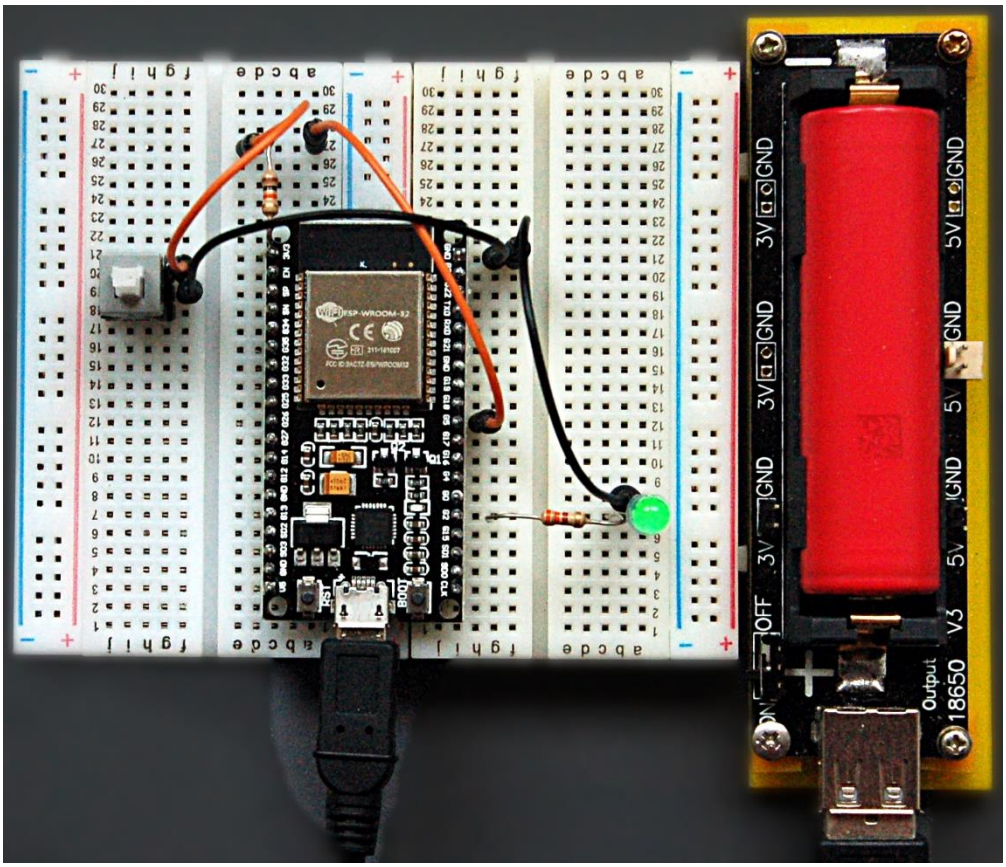


Projekte mit MicroPython und dem ESP8266/ESP32 (Teil 1)

Der Blogbeitrag besteht aus mehreren Teilen. Im ersten Teil geht es um das Bereitstellen der Firmware auf einem ESP32-Board und einige erste Versuche in der neuen Umgebung.

Im zweiten Teil befassen wir uns mit weiterer Peripherie, binden den ESP32 als Server ins WLAN ein und messen analoge Spannungen. Das alles sind Vorbereitungen auf den dritten Teil.

Ziel der bisherigen Unternehmungen ist es, den ESP32 als Messwertaufnehmer für Kernstrahlung zu verwenden. Dabei kommen alle vorbereiteten Teile wieder zum Einsatz. Wir fügen alles zu einem System zusammen, das mit einer einfachen Infrarot-Empfangsdiode Kernstrahlung detektieren und nach Energieklassen sortieren kann. Die Ergebnisse werden am Display angezeigt und können per WLAN oder auch über das Internet abgerufen werden.

Einige Hinweise gleich zu Beginn:

- Dieses Projekt ist nur auf einem 64-Bit-System durchzuführen, weil dafür Software erforderlich ist, die nur für 64-Bit-Systeme verfügbar ist.
- Die Aktionen aus diesem Teil laufen auf dem ESP32 nicht autonom, sondern nur, wenn sie vom PC aus via µPyCraft gestartet werden. So wird es auch im 2. Teil gehandhabt, weil dieses Vorgehen Änderungen und Verbesserungen

am Programmtext viel besser zulässt. Für das fertige Produkt, das im dritten Teil vorgestellt wird, ändern wir das freilich, sodass der ESP32 auch ohne PC seine Aufgaben erfüllen kann.

- Wenn Sie der Meinung sind, MicroPython bringt's nicht, dann würde ich das zwar bedauern. Sie können Ihren ESP32/ESP8266 aber jederzeit wieder mit der Arduino-IDE programmieren, indem Sie die MicroPython-Firmware auf dem Board einfach mit einem compilierten Arduino-Sketch überschreiben. Mehr ist nicht zu unternehmen.

Was ist MicroPython

MicroPython wurde als schlanke Programmiersprache für Mikrocontroller-Plattformen aus Python 3.4, aka CPython, abgeleitet. Damit Systeme mit begrenztem Speicherplatz mit der Implementierung arbeiten können, wurden viele der Merkmale von CPython weggelassen, andere Eigenschaften, die für Microcontroller nützlich und wichtig sind hinzugefügt. Neben LUA und der Arduino-IDE, die auf dem SDK2.2.1 von Espressif beruht, ist MicroPython eine gebräuchliche Programmierumgebung für die Entwicklung von Anwendungen mit Microcontrollern. Die Firmware benötigt 256kB an Flashspeicher und 16kB RAM.

Wie MicroPython auf einem Microcontroller installiert wird und wie man damit arbeiten kann, werde ich im Folgenden darstellen, und ich lade Sie ein, an diesem Experiment teilzunehmen.

In diesem Beitrag werden Sie erfahren,

- welche Hardware man für diesen Artikel braucht
- welches Werkzeug man benötigt, um MicroPython auf einen Microcontroller zu übertragen
- woher man die Software bekommen kann
- wie man sie einsetzt und bedient
- wie man ein erstes Demoprogramm zum Laufen bringt.

MicroPython auf einen Mikrocontroller flashen

Was braucht man an Hardware?

Für diese Anleitung braucht man
ein ESP8266-Board
NodeMCU Lua LoLin V3 Module ESP8266 ESP-12F oder
NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F oder
D1 Mini NodeMcu mit ESP8266-12F oder

oder

ein ESP32-Board z.B.
ESP-32 Dev Kit C V4
ESP32 NodeMCU Module WLAN WiFi Development Board

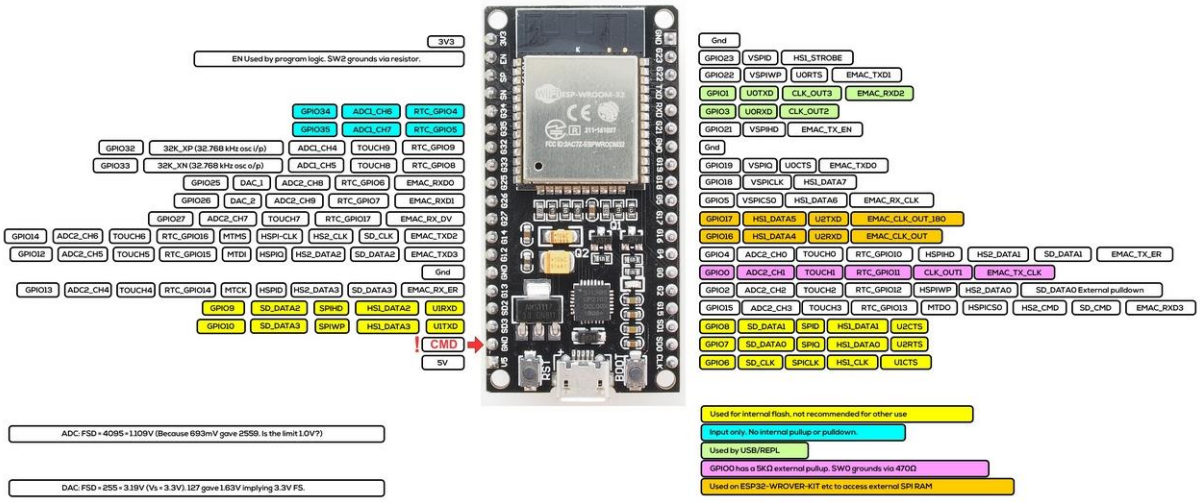
Alle Boards haben bereits einen USB-RS232-Adapter an Bord. Sie sind mit 4MB (4 MByte = 32 Mb = 32 Mbit Flashspeicher (aka Programmspeicher) ausgestattet.

