davon aus, dass du die jeweilige Lektion gesehen und nachvollzogen hast.

Mit jeder Lektion wächst das Handbuch um ein Kapitel. Die neuste Version findest du immer im Begleitmaterial zur Lektion. In diesem Begleitmaterial findest du auch alle Source - Codes der Experimente.

Support gibt es ausschliesslich über das Forum.

Auf <https://community.hobbyelektroniker.ch/wbb/index.php?board/40-die-lektionen-zu-micropython-mit-esp32/> findest du für jede Lektion einen eigenen Bereich in dem du Fragen stellen kannst.

Selbstverständlich kannst du dort auch auf Fragen anderer Zuschauer antworten.

Auf Micropython wird soweit eingegangen, wie es für das Verständnis der Experimente notwendig ist. Wenn du eine systematischere Einführung möchtest, empfehle ich dir den parallel laufenden Kurs **Micropython Grundlagen** zu verfolgen. Auch hier gibt es einen Support - Bereich im Forum: <https://community.hobbyelektroniker.ch/wbb/index.php?board/45-die-lektionen-zu-micropython-grundlagen/>

Es wird auch zwischendurch immer wieder einmal eine Übungsaufgabe geben. Versuche diese selbstständig zu lösen. Wenn du nicht weiterkommst, suche im Internet die notwendigen Informationen. Wenn das nichts hilft, dann stelle eine Frage im Forum und diskutiere das Problem mit anderen Kursteilnehmern. Erst ganz zum Schluss solltest du dir die Musterlösung anschauen. Diese Lösung ist immer nur ein Vorschlag. Vielleicht ist deine Lösung sogar besser.

Also, dann kann es los gehen. Ich wünsche dir viel Spass!

Lektion 1: Was dich in diesem Kurs erwartet

Wir werden und mit einem etwas weiterentwickelten Mikrokontroller beschäftigen, dem ESP 32. Anders als im Arduino werden wir nicht in der Sprache C oder C++ programmieren, sondern in Micropython. Micropython ist ein auf Mikrokontroller angepasstes Python 3. Du wirst also die Sprache Micropython lernen.

Musst du irgendwelche Voraussetzungen erfüllen?

Der Kurs ist nicht für absolute Mikrokontroller - Neulinge oder Programmieranfänger geeignet. Wenn du aber schon eine Led an einem Arduino zum Leuchten gebracht hast und den Unterschied von Strom und Spannung kennst, solltest du den Kurs mitverfolgen können. Programmierseitig solltest du wissen, was Variablen sind und wozu Funktionen gut sind. Der Arduinokurs auf meinem Youtube - Kanal ist eine sehr gute Voraussetzung. Python oder ESP32 - Kenntnisse sind nicht notwendig. Ein beliebiger Mikrokontroller und eine beliebige Programmiersprache bilden eine gute Grundlage.

Wie weit möchtest du in die Materie eintauchen?

Am meisten lernst du natürlich, wenn du alles was ich zeige nachvollziehst und nicht locker lässt, bevor du es vollständig verstanden hast. Dabei wirst du durch gelegentliche Übungsaufgaben unterstützt. Du wirst dann danach in der Lage sein selbstständig ganz andere Projekte zu realisieren.

Teilweise gehe ich aber auf Details ein, die dich vielleicht gar nicht so interessieren. Wir sind hier nicht in der Schule, du kannst also auch Dinge auslassen. Da alle verwendeten Programme zum Download bereit stehen, kannst du sie auch benutzen, ohne deren innere Arbeitsweise vollständig zu verstehen.

Dieser Kurs ist auf Praxis ausgelegt. Wir werden Experimente aufbauen und dabei jeweils die dazugehörigen Befehle von Micropython besprechen. Das wird genügen, falls du schon Kenntnisse von Micropython hast oder gar nicht allzu tief in die Details eintauchen möchtest. Für alle Anderen wird parallel dazu auch der Kurs **Micropython Grundlagen** angeboten. Hier liegt dann der Schwerpunkt auf der Programmierung.

Wie gehen wir vor?

Alles beginnt mit der Beschaffung der Hardware und der Installation der notwendigen Software. Ich zeige dir die Installation für Mac und Windows. Unter Linux sollte es mehr oder weniger identisch sein.

Danach gibt es einige Grundlagen zum Thema Micropython. Dabei machen wir auch erste Experimente mit unserem ESP32.

Bald schon wird es aber konkret. Wir starten mit dem Projekt 'Wetterstation', das uns den ganzen Kurs begleiten wird. Du lernst, wie die verschiedenen Sensoren abfragt und die gemessenen Werte ausgegeben werden. Die Ausgabe wird zuerst auf die Konsole, später aber auf ein kleines Display erfolgen.

Da der ESP32 WLAN-tauglich ist, werden wir auch eine Ausgabe über einen Webserver erstellen. Damit kannst du mit jedem Computer, jedem Handy oder Tablet im WLAN deine Wetterdaten über den Browser abfragen.

Was ist schon heutzutage eine Wetterstation ohne Wettervorhersagen? Also werden wir über das Internet Wettervorhersagen beziehen und diese ebenfalls darstellen. Das ist aber noch nicht alles, einige Ideen werden sicher noch dazukommen.

Mit jedem Schritt wirst du neue Sprachelemente von Micropython kennenlernen, so dass du am Schluss in der Lage sein wirst, eigene Ideen zu verwirklichen.

Brauchst du exakt die vorgeschlagene Hardware?

Nein, eigentlich nicht. Das hängt aber von deinen Vorkenntnissen ab. Es gibt sehr viele geeignete ESP32 - Module. Wenn du in die notwendige Dokumentation beschaffen kannst und sie auch verstehst, dann ist das kein Problem. Du musst in der Lage sein andere Pins zu verwenden und natürlich dadurch auch die Verdrahtung und die Programme anzupassen. Wenn du hier unsicher bist, dann verwende einfach die vorgeschlagene Hardware.

Bei vollständig anderen Sensoren ist allerdings Vorsicht angebracht. Oft werden diese auf ganz andere Art programmiert und du bist dann auf dich allein gestellt.

Wo bekommst du Unterstützung?

Im Forum selbstverständlich. Die normalen Kommentare auf Youtube eignen sich nicht für ausführlich Antworten auf Fragen.

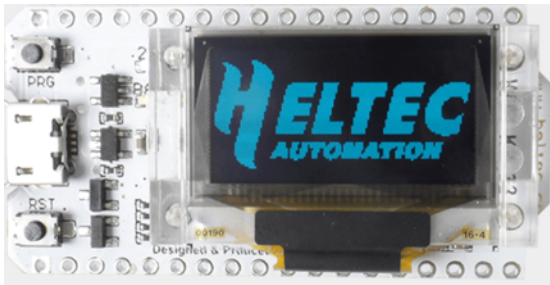
<https://community.hobbyelektroniker.ch/wbb/index.php?board/40-die-lektionen-zu-micropython-mit-esp32/>

und

<https://community.hobbyelektroniker.ch/wbb/index.php?board/45-die-lektionen-zu-micropython-grundlagen/>

Zu jeder Lektion wird es einen Bereich geben, in dem Fragen beantwortet werden. Auch deine Antworten sind selbstverständlich immer sehr erwünscht.

Lektion 2: Die Hardware



Heltec WiFi Kit 32

<https://heltec.org/project/wifi-kit-32/>

Ich empfehle dir eine Bestellung direkt beim Hersteller, Bangood oder Ali-Express. Mit 2 - 3 Wochen Lieferzeit musst du aber rechnen. Bestelle mindestens 2 Stück. So hast du ein Board zum Experimentieren und ein zweites zum fixen Einbau in deine Wetterstation.

Das Board bekommst du vielleicht auch über Amazon oder Ebay. Achte darauf, dass du nicht die LoRa - Variante bestellst. Auf diesem Board sind viele Pins bereits belegt.