Die Boards werden über die USB-Schnittstelle mit Firmware und eigenen Programmen versorgt und können mit dem PC Daten austauschen. Die Firmware ist für einen Microcontroller das Betriebssystem, ohne das der Controller weder ansprechbar noch arbeitsfähig ist. MicroPython ist so ein Betriebssystem. Die genannten Boards werden aber meist mit NodeMCU-LUA oder AT-Firmware ausgeliefert. Wer mit MicroPython darauf arbeiten will, muss zunächst die neue Firmware auf das Board übertragen. MicroPython-Programme werden auf die gleiche

Weise übertragen, nachdem die Firmware, also das Betriebssystem, hochgeladen wurde.

Die beiden Gruppen von Boards unterscheiden sich zunächst einmal durch den darauf verbauten Microcontroller, den Umfang an Programmspeicher und durch die Anzahl an herausgeführten Anschlüssen, die Pins. Der leistungsfähigere ESP32 ist der Nachfolger des kleineren ESP8266. Ferner befindet sich auf den Boards ein USB-Schnittstellenbaustein. Welcher das ist, lässt sich aus den Produktbeschreibungen, die durch die obigen Links aufgerufen werden können, ersehen. Ich gehe gleich bei der Softwarebeschaffung und Installation noch einmal darauf ein.

Warum habe ich mich entschlossen, das folgende Board zusammen mit MicroPython auszuprobieren? Das **ESP-32 Dev Kit C V4** hat ausreichend Anschlusspins, um damit alle möglichen Sensoren anzuschließen. Im Zweiten Teil des Beitrags geht es nämlich um eine Messstation, deren Werte über einen Browser oder eine Handy-App abrufbar sein sollen. Weil der ESP32 über diverse Busschnittstellen wie I2C, Onewire, SPI usw. verfügt, hat jeder Pin meistens mehr als eine Aufgabe und Bezeichnung.



Und warum MicroPython? Na ja, weil ich mich schon lange einmal mit dieser Programmiersprache befassen wollte. In der Vergangenheit hatte ich einige Projekte mit LUA und mit der Arduino-IDE ausprobiert und möchte jetzt mal etwas Neues kennenlernen.

Doch halt – ein paar Sachen zur Hardwareausstattung fehlen noch. Um die Platinen mit dem PC verbinden zu können, benötige ich für diese Module noch ein **USB-Kabel** mit **USB A- zu Micro-B**-Stecker. Und für die erste "Unterhaltung" mit dem ESP32 brauchen Sie noch eine **LED, rot oder grün** und einen **Widerstand von 330 Ohm**. Für einen Tasteranschluss brauchen wir den **Taster** und einen **Widerstand von 10kOhm**. Für die Verbindungen halten wir einige **Jumperkabel** und ein Breadboard bereit.

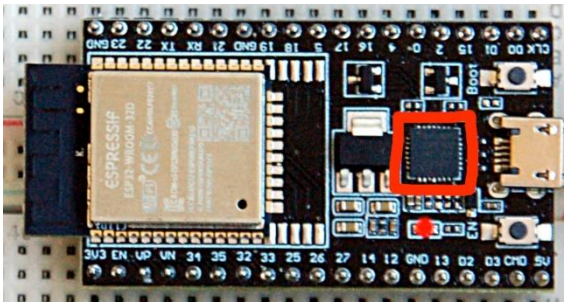
Zusammengefasst kommen also in diesem Beitrag zum Einsatz:

- **1 ESP-32 Dev Kit C V4**
- **1 USB-Kabel USB-A zu UBS-Mikro-B**
- **1 rote oder grüne LED**
- **1 Widerstand 330 Ohm**
- **1 Taster**
- **1 Widerstand 10kOhm**
- **Jumperkabel**
- **Breadboard(s)**

Besorgen der Software

Alle aufgeführten Softwareteile unterliegen entweder der GPL oder vergleichbaren Lizenzbestimmungen und sind kostenfrei aus dem Internet zu beziehen.

Um die Boards ansprechen zu können, ist zunächst der Treiber für den USB-Schnittstellenbaustein nötig. Je nach Board handelt es sich um einen **CH340G** oder einen **CP2102**, wie in meinem Fall, beim ESP-32 Dev Kit C V4-Board. Der Baustein ist im Bild rot eingerahmt.



CH340G-Treiber (keine Lizenzangaben) wird für das Amica-Board für Windows und Mac gebraucht, Linux benötigt keinen extra Treiber: https://github.com/nodemcu/nodemcu-devkit.

CP2102-Treiber für Windows, Mac, Android, Linux benötigt keinen extra Treiber: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

µPyCraft
Das Werkzeug zum Flashen der Firmware MicroPython ist µPyCraft (keine Lizenzangaben). Das Tool kann von GitHub heruntergeladen werden. Das Paket enthält Versionen für Windows, Linux und Mac. Neben den Werkzeugen zum Flashen der Firmware bietet µPyCraft einen Editor für MicroPython-Programme und ein Terminal für den unmittelbaren Kontakt zum MicroPython-Kommandozeilen-Interpreter (REPL).

Python 3.8.5
Damit µPyCraft überhaupt läuft, muss auf dem PC Python ab Version 3.7 aufwärts installiert sein. Das aktuelle Python3-Release ist 3.8.5. Folgen Sie zum Download dem Link Latest Python 3 Release - Python 3.8.5 (GPL-compatible). Aus der Liste "Files" am unteren Ende der Seite wählen Sie die Version, die zu Ihrer Windowsinstallation passt.

Firmware Binaries
Dann kommt das Wichtigste, die MicroPython-Firmware (MIT License), ohne die gar nichts geht. Die hole ich von https://micropython.org/download/all/. Der direkte Link für einen ESP8266 ist

https://micropython.org/resources/firmware/esp8266-1m-20200902-v1.13.bin

und die Firmware für den ESP32 bekomme ich hier

https://micropython.org/resources/firmware/esp32-idf3-20200902-v1.13.bin

Laden Sie bitte alle Teile der Software sowie die entsprechende Firmware herunter, und legen Sie diese in einem Verzeichnis Ihrer Wahl ab. Ist das getan, geht es ans Auspacken und Installieren.

Installation der Software

Treiberinstallation.