Sensor BME280 mit I2C (SCL, SDA) für Luftdruck, Feuchte und Temperatur

Die sind sehr verbreitet, du solltest sie an vielen Orten bekommen. Wichtig ist einfach, dass es sich um eine I2C - Variante handelt.

Ich werde auch den **Sensor BME680** ausprobieren (<https://www.aliexpress.com/item/32902672818.html>), dieser wird aber nicht Voraussetzung für den Bau der Wetterstation sein.

Ich werde auch Experimente mit **anderen Displays** machen. Welche das genau sind, steht noch nicht fest. Dazu werde ich in meiner Bastelkiste graben und hoffen, dass ich etwas Passendes finde. Ob es dann in der Wetterstation verwendet wird, entscheidet sich erst später. Die vollständige Anzeige wird sowieso nur über einen Browser zu sehen sein.

Der Rest (Steckbrett, Anschlussdrähte, USB - Kabel usw.) sollte ja schon in deiner Bastelkiste vorhanden sein.

Andere ESP32 - Board können verwendet werden, wenn du damit zurecht kommst. Eine andere Pin - Belegung wird dann von dir immer Anpassungen an der Verdrahtung und dem Programm verlangen. Sorge auf jeden Fall dafür, dass du vom Hersteller oder Lieferanten die entsprechenden Informationen bekommst.

Beim Sensor BME280 empfehle ich dir dringen, dich daran zu halten. Sonst wirst du auf dich alleine gestellt sein.

Spezifikationen

- ESP32 (240MHz Tensilica LX6 dual-core + 1 ULP, 600 DMIPS)
- 520KB SRAM
- Wi-Fi, dual mode Bluetooth
- 3 x UART, 2 x SPI, 2 x I2C, 1 x I2S
- 29 x general GPIO
- 2 x 12 bit ADC, an 18 Pins
- 2 x 8 bit DAC
- 4 MB (32 MBits) Flash
- 1 x Micro - USB
- USB to UART: CP2102 (Treiber von silabs.com)
- Battery connector: 3.7V Lithium
- Display: OLED 0.96 inch, 128 x 64

Zur Verfügung stehende Pins

Diese Pins werden beziehen sich auf das Heltec - Board. Falls du ein anderes Board verwendest, musst du die Liste anpassen.

Achtung: Alle Pins nur 3.3V, sie sind nicht 5V tolerant !!!

GPIO 0	Eingebauter Button	
GPIO 2	frei	
GPIO 4	OLED	SDA
GPIO 5	frei	
GPIO 12	frei	
GPIO 13	frei	
GPIO 14	frei	
GPIO 15	OLED	SCL
GPIO 16	OLED	muss HIGH gesetzt werden
GPIO 17	frei	
GPIO 18	frei	SCK (SPI)
GPIO 19	frei	MISO (SPI)
GPIO 21	frei	SDA (I2C)
GPIO 22	frei	SCL (I2C)
GPIO 23	frei	MOSI (SPI)
GPIO 25	Eingebaute LED	
GPIO 26	frei	
GPIO 27	frei	
GPI 34	frei	nur Input, kein PULL_UP/PULL_DOWN
GPI 35	frei	nur Input, kein PULL_UP/PULL_DOWN
GPI 36	???	nur Input, kein PULL_UP/PULL_DOWN
GPI 37	???	nur Input, kein PULL_UP/PULL_DOWN
GPI 38	???	nur Input, kein PULL_UP/PULL_DOWN
GPI 39	???	nur Input, kein PULL_UP/PULL_DOWN



Lektion 3: Die Software

Die Entwicklungsumgebung

Wir werden mit der Thonny - IDE arbeiten. Sie ist sehr einfach aufgebaut und lässt sich leicht installieren. Falls du die entsprechenden Kenntnisse hast, kannst du selbstverständlich auch grosse Entwicklungsumgebungen wie PlatformIO oder PyCharm verwenden. Das ist aber gar nicht so einfach, wie es auf den ersten Blick aussieht.

Ich verwende im Kurs Thonny (<https://thonny.org/>) und zwar in der **Version 3.1.2**. Zum Zeitpunkt des Kursstartes war die Version 3.2.0 im Beta - Stadium und noch nicht stabil genug. Das ist schade, da die Version 3.2 einige interessante Neuerungen kennt. Einige Funktionen fallen aber weg, so dass sich die Arbeitsweise ändern wird.

Die Version 3.1.2 kann für Windows und Mac hier heruntergeladen werden:

Windows: <https://github.com/thonny/thonny/releases/download/v3.1.2/thonny-3.1.2.exe>

Mac: <https://github.com/thonny/thonny/releases/download/v3.1.2/thonny-3.1.2.dmg>

Weitere Informationen auf Github: <https://github.com/thonny/thonny/releases/tag/v3.1.2>

Installation

1. Treiber für USB-Port herunterladen und installieren:

<https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>

2. Download Micropython für ESP32:

<https://micropython.org/download>

3. Thonny herunterladen und installieren:

<https://thonny.org> oder die Links im vorherigen Abschnitt.

4. ESP32 - Board anschliessen

5. Thonny starten

6. Tools / Manage plug-ins...

- Suchen nach esptool, installieren
- Suchen nach thonny-esp, installieren

7. Thonny beenden und neu starten

8. Preferences / Interpreter

- MicroPython on ESP32
- Port: CP2102

9. Device / Erase ESP8266/ESP32 flash ausführen

10. Device / Install Micropython to ESP8266/ESP32 ausführen

11. Run / Stop/Restart backend ausführen

Micropython sollte sich jetzt melden.



```
Shell x Exception x
MicroPython v1.11-146-g154062d9c on 2019-07-12; ESP32 module with ESP32
Type "help()" for more information. [backend=ESP32]
>>>
```

Erste Schritte

Jetzt wollen wir aber wissen, ob es funktioniert. Also schreiben wir unser erstes Python - Programm und führen es aus. Das Programm muss noch näher betrachtet werden. Das ist dann aber ein Thema für die nächste Lektion.

blink.py

```
# Ein einfaches Blink - Programm
import time
from machine import Pin

interne_led = Pin(25, Pin.OUT)

print("blink.py gestartet")
while True:
    interne_led.on()
    print("ein")
    time.sleep_ms(500) # 500 ms
    interne_led.off()
    print('aus') # auch diese Schreibweise ist möglich
    time.sleep(0.5) # 0.5 Sekunden
```

Dieses Programm läuft in einer Endlosschleife und muss abgebrochen werden! Der Abbruch kann mit **Stop/Restart backend** oder mit **ctrl d** durchgeführt werden.

[illegible]

Lektion 4: Digitale Ein- und Ausgänge, Delay

Ein digitaler Pin muss erstellt werden. Dazu wird eine Variable angelegt, die dann dem Pin entspricht. Es handelt sich dabei nicht nur um eine Pinnummer, sondern um ein komplettes Objekt, welches alle Aspekte seines Pins kennt.

Um Pins zu verwenden, müssen wir die Klasse **Pin** aus dem Modul **machine** importieren.

```
from machine import Pin
```

Digitale Ausgänge

Initialisierung:

```
led_pin = Pin(25, Pin.OUTPUT)
```

25 ist hier die GPIO - Nummer. Das entspricht oft nicht der Pinnummer an deinem Board.

Bei der Initialisierung kann auch noch ein Startwert angegeben werden:

```
led_pin = Pin(2, Pin.OUT, value = 1)
```

Setzen des Pins (er gibt dann 3.3V aus):

```
led_pin.on() oder led_pin.value(1)
```

Zurücksetzen des Pins (er gibt dann 0V aus):

```
led_pin.off() oder led_pin.value(0)
```

Abfragen des Pins:

```
status = led_pin.value()
```

Ist der Pin eingeschaltet, wird 1 zurückgegeben, sonst ist der Wert 0.

Zustand wechseln (blinken):

```
led_pin.value(not led_pin.value())
```

Digitale Eingänge

Initialisierung:

```
button_pin = Pin(2, Pin.IN)
```

Wie beim Arduino kann man auch hier einen internen Pull-Up - Widerstand zuschalten:

```
button_pin = Pin(2, Pin.IN, Pin.PULL_UP)
```

Der ESP32 stellt auch einen Pull-Down Widerstand zur Verfügung:

```
button_pin = Pin(2, Pin.IN, Pin.PULL_DOWN)
```

Es scheint aber so, dass diese Option standardmässig aktiviert ist, wenn nichts angegeben wird. Darauf sollte man sich aber nicht verlassen. Falls kein externer Widerstand verbaut wird, immer PULL_UP oder PULL_DOWN angeben!

Ohne Widerstand kann ebenfalls gearbeitet werden:

```
button_pin = Pin(2, Pin.IN, Pin.PULL_HOLD)  
button_pin = Pin(2, Pin.IN, Pin.OPEN_DRAIN)
```


sleep (delay) und ticks (millis)

Im Blink - Beispiel der vorangegangenen Lektion haben wir einen Delay kennengelernt. Dieser ist im Modul **time** enthalten und muss zuerst importiert werden.

```
import time
```

Wir behandeln hier nur die Delay-Funktionen von **time**. **time** kann noch wesentlich mehr, das soll aber ein Thema für eine spätere Lektion sein.

Verzögerung in Sekunden:

```
time.sleep(2.5)    # 2.5 Sekunden Pause
```

Verzögerung in Millisekunden:

```
time.sleep_ms(2500) # 2500 Millisekunden Pause
```

Verzögerung in Mikrosekunden:

```
time.sleep_us(2500) # 25 Mikrosekunden Pause
```

Diese Funktionen arbeiten wie delay() im Arduino. Sie sind also auch blockierend.

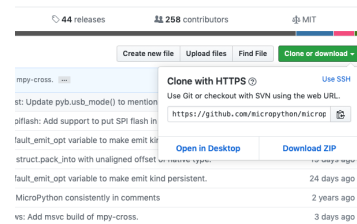
Es stehen auch Funktionen zur Verfügung, die wie millis() im Arduino arbeiten:

```
time.ticks_ms()  
time.ticks_us()
```

Lektion 5: Das Display

Micropython bringt standardmässig bereits eine Vielzahl von Bibliotheken mit. Diese Module können mit **import** dem Programm verfügbar gemacht werden.

Zusätzlich gibt es aber auch viele externe Module, die zusätzlich geladen werden können. Ein ganzes Bündel davon findest du auf Github (<https://github.com/micropython/micropython>). Lade einfach das ganze Zip - File herunter und entpacke es auf deinem Computer. Heute interessieren wir uns für den OLED - Treiber. Du findest ihn unter **drivers / display** als Datei **ssd1306.py**.



Initialisierung

Zuerst muss das Modul **ssd1306.py** auf das Board kopiert werden! Die Angaben gelten für das Heltec - Board. Bei anderer Hardware müssen möglicherweise die Pinnummern für **scl** und **sda** angepasst werden. Dasselbe gilt für **pin16**, bei einigen Boards kann auch auf ihn verzichtet werden.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c = I2C(scl=Pin(15), sda=Pin(4))
pin16 = Pin(16, Pin.OUT)
pin16.on()
oled = SSD1306_I2C(128, 64, i2c)
```

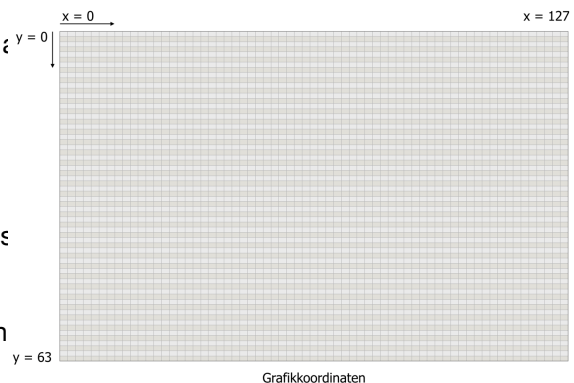
Damit ist das Display bereit und kann über die Variable **oled** : $y = 0$

Zuerst wird eine Schnittstelle benötigt:

```
i2c = I2C(scl=Pin(15), sda=Pin(4))
```

Diese Schnittstelle muss dem Konstruktor der Display - Klasse **SSD1306_I2C** (with, height, i2c, addr=0x3C,

Die beiden hintersten Parameter werden nur übergeben, wenn



Direkte Befehle

Nur wenige Befehle wirken sich direkt auf das Display aus. Normalerweise wird der Inhalt zuerst in einen internen Speicher geschrieben und erst mit dem Befehl **show()** angezeigt.

oled.poweroff()

Schaltet die Anzeige aus. Das Modul bleibt aber weiterhin aktiv. Es können Daten zum Display gesendet werden, diese werden aber erst sichtbar, wenn **poweron()** aufgerufen wird.

oled.poweron()

Die Anzeige wird wieder sichtbar.

oled.contrast(contrast)

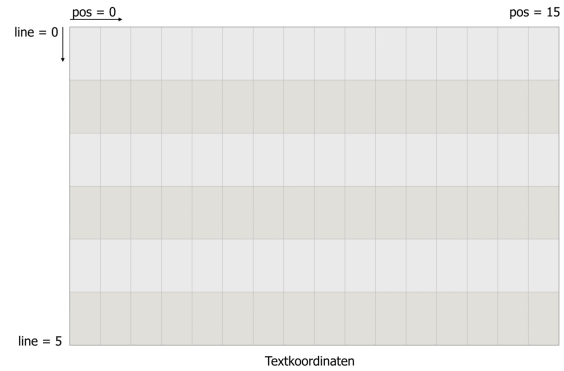
Es kann ein Kontrast zwischen 0 und 255 eingestellt werden mit 255.

oled.invert(invert)

Der Inhalt kann invertiert dargestellt werden. Der Parameter 0 gilt als **False**, jeder andere Wert wird als **True** interpretiert.

oled.show()

Der aktuelle Displayinhalt wird angezeigt.



Grafikbefehle

Es handelt sich hier grundsätzlich um ein Grafikdisplay mit den zwei Farben schwarz und hell. Diese werden als 0 und 1 übergeben. Die Auswirkungen der Befehle werden erst mit **show()** sichtbar. Normalerweise werden diese mit einem Parameter namens **c** übergeben.

Die x - Koordinaten liegen zwischen 0 und 127, die y - Koordinaten zwischen 0 und 63.

Es werden hier nicht alle Befehle aufgelistet. Es sind alle Befehle der Klasse FrameBuffer verfügbar. Eine vollständige Referenz ist unter <https://docs.micropython.org/en/latest/library/framebuf.html> zu finden.

oled.fill(c)

Der ganze Bildschirm wird mit der angegebenen Farbe gefüllt. **fill(0)** kann daher gut zum Löschen des Bildschirms verwendet werden.

oled.pixel(x, y, c)

Setzen oder Löschen eines einzelnen Pixels. Bei unserem Display löscht die Farbe 0 den Pixel, jeder andere Wert setzt ihn.

oled.pixel(x, y)

Gibt die Farbe des Pixels zurück (0 oder 1).

oled.hline(x, y, w, c)

Zeichnet eine horizontale Gerade mit der Länge **w**.

oled.vline(x, y, h, c)

Zeichnet eine vertikale Gerade mit der Höhe **h**.

oled.line(x1, y1, x2, y2, c)

Zeichnet eine beliebige Gerade von x1/y1 zu x2/y2.

oled.rect(x, y, w, h, c)

Zeichnet einen rechteckigen Rahmen mit Breite **w** und Höhe **h**.

oled.fill_rect(x, y, w, h, c)

Zeichnet ein ausgefülltes Rechteck mit Breite **w** und Höhe **h**.

Textbefehle

Es gibt nur einen einzigen Text - Befehl. Dieser ist in der Lage einen Text an einer beliebigen Stelle auszugeben. Der Text überlagert den bestehenden Inhalt, es wird also der darunter liegende Text nicht gelöscht.

oled.text(s, x, y, c)

Schreibt den Text in **s** an die Grafikkoordinaten **x** und **y**.

Ausgabe an Zeichenposition

Dazu gibt es keinen passenden Befehl.

Es wird daher folgende Funktion vorgeschlagen:

```
def text_line(text, line, pos = 0):  
    x = 10 * pos;  
    y = (line) * 11  
    oled.text(text,x,y)
```

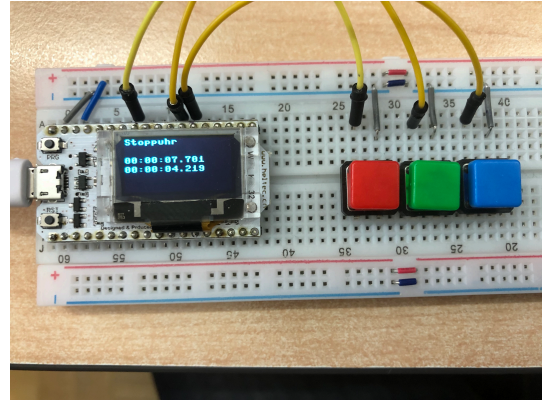
line ist die Zeilennummer (0 .. 5) und **pos** die horizontale Zeichenposition (0 .. 15). Wenn **pos** nicht mitgegeben wird, beginnt der Text am Zeilenanfang.

Lektion 6: Die Stoppuhr

Heute bauen wir eine kleine Stoppuhr.
Sie wird über drei Tasten gesteuert:

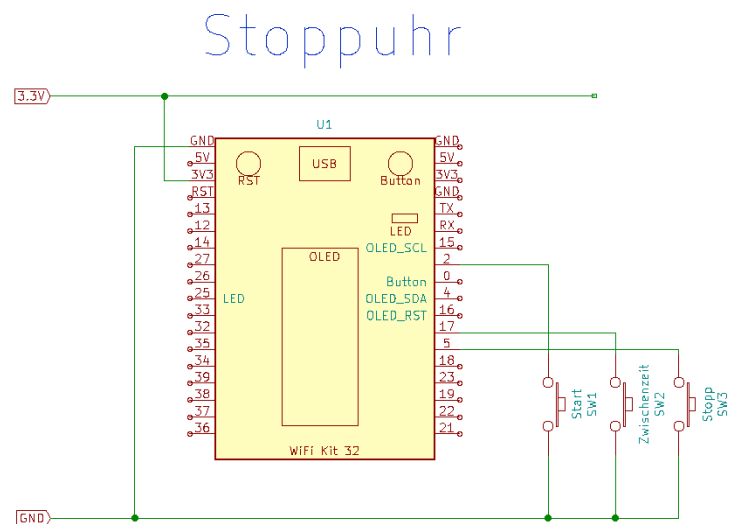
- Start
- Zwischenzeit
- Stopp

Wir müssen eine sinnvolle Verdrahtung erstellen und die Stoppuhr dann programmieren. Dazu benötigen wir das Wissen aus den vorherigen Lektionen, sowie weitere Informationen, die wir in dieser Lektion erarbeiten.



Die Verdrahtung

Für die 3 Taster benötigen wir drei freie Eingabepins. Das führt dann zu diesem Schema:



Der Ablauf

Oft ist es sinnvoll, einen ungefähren Ablauf des Programmes festzulegen.

- Zuerst fragen wir die Tasten ab
- Falls die Stoppuhr noch nicht aktiv ist, reagieren wir nur auf die Start-Taste. Wird sie gedrückt, starten wir die Uhr und merken uns den Startzeitpunkt.
- Falls die Stoppuhr aktiv ist, berechnen wir die aktuell vergangene Zeit. Wenn die Zwischenzeittaste gedrückt wird, speichern wir diese. Die Stopp-Taste hält die Uhr an.
- Am Schluss zeigen wir den aktuellen Stand an.

Wir müssen also den aktiv-Zustand und die Zeiten speichern. Dazu legen wir vier globale Variablen an:

```
start_zeit = 0
total_zeit = 0
zwischen_zeit = 0
aktiv = False
```

Initialisierung der Komponenten

Um es einfach zu halten, betrachten wir momentan nur die zwei Tasten **Start** und **Stopp**.

```
start_taste = Pin(2, Pin.IN, Pin.PULL_UP)
stopp_taste = Pin(5, Pin.IN, Pin.PULL_UP)
```

Wir verwenden bei den Eingängen die internen PULL_UP - Widerstände.

Den Code für das Display können wir aus einer früheren Lektion übernehmen.

Eine Funktion zum Abfragen der Tasten

Wir definieren die Funktion **tasten_abfragen()**:

Eine Funktion wird immer mit dem Schlüsselwort **def** eingeleitet. Es gibt keinen Unterschied zwischen Funktionen, die etwas zurückgeben und solchen, die nur einige Befehle ausführen.

Der Funktionskopf lautet also

```
def tasten_abfragen():
```

- **def** leitet die Funktionsdefinition ein
- **tasten_abfragen** ist der Name der Funktion
- Zwischen den beiden Klammern **()** könnte man Werte an die Funktion übergeben. In unserem Fall haben wir keine Parameter, deshalb bleiben nur die beiden Klammern übrig.
- Der Doppelpunkt **:** signalisiert, dass jetzt der Code der Funktion beginnt.

Danach folgen die Befehle, die von der Funktion ausgeführt werden sollen.
Alle Befehle, die zur Funktion gehören, sind um 4 Zeichen eingerückt.

```
    start_gedrueckt = start_taste.value()
    stopp_gedrueckt = stopp_taste.value()
    return start_gedrueckt
```

Mit **return** kann ein Wert zurückgegeben werden. Was machen wir aber, wenn wir beide Werte zurückgeben sollen? Das ist in Python ganz einfach! Wir geben einfach beide Werte zurück.

```
    return start_gedrueckt, stopp_gedrueckt
```

Wenn wir das ausprobieren, werden wir feststellen, dass es nicht wie gewünscht funktioniert. Die Taste wird als gedrückt zurückgegeben, wenn wir sie nicht drücken und umgekehrt. Das liegt an der Verdrahtung und am PULL_UP. Wir müssen also den Zustand umkehren. Das erfolgt mit dem Schlüsselwort **not**.

```
    return not start_gedrueckt, not stopp_gedrueckt
```

Es ist nicht notwendig, die Zustände in eigenen Variablen zu speichern. So funktioniert auch die verkürzte Version:

```
def tasten_abfragen():
    return not start_taste.value(), not stopp_taste.value()
```

Unten im Hauptprogramm nehmen wir beide Rückgabewerte gleichzeitig in Empfang und speichern sie in den Variablen **start** und **stopp**:

```
start, stopp = tasten_abfragen()
```

Was machen wir, wenn Start gedrückt wird?

Auf die Start - Taste reagieren wir nur, wenn die Uhr nicht aktiv ist. Das müssen wir testen:

if not aktiv:

Auch hier haben wir wieder diesen Doppelpunkt und die anschliessende Einrückung. Damit definieren wir, welcher Code ausgeführt werden soll, wenn die Bedingung erfüllt ist. Darunter testen wir noch auf die gedrückte Starttaste. Nur dann wird die Uhr gestartet.

```
if not aktiv:
    if start: # Start ist gedrückt
        start_zeit = time.ticks_ms()
        aktiv = True
```

Was machen wir, wenn die Uhr aktiv ist und wie reagieren wir auf Stopp?