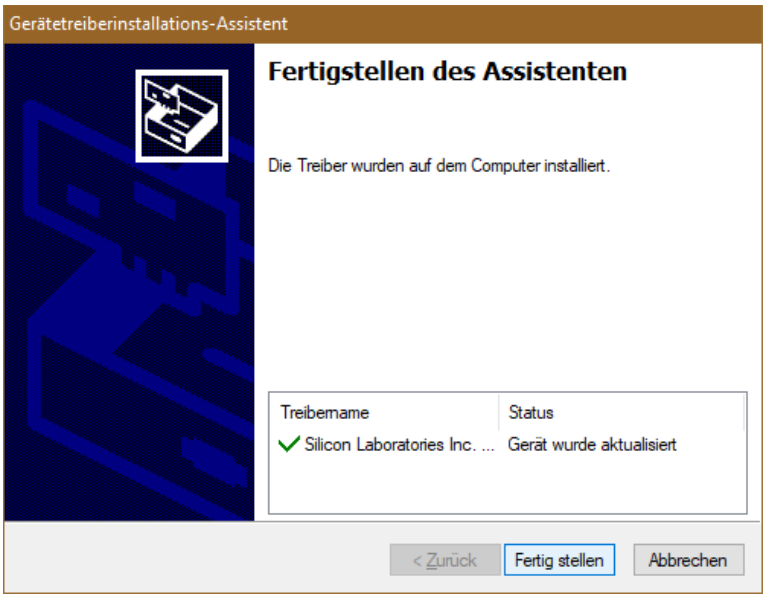
Der **Treiber** für den **CP210x** kommt in dem ZIP-Archiv

CP210x_Universal_Windows_Driver.zip,

das man in ein Verzeichnis der eigenen Wahl entpackt. Im entstandenen Ordner wird die Datei

CP210xVCPInstaller_x64.exe
gestartet.

Im Willkommensfenster auf **Weiter** klicken und schließlich auf **Fertig stellen**.

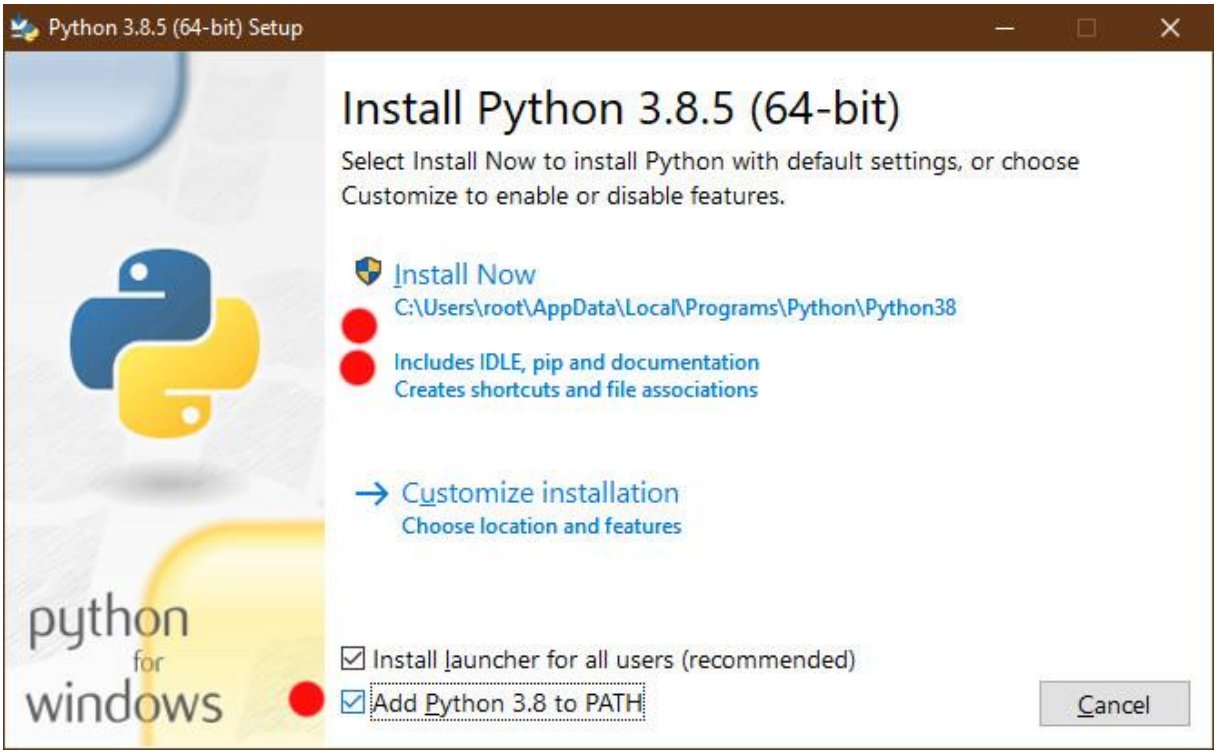


Python 3.8.5

Die Installationsdatei

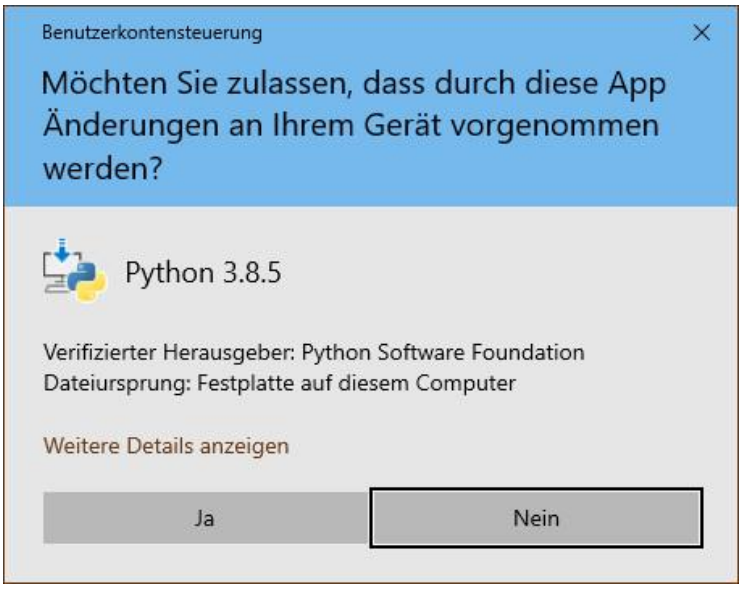
python-3.8.5-amd64.exe

wird gestartet.

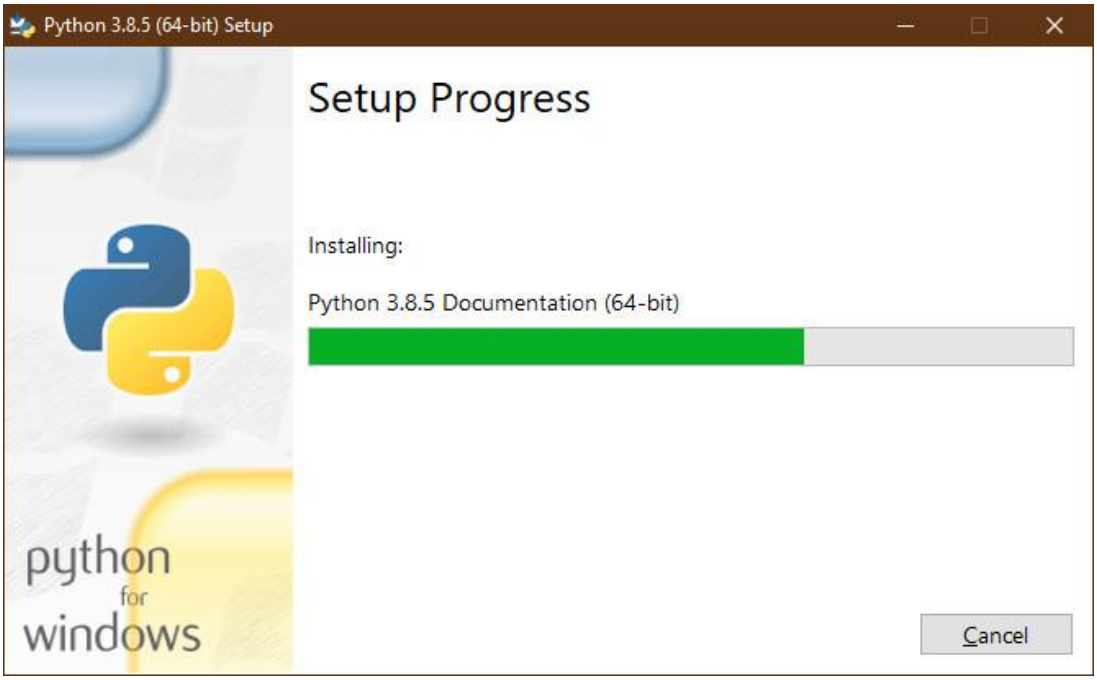


Das Kästchen **Add Python 3.8 to Path** bekommt einen Haken, dann klicke ich auf **Install Now**. Die Umgebungsvariable PATH enthält Einträge von Pfaden zu Anwendungsverzeichnissen. Durch so einen Eintrag lässt sich die jeweilige Anwendung aus beliebigen Verzeichnissen heraus starten.

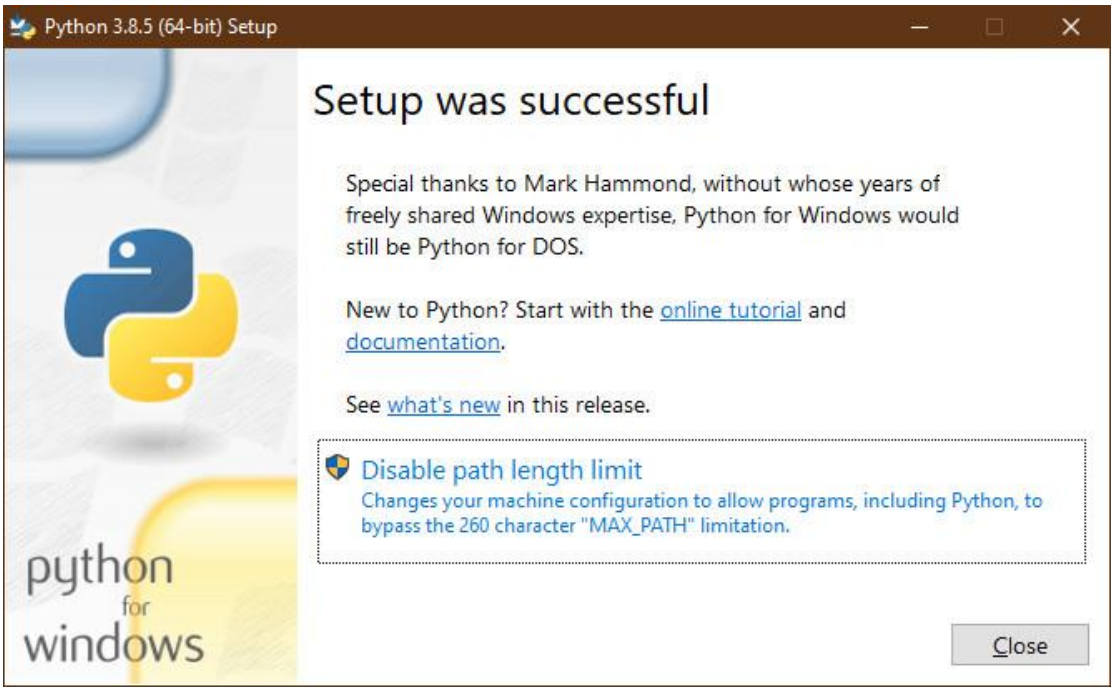
Die Frage ob Änderungen am System vorgenommen werden dürfen beantworte ich mit **Ja**.



Das Setup läuft durch.



Und schließlich erscheint die Erfolgsmeldung. Das Fenster wird geschlossen.

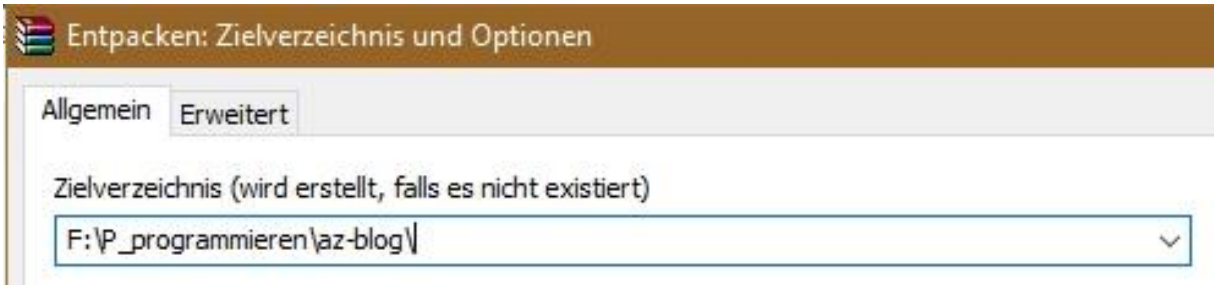


µPyCraft

Die heruntergeladene Datei ist das ZIP-Archiv

uPyCraft-master.zip.

Ich entpacke es in einen Ordner meiner Wahl.



Es ist ein Ordner mit gleichem Namen entstanden. Die Software muss nicht weiter installiert werden und wird später aus dem Verzeichnis gestartet, in das sie jetzt entpackt wurde. Man kann auch einen Link dorthin auf dem Desktop erstellen.

Jetzt kann es losgehen. Das Vorbereiten der Hardware dauert nur Sekunden und dann bringen wir dem ESP32 bei, wie Python geht.

Haben Sie alles an Software beisammen?

- CP2102-Treiber
- CPython
- µPyCraft
- MicroPython-Firmware Binaries

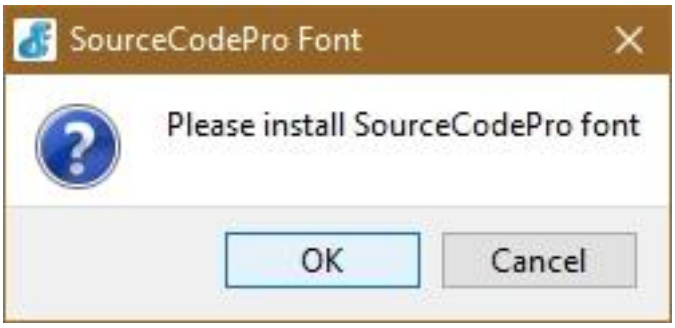
Dann schaffen wir die Firmware auf den ESP32.

Flashen der Firmware

Ich verbinde nun das USB-Kabel mit einer freien USB-Buchse am PC und den USB-Mikro-B-Stecker mit dem ESP-12F-Board. Die rote LED zeigt durch kurzes Blinken an, dass das Board unter Spannung steht.

Das Flashtool **µPyCraft** wird aus dem Verzeichnis gestartet, in das es entpackt wurde. Aus der Reihe der Windowsapplikationen wähle ich das mit der höchsten Revisionsnummer, also **uPyCraft_V1.0.exe**.

Nach dem Start erscheint ein Fenster zum Installieren eines Zeichensatzes, das ich mit OK quittiere.