```
if aktiv:
    total_zeit = time.ticks_ms() - start_zeit
    if stopp: # Uhr anhalten
        aktiv = False
```

Wir testen zuerst auf **aktiv**. Nur dann speichern wir die aktuell vergangene Zeit in **total_zeit**. Diese wird berechnet aus der aktuellen Zeit minus der Startzeit.

Dann wird noch auf die Stopptaste geprüft. Wenn sie gedrückt ist, halten wir die Uhr an.

Anzeige auf dem Display und die format() - Anweisung

Wir rufen einfach **anzeigen()** auf.

Leider haben wir da ein kleines Problem. **anzeigen()** gibt es noch gar nicht. Wir müssen diese Funktion also schreiben und dies ist eine anspruchsvolle Arbeit.

Der Anfang ist ganz einfach. Wir löschen den Bildschirm, schreiben die Texte und zeigen alles am Schluss mit **.show()** an.

```
def anzeigen():
    oled.fill(0) # Bildschirm löschen
    text_line("Stoppuhr",0)
    oled.show() # Alles anzeigen
```

Das Wichtigste fehlt hier aber. Wir wollen ja die vergangene Zeit anzeigen. Diese steht in der Variablen **total_zeit**. Es wäre nicht sehr hilfreich, wenn wir einfach diese Zeit in Millisekunden anzeigen würden. Wir müssen sie also in Stunden, Minuten, Sekunden und Tausendstel aufteilen.

Dazu brauchen wir die numerischen Operatoren (sie wurden bereits im Kurs [Micropython Grundlagen](#) besprochen).

```
millis = zeit % 1000
sekunden = (zeit // 1000) % 60
minuten = (zeit // 1000 // 60) % 60
stunden = (zeit // 1000 // 60 // 60) % 24
```

% ist der Modulo-Operator, er gibt uns den Rest einer ganzzahligen Division zurück

// ist der Operator für eine ganzzahlige Division

Jetzt müssen wir diese Werte nur noch in einen Text verwandeln.

Da wir später diese Umwandlung von **total_zeit** zu unserem Ausgabetext sowohl für die Gesamtzeit als auch für die Zwischenzeit brauchen, lagern wir sie in eine Funktion aus.

Diese Funktion soll uns einen String (ein Text in Form einer Zeichenkette) zurückgeben. HH:MM:SS.TTT soll das Format sein.

Dazu gibt es die Format-Funktion. Wenn ich schreibe

```
"{:}:{:}:{:}.".format(stunden, minuten, sekunden, millis)
```

werden die Platzhalter {} mit den entsprechenden Werten gefüllt. Jetzt können aber diese Werte verschiedene Anzahl Stellen aufweisen. Das ist unschön, lässt sich aber mit zusätzlichen Format - Anweisungen verbessern.

```
def zeit_text(zeit):  
    millis = zeit % 1000  
    sekunden = (zeit // 1000) % 60  
    minuten = (zeit // 1000 // 60) % 60  
    stunden = (zeit // 1000 // 60 // 60) % 24  
    return "{:02d}:{:02d}:{:02d}.{:03d}".format(stunden,minuten,sekunden,millis)
```

Unsere Funktion **anzeigen()** sieht dann so aus:

```
def anzeigen():  
    oled.fill(0) # Bildschirm löschen  
    text_line("Stoppuhr",0)  
    total_text = zeit_text(total_zeit)  
    text_line(total_text,2)  
    oled.show()
```

Deine Übungsaufgabe

Jetzt bist du dran. Das Projekt erfüllt ja noch nicht alle Kriterien, da wir noch keine Zwischenzeit haben. Das zu programmieren, ist deine Aufgabe. Ich wünsche dir viel Spass.

Eine mögliche Lösung werde ich in der nächsten Lektion zeigen.

Unsere Stoppuhr hat noch diverse kleine Schönheitsfehler, der die genaue Zeitmessung erschwert. Hast du eine Idee, welche das sind? Auch das werden wir in der nächsten Lektion behandeln.

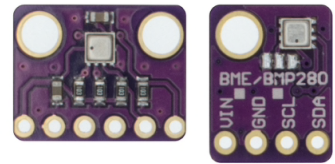
Lektion 7: Eine mögliche Lösung

Bitte schau dir das Video an und experimentiere mit den in den Begleitunterlagen vorhandenen Dateien.

Die Stoppuhr soll automatisch ausgeführt werden. Wie geht das?

Wenn das Board neu gestartet wird, wird automatisch zuerst `boot.py` und dann `main.py` ausgeführt. `boot.py` ist zwar auf dem Board vorhanden, enthält aber nur auskommentierte Zeilen. `main.py` musst du selber auf das Board kopieren.

Du kannst dein Stoppuhrprogramm mit dem Menüpunkt **Device / Upload current script as main script** auf das Board kopieren. Das gilt aber nur für Thonny bis zur Version 3.1.2. Ab der Version 3.2.1 gibt es diesen Punkt nicht mehr. Du kannst aber unter **File / save as..** das Programm als `main.py` auf das Board speichern.



Lektion 8: Druck, Temperatur und Luftfeuchtigkeit (BME280)

Überblick

Beim BME280 handelt es sich um einen Sensor von Bosch, der die gewünschten Werte liefern kann. Es gibt zwei Versionen von Breakout-Boards. Die eine ist für 5V ausgelegt, die andere für 3.3V.

Die 5V - Variante darf nur mit 3.3V betrieben werden, da sonst der ESP32 Schaden nehmen könnte.

Hier die wichtigsten Daten:

Schnittstellen: I2C und SPI
Betriebsspannung Sensor: 1.71 - 3.6V

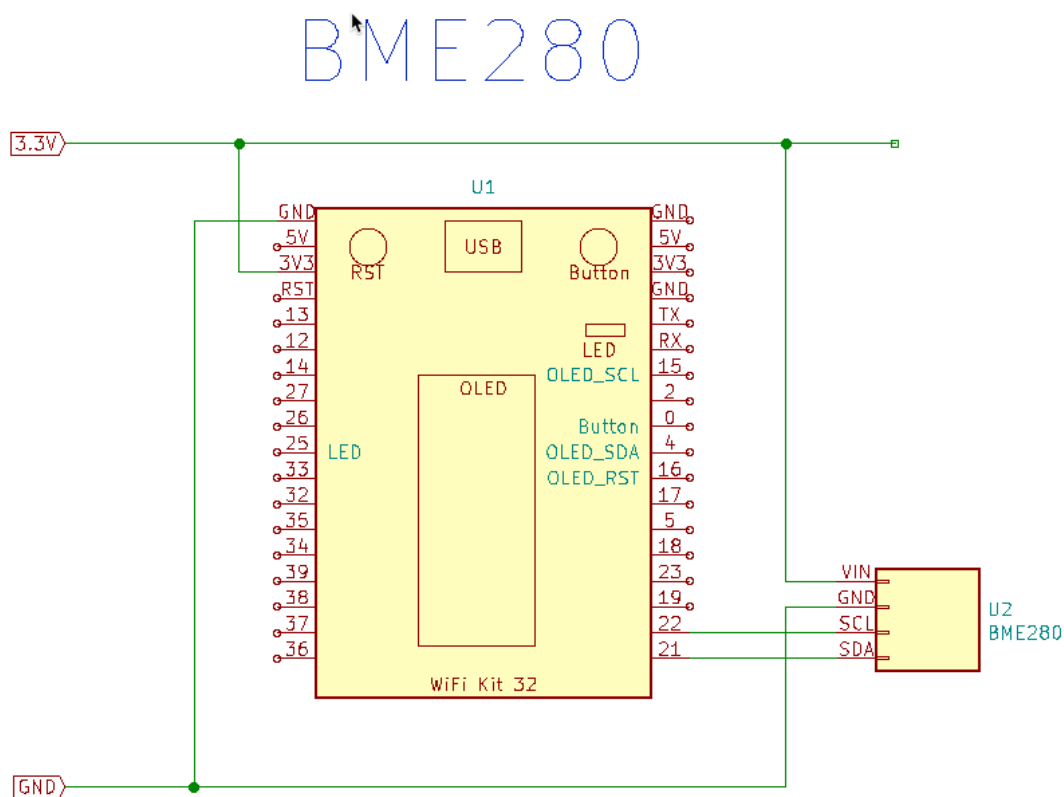
Genauigkeit:

Feuchte: +/- 3%
Druck: +/- 1.5 hPa
Temperatur: +/- 1 °C

Webseite: https://www.bosch-sensortec.com/bst/products/all_products/bme280

Datenblatt: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BME280-DS002.pdf

Der Versuchsaufbau



Die Software

Bosch stellt eine C - Bibliothek zur Verfügung, von der eine gute Umsetzung in Micropython existiert.

Von Bosch (C): https://github.com/BoschSensortec/BME280_driver

Micropython: https://github.com/triplepoint/micropython_bme280_i2c


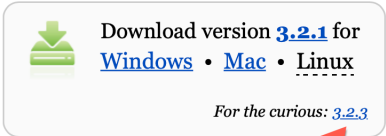
Die Datei **bme280_i2c.py** enthält die Python-Bibliothek und muss auf das Board kopiert werden.

Die Quelltexte der ersten Messungen sind in den Begleitunterlagen zu finden.

Die Werte von Temperatur und Luftfeuchtigkeit entsprechen etwa den erwarteten Werten. Der Luftdruck erscheint im Moment als sehr ungenau. Das liegt aber daran, dass der Sensor den absoluten Luftdruck misst. Angegeben wird aber normalerweise ein auf Meereshöhe normierter Wert. Diese liegt höher als der

Thonny

Python IDE for beginners




ReleasesTags

Latest release

v3.2.3
f3ff7c5
Verified



Version 3.2.3

 aivarannamaa released this 7 days ago

3.2.3 together with short lived 3.2.2 are bug-fix releases.

Changes in 3.2.2 and 3.2.3:

- NEW: ESP plug-in has been merged into main Thonny package

 thonny-3.2.3.dmg	18.7 MB
 thonny-3.2.3.exe	14.1 MB

Lektion 9: Die Thonny-Version 3.2.3

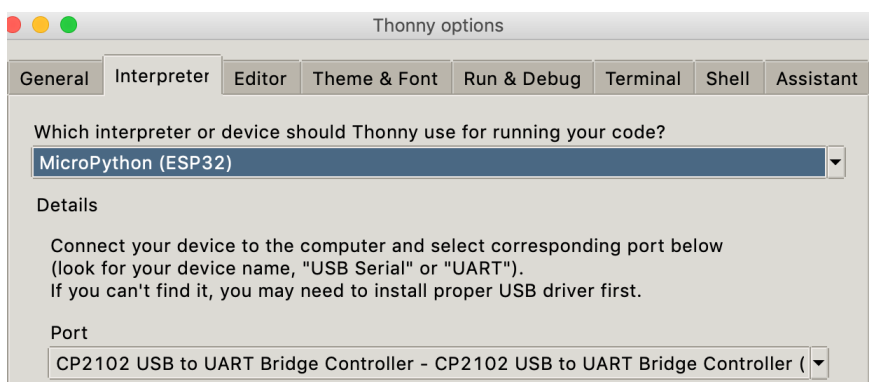
Die neue Thonny - Version unterscheidet sich in wichtigen Punkten von der Version 3.1.x. Momentan ist sie erst als Vorversion verfügbar, daher müssen auch einige Dinge beachtet werden.

Installation von Thonny

Auf der Webseite von Thonny (<https://thonny.org>) findet man den Download - Link. Falls als aktuelle Version immer noch Version 3.2.1 angeboten wird, sollte die Vorversion 3.2.3 heruntergeladen werden. Der Link führt zu Github, dort sind unten die Versionen für Mac, Windows und diverse Linux-Varianten erhältlich.

dd

Falls bereits eine ältere Version vorhanden war, bleiben bei der Installation alle Plugins und die Einstellungen erhalten. Das Plugin esptool liegt inzwischen In Version 2.8 vor und sollte allenfalls aktualisiert werden. Das Plugin thonny-esp wird nicht mehr benötigt und kann deinstalliert werden.



In den Einstellungen sollte überprüft werden, ob der richtige Interpreter und der korrekte Treiber ausgewählt ist.

Aktualisieren der Micropython - Version auf dem Board

Bei dieser Gelegenheit kann auch die Micropython - Version auf dem Board aktualisiert werden.

Es gibt drei Möglichkeiten:

1. Die aktuellste ESP-IDF v3.x

Diese Version unterstützt noch kein Bluetooth, enthält aber alle Teile, die wir benötigen. Es handelt sich um einen aktuellen Build, der beinahe täglich erneuert wird.