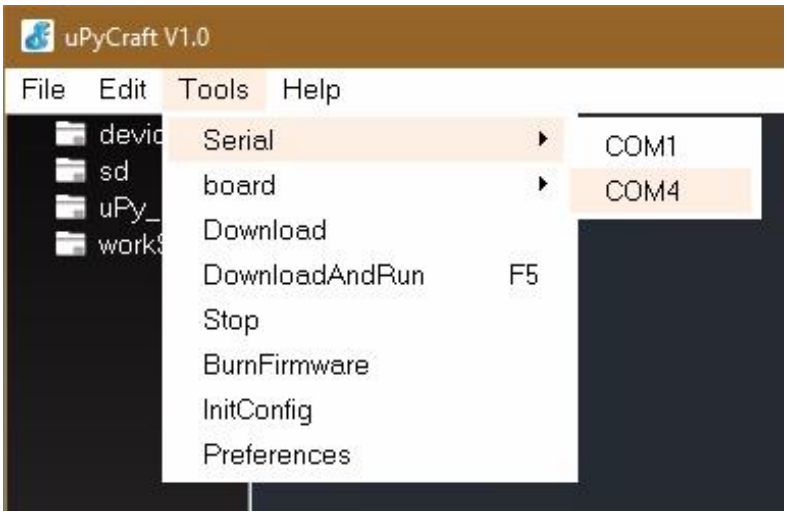


Drei Dinge sind nun noch nötig, dann kann man eine schnelle Tasse Kaffee nehmen. Alle Aktionen laufen über das Menü **Tools**.

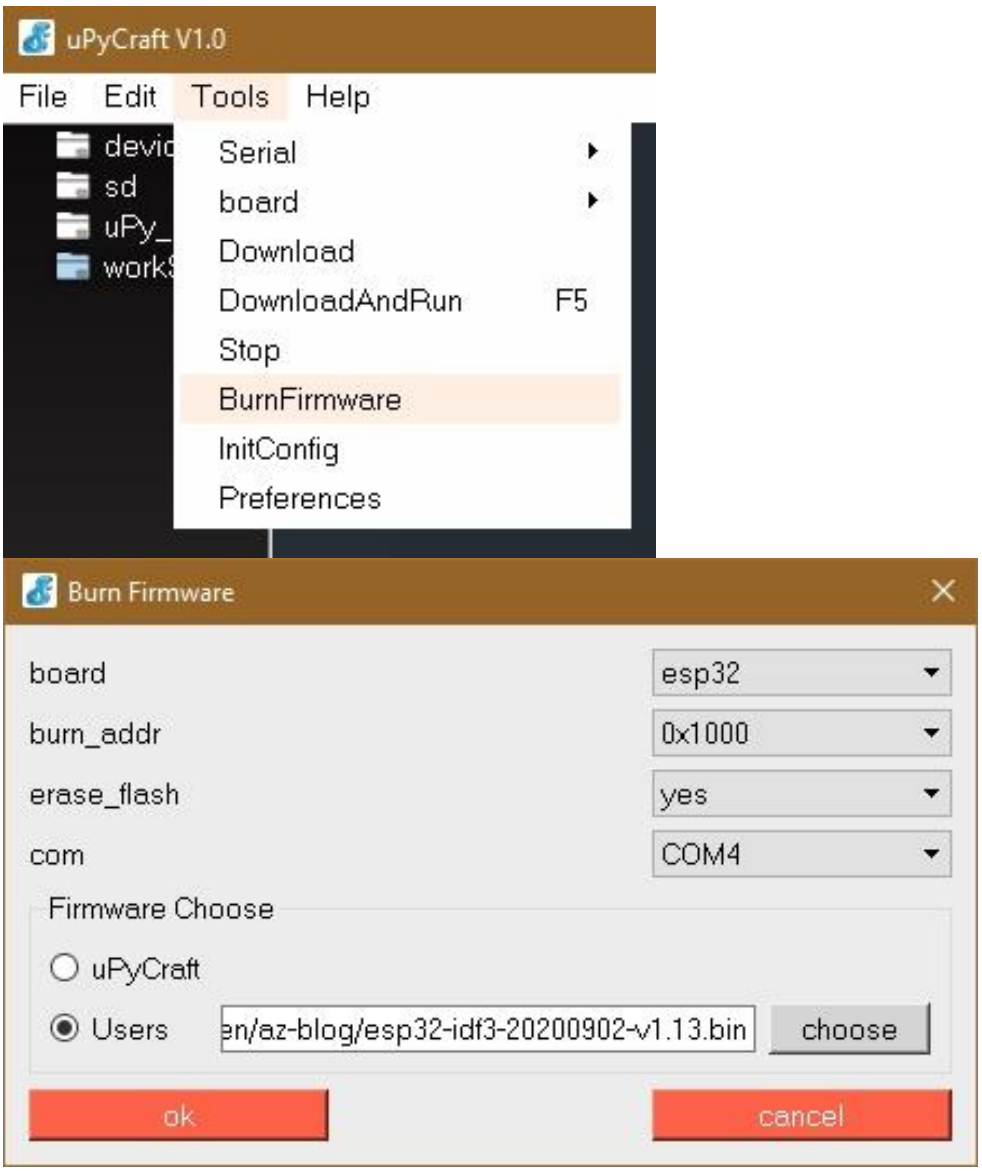
Das Board, beziehungsweise der Prozessortyp muss ausgewählt werden.

Dann kommt die serielle Schnittstelle dran. Wenn der Treiber ordentlich installiert wurde, sieht man jetzt mindestens einen Eintrag in der Liste der COM-Ports. COM1 ist bei mir die echte serielle Schnittstelle, der CP2102 ist demnach COM4. Die Bezeichnung COM4 kann auf Ihrem System eine andere laufende Nummer haben und ist entsprechend auszuwählen.

Eine echte serielle Schnittstelle wird durch eine Steckkarte oder einen entsprechenden Chip auf dem Mainboard des PCs bereitgestellt. Der CP2102 emuliert eine solche Schnittstelle via USB. Das heißt der Baustein übersetzt USB-Signale in serielle Signalfolgen und umgekehrt.

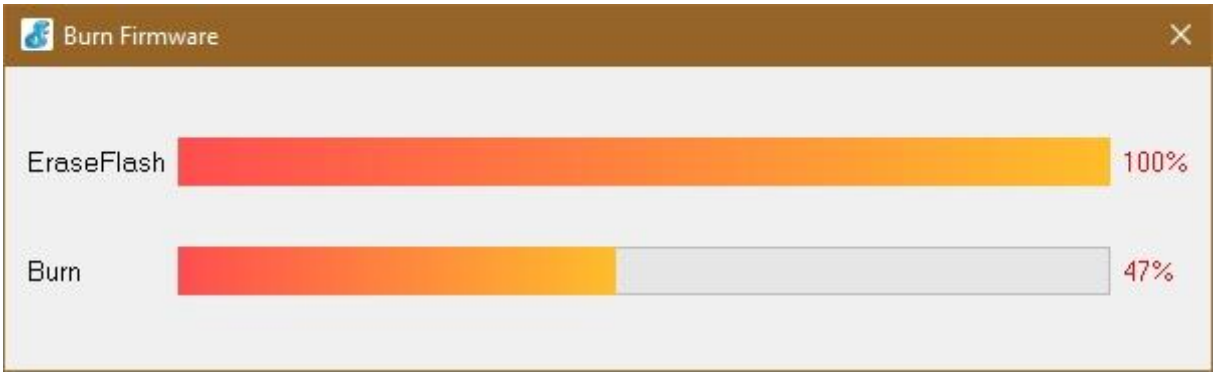


Beim dritten Aufruf des Tools-Menüs wähle ich BurnFirmware. In dem Fenster gibt es noch einiges einzustellen.



Der ESP32 wird ab 0x1000 geflasht, die Startadresse bei einem ESP8266 ist 0x0000. Die Firmware wählen wir als **User** aus und zwar ist es die herunter geladene Datei von MicroPython, also **choose**.

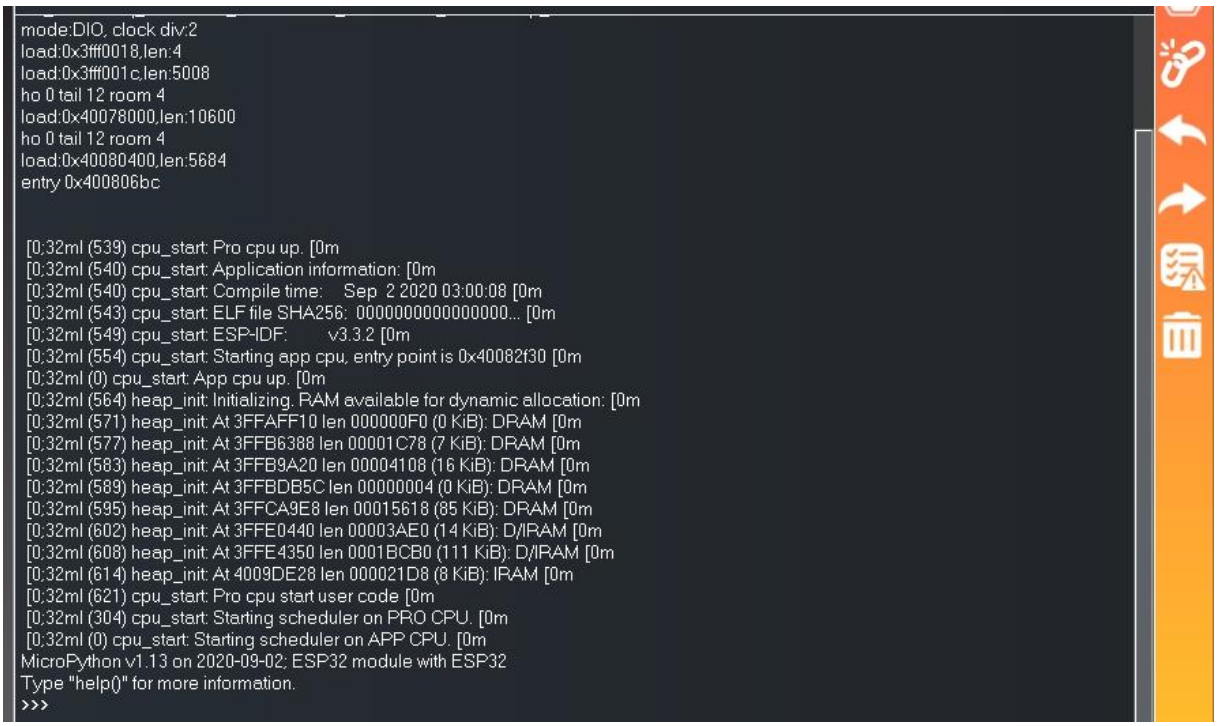
Mit einem Klick auf **ok** startet der Brennvorgang.



Das war's dann auch schon mit den Vorbereitungen. Jetzt probieren wir mal aus, ob alles wirklich geklappt hat.

Test der neuen Umgebung

Nach dem Flashvorgang muss der ESP32 neu gestartet werden, drücken Sie die Taste EN auf dem Board neben der USB-Buchse.



Im Terminalfenster von µPyCraft erscheint ein ganzer Schwall an Meldungen und zum Schluss werden drei Größer-Zeichen ausgegeben ">>>". Das ist der Kommandozeilenprompt von MicroPython. Das bedeutet, dass Sie jetzt an dieser Stelle Befehle an den Microcontroller senden können. Testen wir das mit dem Allerweltsgruß. Ich gebe folgendes ein und schließe mit der Entertaste ab.

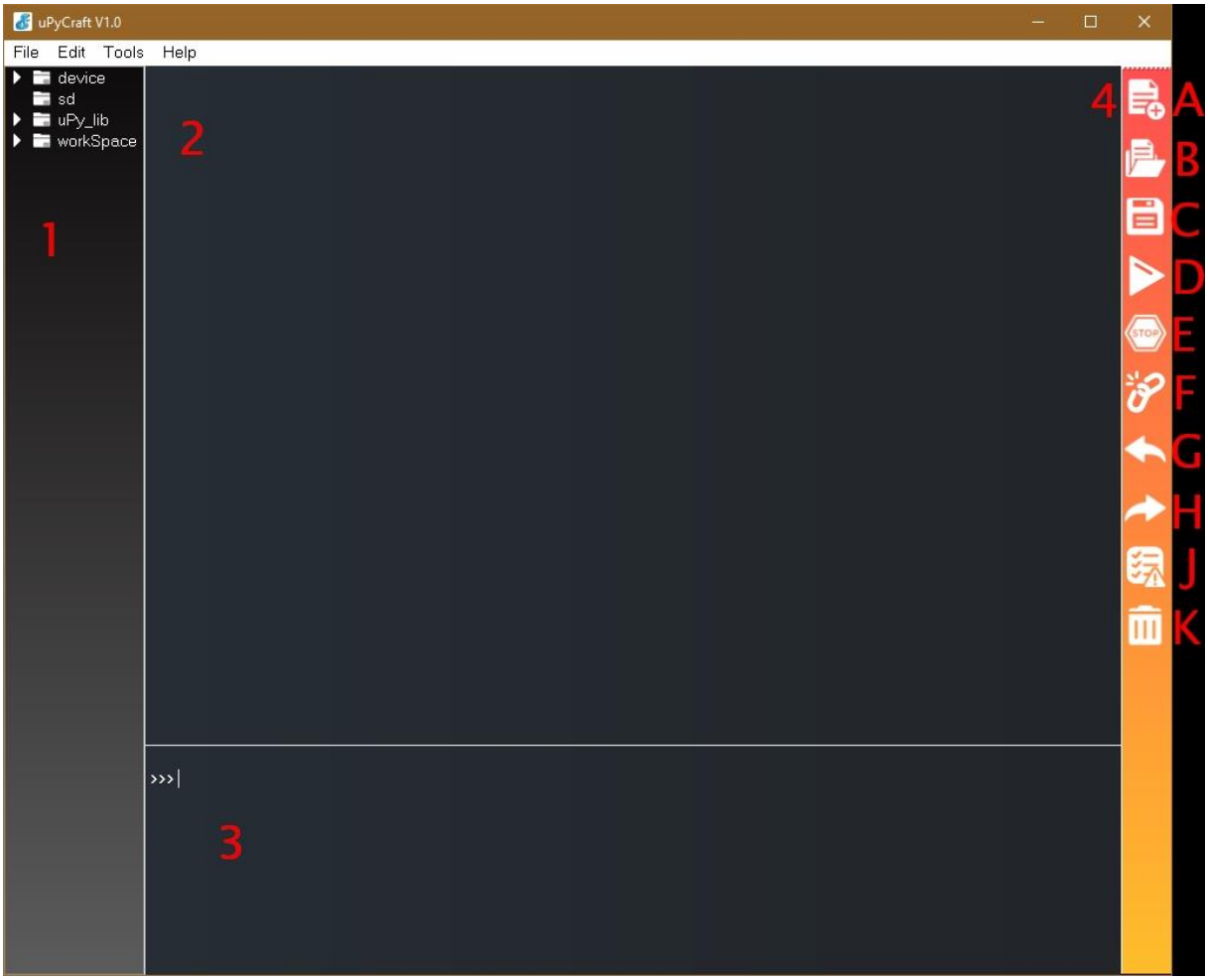
```
print("Hallo Welt")
```

```
>>> print("halloWelt")
halloWelt
>>>
```

MicroPython kann man auch als Taschenrechner benutzen.

```
>>> 234+432
666
>>>
```

Bevor wir daran gehen, Programme zu erstellen, werde ich noch kurz die wichtigsten Funktionen und Eigenheiten von µPyCraft erklären.