2. Die stabile ESP-IDF v3.x

Diese Version ist ebenso geeignet. Sie stammt vom 29.05.2019 und gilt als stabile Version 1.11

3. Die aktuellste ESP-IDF v4.x

Ich habe diese Version installiert, da sie Experimente mit Bluetooth erlaubt. Das wird allerdings im Kurs momentan nicht benutzt.

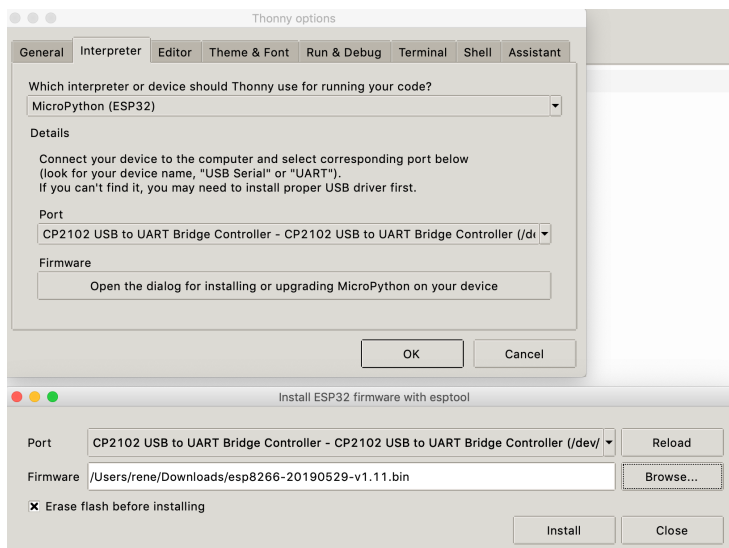
Firmware built with ESP-IDF v3.x, with support for LAN and PPP but no bluetooth:

- GENERIC : [esp32-idf3-20191110-v1.11-571-g7e374d231.bin](#)
- GENERIC : [esp32-idf3-20190529-v1.11.bin](#)
- GENERIC : [esp32-idf3-20190125-v1.10.bin](#)
- GENERIC : [esp32-idf3-20180511-v1.9.4.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20191110-v1.11-571-g7e374d231.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20190529-v1.11.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf3-20190125-v1.10.bin](#)
- TinyPICO : [tinypico-idf3-20191110-v1.11-571-g7e374d231.bin](#)

Firmware built with ESP-IDF v4.x, with support for bluetooth but no LAN or PPP:

- GENERIC : [esp32-idf4-20191110-v1.11-571-g7e374d231.bin](#)
- GENERIC-SPIRAM : [esp32spiram-idf4-20191110-v1.11-571-g7e374d231.bin](#)
- TinyPICO : [tinypico-idf4-20191110-v1.11-571-g7e374d231.bin](#)

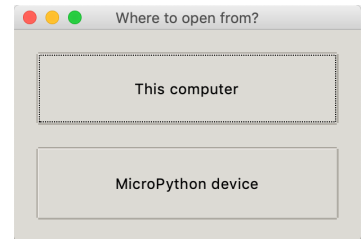
Die Installation erfolgt über Tools / Options / Interpreter



Die Option **Erase flash before installing** sollte aktiviert sein.

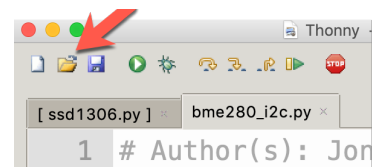
Neue Arbeitsweise mit Thonny 3.2.3

Das Laden, Speichern und Kopieren von Dateien wird über Funktionen des Menüs File durchgeführt. Dabei hat man immer die Wahl zwischen der lokalen Festplatte und dem ESP32 - Board. Dabei kann es vorkommen, dass der Text **This Computer** nicht sichtbar ist. Die Auswahl funktioniert aber trotzdem. **Alternativ können die meisten Aufgaben auch über den File - Browser vorgenommen werden.**



Laden einer Datei

Die Funktion wird über **File / Open** oder das Laden - Symbol aufgerufen. Der Inhalt der geladenen Datei erscheint dann im Edit - Fenster. Der Dateiname erscheint im Titel des Tabs. Wenn er von eckigen Klammern umschlossen ist, wurde die Datei vom Board geladen, sonst vom PC.

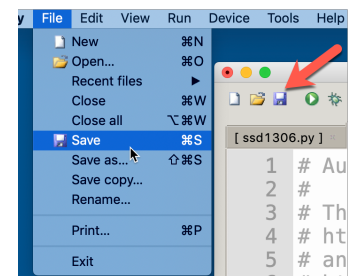


Speichern einer Datei

Die Funktion wird über **File / Save** oder das Speichern - Symbol aufgerufen. Dabei wird der Inhalt des aktuellen Edit-Fensters ohne Nachfrage in die ursprüngliche Datei kopiert.

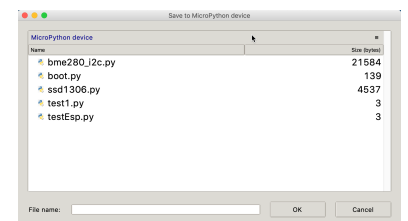
Speichern einer neuen Datei

Neue Dateien, die noch nie gespeichert wurden, sollten mit **File / Save as...** gespeichert werden.



Kopieren einer Datei von der lokalen Festplatte auf das Board

Das File wird mit **File / Load** von der lokalen Festplatte geladen und dann mit **Save copy...** auf das Board kopiert. Leider bietet das Programm den ursprünglichen Dateinamen nicht als Vorschlag an. Er muss deshalb eingegeben werden.



Umbenennen einer Datei

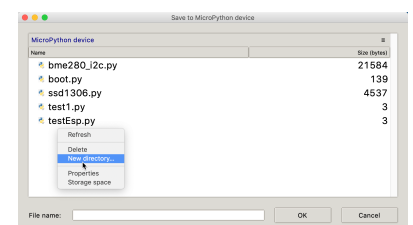
Das Umbenennen erfolgt mit **Rename...**. Der Befehl bezieht sich immer auf die aktuelle Datei im Editor - Fenster. Sie muss also dort geöffnet sein.

Verzeichnis anlegen

Dazu gibt es keinen direkten Befehl. Es gibt aber die Möglichkeit in der Dateiauswahl von Thonny mit der **rechten Maustaste** einen **New directory...** - Befehl aufzurufen.

Löschen einer Datei oder eines Verzeichnisses

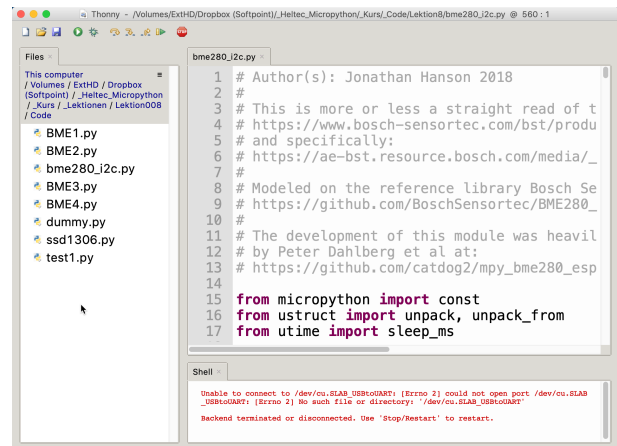
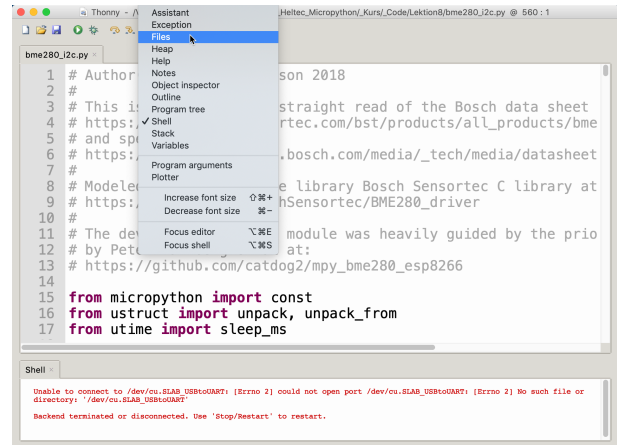
Mit der **rechten Maustaste** erhält man auch die Möglichkeit den **Delete** - Befehl auszuführen. Er kann Dateien und Verzeichnisse löschen.



Der File - Browser

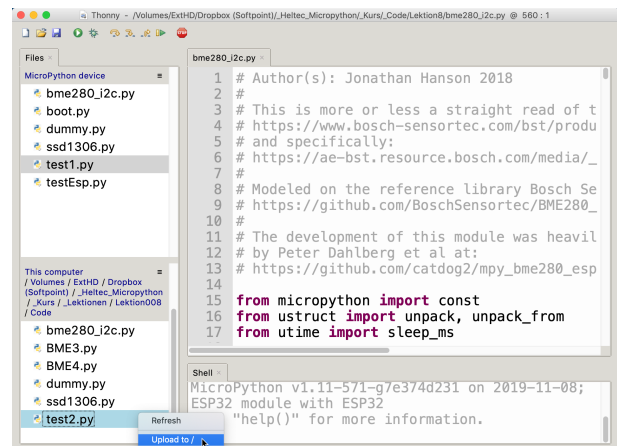
Insbesondere das Kopieren von Dateien zwischen Festplatte und Board lässt sich bequem mit dem File-Browser erledigen. Er ist erreichbar über View / Files.

Falls du dann nur deine lokale Festplatte siehst, ist dein Board nicht verbunden.



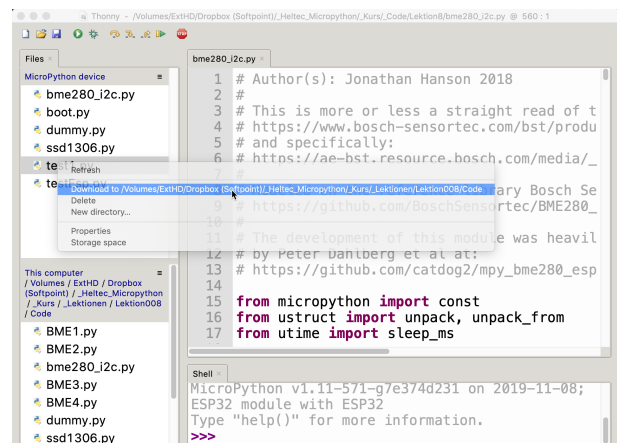
Sobald das Device verbunden ist, erscheint das zweite Fenster mit den Dateien auf dem Board.

Mit der **rechten Maustaste** kannst du die Funktion **Upload to** / auslösen. Diese kopiert die Datei unter demselben Namen auf das Board.



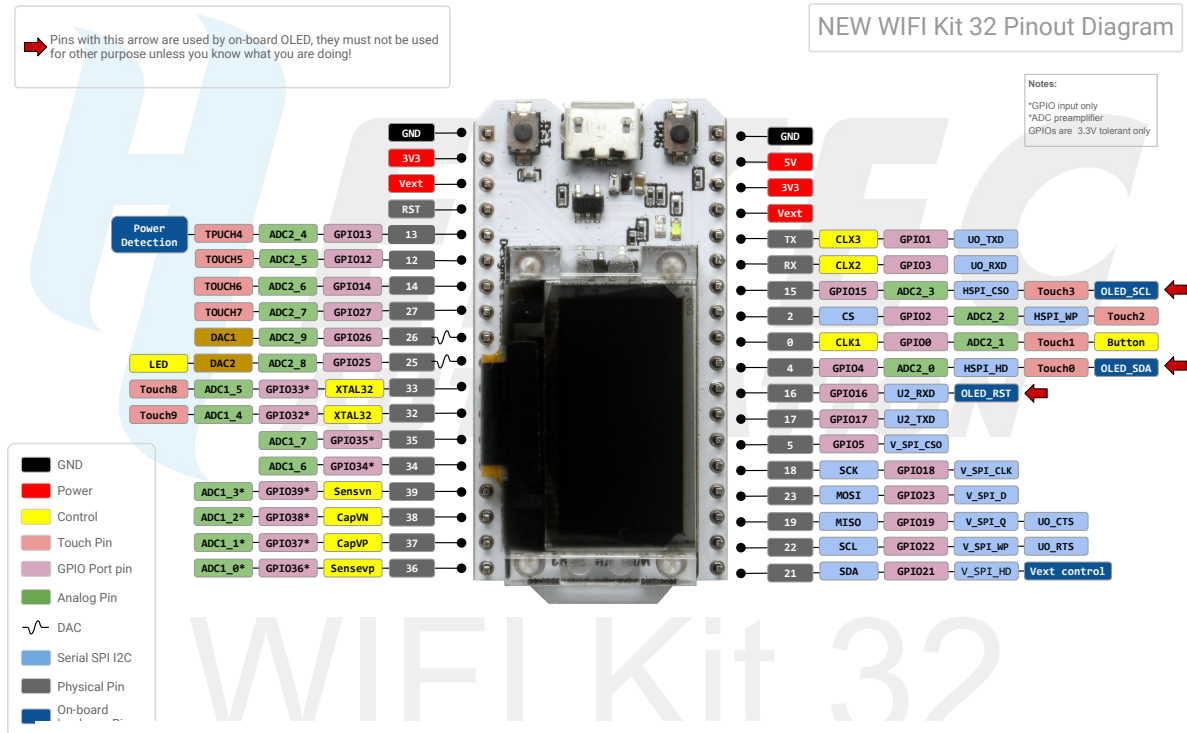
Der umgekehrte Weg ist ebenfalls möglich. Die Funktion **Download to** erlaubt eine Datei vom Board auf deine Festplatte zu kopieren.

Zusätzlich stehen im Kontextmenu auch die Funktionen **Delete** und **New directory...** zur Verfügung.

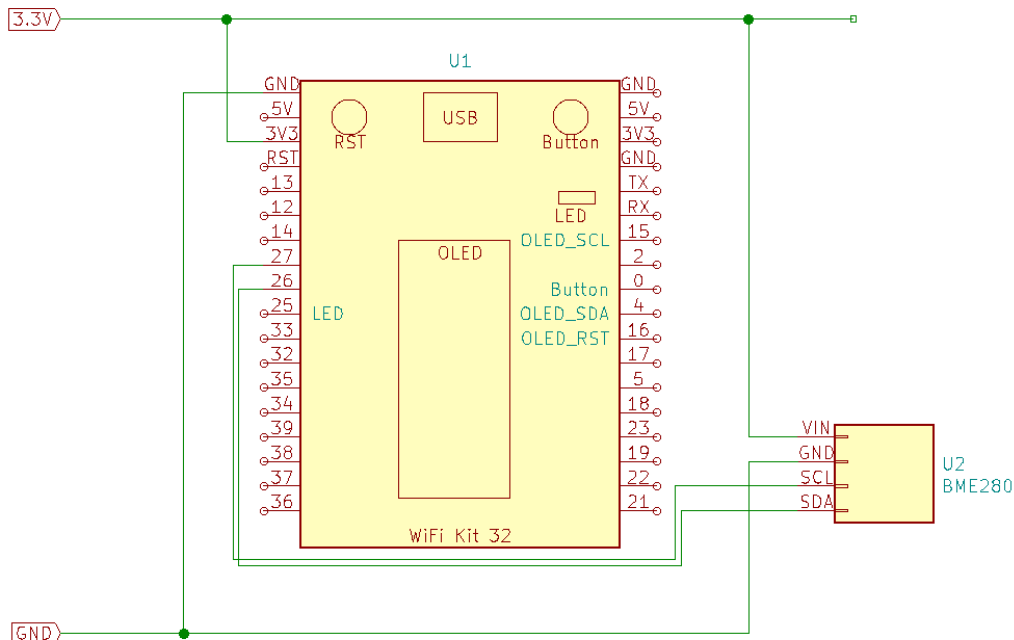


Lektion 10: Neue Heltec-Version

Es gibt eine neue Version des Heltec - Boards. Damit unser Programm auf beiden Boards läuft, passen wir unsere Schaltung an.



BME280



Das führt zu zwei Programmänderungen:

statt

```
i2c_bme = I2C(scl=Pin(22), sda=Pin(21), freq=10000)
...
i2c_oled = I2C(scl=Pin(15), sda=Pin(4), freq=10000)
```

schreiben wir

```
i2c_bme = I2C(scl=Pin(27), sda=Pin(26), freq=10000)
...
i2c_oled = I2C(scl=Pin(15, Pin.OUT, Pin.PULL_UP), sda=Pin(4, Pin.OUT,
Pin.PULL_UP))
```

Lektion 11: Nochmals BME280

Wir verwenden also den BME280 auf den Ports 26 und 27. So funktioniert die Schaltung sowohl mit der neuen Boardversion, wie auch der alten.

Wir haben auch festgestellt, dass beide BME - Versionen (5V und 3.3V) einsetzbar sind. Beide müssen mit 3.3V betrieben werden.

In einer früheren Lektion haben wir festgestellt, dass unsere Luftdruckwerte überhaupt nicht mit denen im Wetterbericht übereinstimmen. Das liegt daran, dass wir den tatsächlichen Luftdruck messen, der von Wetterstationen angegebene Luftdruck aber auf Meereshöhe umgerechnet wurde. Der Luftdruck hängt von der Höhe, der Temperatur und in geringen Masse auch von der Luftfeuchtigkeit ab. Um vergleichbare Werte zu erhalten, wird er normiert. Dies kann über die barometrische Höhenformel erfolgen. Der Blick auf die Formel in Wikipedia zeigt uns aber sofort, dass das viel zu kompliziert ist.

So verwenden wir eine viel einfachere Lösung. Wir arbeiten mit einem leicht zu ermittelnden Korrekturfaktor.

Das ist ganz einfach:

- Wir suchen im Internet nach einer offiziellen Wetterstation in der Nähe
- Das Wetter sollte ruhig sein. So können wir annehmen, dass der Luftdruck in unserem Umkreis mehr oder weniger konstant ist.
- Den Druck bei der offiziellen Station dividieren wird durch unseren gemessenen Druck. Das gibt dann den gewünschten Faktor.

Dieser Faktor sollte mehr oder weniger konstant bleiben, da er hauptsächlich von unserem Standort abhängt. Er ist aber auch temperaturabhängig, so dass es vermutlich einen Unterschied zwischen Sommer und Winter gibt. Ob dadurch weitere Korrekturen notwendig sind, wird sich noch zeigen.