1 Dateifenster

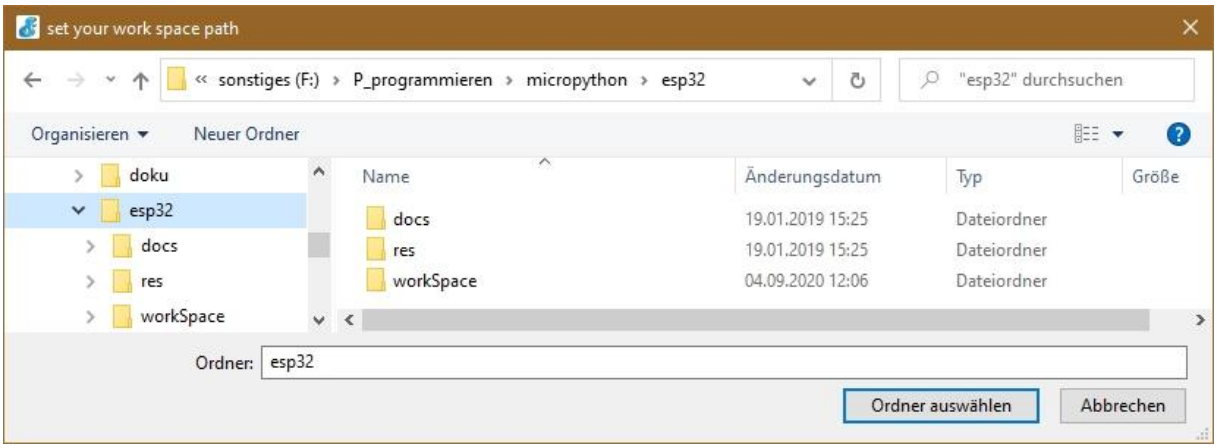
device Hier werden die Dateien aufgelistet, die sich auf dem ESP32 befinden. Man kann Dateien vom Arbeitsverzeichnis auch mit der

Maus hier hin befördern. Mit Rechtsklick wird das Kontextmenü mit weiteren Dateibefehlen aufgerufen.

sd — Hier werden die Dateien aufgelistet, die sich auf der SD-Karte befinden, falls der ESP32 einen Kartenslot hat und eine Karte eingelegt ist.

uPy_lib — Hier befinden sich die in µPyCraft integrierten Bibliotheken.

workSpace — Das ist das Arbeitsverzeichnis auf dem PC. Beim ersten Klick nach dem ersten Programmstart wird das Arbeitsverzeichnis festgelegt. Will man es ändern, klickt man im **Tools**-Menü auf **initConfig**.



2 Editorfenster

Hier werden Programme editiert, die neu erstellt, aus dem Arbeitsverzeichnis oder vom ESP32 geladen werden.

3 Terminalfenster (REPL = Read-Eval-Print-Loop)

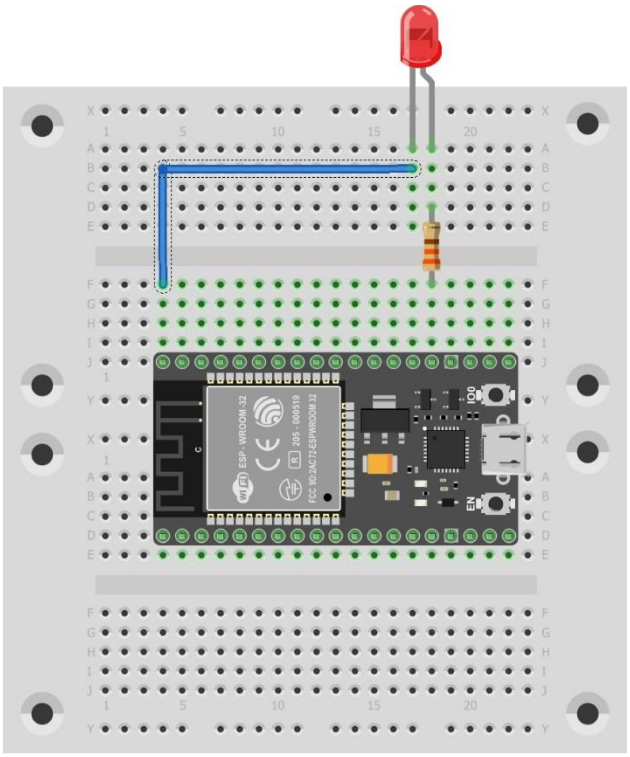
Hier kann man Direktbefehle eingeben wie auf einer Linux- oder DOS-Shell. In Windows heißt das Eingabeaufforderung. Das haben wir bereits ausprobiert.

4 Schalterleiste

Sie stellt folgende Befehle bereit.

A — Ein neues Dateifenster im Editorbereich erstellen
B — Eine Datei aus dem Arbeitsverzeichnis öffnen
C — Eine Datei im Arbeitsverzeichnis abspeichern
D — Eine Datei zum ESP32 hochladen und starten
E — Ein auf dem ESP32 laufendes Script anhalten
F — Die Verbindung zum ESP32 kappen oder aufbauen
G — Letzten Editiervorgang zurücknehmen
H — Editiervorgang wiederherstellen
J — Programm auf Syntax überprüfen
K — Terminalbereich löschen.

Nachdem der Umgang mit µPyCraft durchsichtiger geworden ist, wagen wir uns an das Bedienen von zusätzlicher Hardware am ESP32. Weil das Board nur eine LED an der TX0-Leitung (Pin1) hat, ergänze ich jetzt die Hardware um eine LED und den erforderlichen Vorwiderstand, wie es die nebenstehende Abbildung zeigt. Die LED liegt über den Widerstand am Pin GPIO2. Am Widerstand ist die Anode der LED angeschlossen, das ist das längere der beiden Beinchen. Die Kathode wird mit GND verbunden



Python und MicroPython werden ähnlich wie LUA und Perl durch Bibliotheken im Funktionsumfang erweitert. Wenn wir die GPIO-Pins nutzen wollen, müssen wir von der Bibliothek **machine** die Klasse **Pin** einbinden. Das geschieht im Terminalfenster.

```
from machine import Pin
```

Weil wir später die LED blinken lassen wollen, brauchen wir auch noch ein Delay, eine Verzögerung. Wir importieren von der Bibliothek **time** die Klasse **sleep**.

```
from time import sleep
```

Das Ganze sieht jetzt so aus.

```
>>> from machine import Pin
>>>
>>> from time import sleep
>>>
```

Wir deklarieren ein Pin-Objekt

```
led = Pin(2, Pin.OUT)
```

Das Pin-Objekt led legt den Pin 2 als Ausgang fest. Die Konstante Pin.OUT hat auch noch "Geschwister": Pin.IN, Pin.OPEN_DRAIN und weitere, mit denen man das genaue Verhalten des Pins festlegen kann. Eine vollständige Aufzählung findet man in der Dokumentation zu MicroPython.

Jetzt schalte ich Pin 2 ein, indem ich der Eigenschaft "value" des Objekts den Wert 1 oder "True" zuweise. Die Methoden des Pin-Objekts sind auch über den Link nachzulesen. Methoden werden allgemein aufgerufen, indem man deren Namen durch einen Punkt getrennt an den Namen des Objekts anhängt. Der Übergabeparameter folgt dahinter in runden Klammern.

```
led.value(1)
```

Zum Ausschalten weise ich den Wert 0 oder "False" zu.

```
led.value(False)
```

Wie in LUA, Perl und C++ ist die 0 mit False = falsch und alles was ungleich 0 ist, mit True = wahr gleichzusetzen.

Haben Sie die LED beim Absenden der letzten beiden Befehle beobachtet? Wenn Sie jetzt beide Befehle im Abstand von jeweils 0,5 Sekunden ständig wiederholen, haben Sie eine blinkende LED – nein, **so** geht das gar nicht. Aber **so** wird's was.

Die Befehle, die wir benutzt haben, werden im Editorfenster eingegeben. Achten Sie beim Schreiben darauf, dass die eingerückten Zeilen mit einem Tabulator begonnen werden und nicht mit Leerzeichen. In Python werden Anweisungsblöcke nicht wie C++, oder LUA, Perl und anderen Sprachen durch geschweifte Klammern oder ein "end" markiert, sondern durch Ein- und Ausrückung des Programmtextes.

```
from machine import Pin
from time import sleep

led = Pin(2, Pin.OUT)
i = 0
while i <=10:
  led.value(not led.value())
  sleep(0.5)
  i += 1
```

Das Umschalten des LED-Zustands erledigt die Zeile

```
  led.value(not led.value())
```

die den Zustand an Pin2 einliest, negiert und dem Ausgang wieder zuweist.

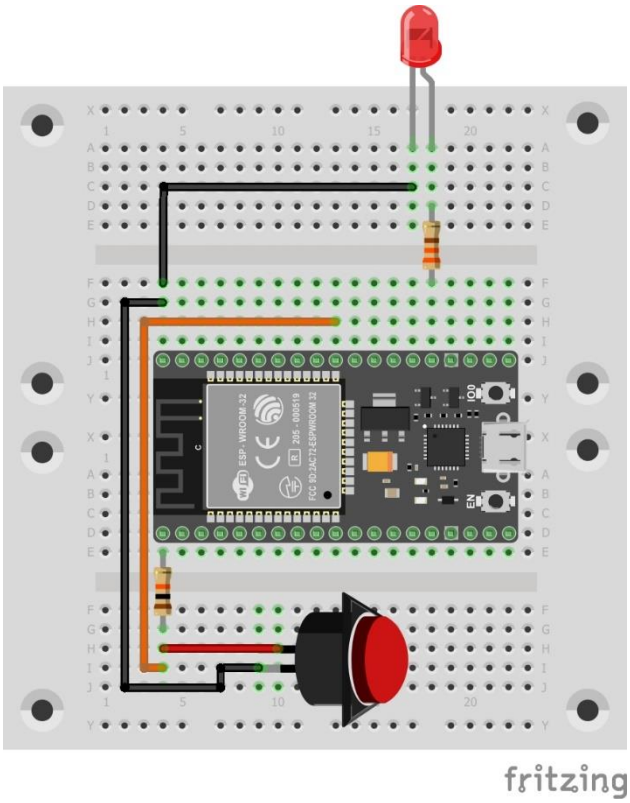
Schicken Sie dieses Script mittels des Upload-Buttons (D) zum ESP32 und nach ein paar Sekunden blinkt die LED genau 5-mal und bleibt dann angeschaltet. Die while-Struktur stellt eine Programmschleife dar. Der eingerückte Schleifenkörper wird solange wiederholt, bis der Ausdruck i <= 10 falsch wird, i also den Wert 11 erreicht. Das Hochzählen erledigt der Ausdruck i += 1, der durch i = i + 1 ersetzt werden kann.

*Blink_LED.py

```
1  from machine import Pin
2  from time import sleep
3
4  led = Pin(2, Pin.OUT)
5  i = 0
6  while i <=10:
7      led.value(not led.value())
8      sleep(0.5)
9      i += 1
```

Zum Abschluss dieses Beitrags stelle ich noch den Anschluss eines Tasters vor, der die Blinksequenz startet, wenn man ihn kurz drückt.

Wir ergänzen die Schaltung gemäß der folgenden Abbildung.



```
from machine import Pin
from time import sleep

led = Pin(2, Pin.OUT)
button=Pin(5,Pin.IN)
while True:
  taste = not button.value()
  if taste:
    print (taste)
    i = 0
    while i <=10:
      led.value(not led.value())
      sleep(0.5)
      i += 1
```

Zusätzlich zum LED-Pin wird ein zweiter Anschluss für den Tastereingang deklariert.

```
button=Pin(5,Pin.IN)
```

Damit eine Endlosschleife entsteht, wie in der Arduino-IDE die loop-Schleife, füge ich eine, den Rest umklammernde while-Struktur ein. True als Ausdruck für die Bedingung sorgt dafür, dass die Schleife bis zum Anhalten des Programms durch die Stopp-Taste (E) oder bis zu einem Reset durchlaufen wird.

```
while True:
```

Der Pullupwiderstand (10k) legt den Eingang GPIO5 auf 3,3V-Potential, welches im Programm als 1 oder High oder True identifiziert wird. Null wird dagegen als False aufgefasst. Wenn die Taste den Eingang Pin5 auf GND-Potential legt, registriert der Lesebefehl das als 0 oder False. Gerade in diesem Zustand soll aber die Blinksequenz starten. Deshalb muss der Zustand negiert werden, damit das folgende if die Blinksequenz frei gibt, wenn die Taste gedrückt ist.

```
  if taste:
```

Der print-Befehl sorgt dafür, dass im Terminalfenster der logische Zustand der Taste mitgeteilt wird.

```
    print (taste)
```

Der Rest ist bekannt.

Hauptzweck dieses abschließenden Programms ist es zu zeigen, wie Strukturen in Python aufgebaut werden. Beachten Sie daher die zunehmende Einrückung (Indentation) mit jeder eingeschachtelten Struktur. Standard sind 2 Zeichenpositionen, die der Editor in µPyCraft von sich aus durch die Tabulatortaste bereitstellt.

Zusammenfassung

Gratulation, Sie haben es geschafft.

- Sie haben erfahren, welche Hardware und Software man benötigt, um MicroPython-Projekte in Angriff zu nehmen.
- Sie wissen, wie man die verschiedenen Werkzeuge installiert und benutzt.
- Zuletzt haben Sie gelernt, die MicroPython-Umgebung zu verwenden und
- ein erstes Python-Programm auf dem Modul laufen zu lassen.
- Sie können einen Ausgangspin schalten und einen digitalen Eingang abfragen.
- Sie kennen die Bedeutung der Einrückung zum Erzeugen von Programmstrukturen.
- Mit Hilfe von µPyCraft sind Sie in der Lage, sich mit Ihrem ESP32-Modul zu unterhalten.

Im nächsten Teil geht es dann um ein weiterführendes Projekt, bei dem Sensorwerte vom ESP-Modul erfasst und über WiFi an ein Endgerät übertragen werden sollen.

Haben Sie Lust zu ein paar Hausaufgaben?

- Können Sie mit MicroPython auf Ihrem Modul ermitteln, welche Zahlenwerte hinter den Konstanten Pin.IN, Pin.OUT und Pin. OPEN_DRAIN stecken?
- Warum bleibt die LED eigentlich zum Schluss an?
- Welche Möglichkeiten gibt es, das Programm so zu verändern, dass die LED zum Schluss aus ist?
- Können Sie das Programm dazu bringen, den Wert des Durchlaufzählers i auszugeben?
- Ändern Sie die letzte Schaltung und das Programm so ab, dass das Negieren der Tasteninformation nicht nötig ist.

Viel Spaß beim Experimentieren.

Interessante und weiterführende Links

Die Startseite von MicroPython
http://micropython.org/

Die Dokumentation von MicroPython
https://docs.micropython.org/en/latest/index.html

MicroPython auf GitHub
https://github.com/micropython