# Python & MicroPython: Programmieren lernen für Einsteiger

## Erste Schritte zur Hardwaresteuerung

Teil 1

**Mit diesem Beitrag soll eine Einführung in die universelle und moderne Programmiersprache „Python“ gestartet werden. Der Fokus wird dabei auf der Hardwaresteuerung und hardwarenahen Programmierung liegen. Die Gründe, warum Python als Programmiersprache immer beliebter wird, sind vielfältig. Einer der wichtigsten Punkte ist sicher die einfache Erlernbarkeit der Sprache. Zudem verfügt Python über eine klare und einfache Syntax, die einerseits leicht zu erfassen und andererseits auch gut in Projekten umzusetzen ist.**

## Python: kompakt und universell einsetzbar

In Python sind meist deutlich weniger Codezeilen im Vergleich zu anderen Programmiersprachen erforderlich. Dadurch wird die Verständlichkeit verbessert und die Entwicklung von Anwendungen beschleunigt. Zudem kann Python für die verschiedensten Applikationen eingesetzt werden, von der Datenanalyse bis hin zu Künstlicher Intelligenz, von Hausautomatisierung bis hin zur komplexen Hardwaresteuerung. Die Sprache bietet eine breite Palette von Bibliotheken und Frameworks, welche die Entwicklung erleichtern.

Darüber hinaus bietet Python eine aktive Entwickler-Community, die kontinuierlich neue Module, Pakete und Tools erstellt. Es gibt eine Fülle von Ressourcen, Tutorials und Dokumentationen, die dabei helfen, Programmierkenntnisse aufzubauen und zu erweitern.

Diese Gründe machen Python zu einer attraktiven Option für Anfänger und erfahrene Entwickler gleichermaßen. Es ist eine vielseitige Sprache, die für eine Vielzahl von Anwendungen geeignet ist und eine starke Unterstützung in der „Community" genießt.

## Überblick und verwendete Programmstrukturen

Neben einer grundlegenden Einführung werden in diesem Beitrag die folgenden Programmstrukturen vorgestellt:

- **Importieren von Bibliotheken**
- **Zeitsteuerung mit sleep**
- **Hardware Steuerung mit**
  - **setmode**
  - **setup**
  - **output**
- **„try/except"-Konstruktion**

## Voraussetzungen und Hardwaregrundlagen

Um die Praxisbeispiele des Kurses erfolgreich umsetzen zu können, sollten die folgenden Voraussetzungen erfüllt sein:

- Sicherer Umgang mit dem Betriebssystem Windows
- Grundlegendes Wissen zum Themenkreis „Elektronik", insbesondere im Bereich „Strom – Spannung - Widerstand"
- Elementare Kenntnisse einer beliebigen Programmiersprache wie Basic oder C

Der Fokus wird auf Hardwaresteuerung und Praxisanwendungen liegen. Typische Programmstrukturen wie „If", „While" etc. sollten daher vom Prinzip her bekannt sein. Bei Bedarf können diese Informationen unter [1] nachgelesen werden. Grundkenntnisse im Umgang mit Linux sind vorteilhaft, aber nicht unbedingt erforderlich, da notwendige Befehle bei den entsprechenden Anwendungen erläutert werden.

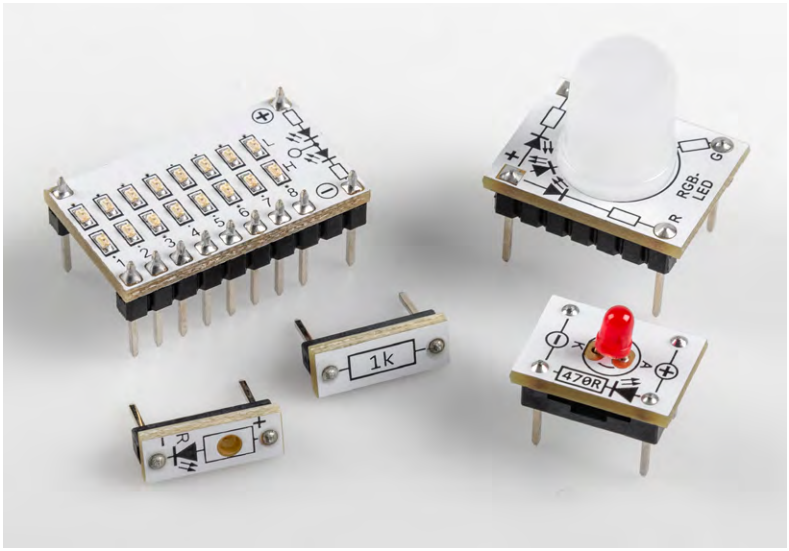



*Bild 1: Einige Komponenten für den praktischen Einsatz unter Python*

Für spezielle Kenntnisse im Bereich Elektronik sei hier auf die umfangreiche Fachliteratur verwiesen. Hinweise dazu finden sich unter „Weitere Infos" am Ende dieses Artikels.

Die Hardware-Aufbauten sind bewusst einfach gehalten, damit sie ohne Probleme auf Laborsteckboards (Breadboards) aufgebaut werden können. Damit auch Einsteiger alle Praxisanwendungen funktionssicher nachbauen können, sollen so weit wie möglich Komponenten aus den sogenannten Prototypen-Adaper-Sets (PADs) zum Einsatz kommen. Diese bieten aufgrund ihrer Beschriftung, ihrer hervorragenden elektrischen Eigenschaften und ihres kompakten Aufbaus eine hervorragende Grundlage für die technische Seite des Programmierkurses.

Bild 1 zeigt einige relevante Komponenten aus dieser Serie. Welche Bauelemente bzw. PAD-Version im einzelnen erforderlich sind, wird bei den entsprechenden Anwendungsbeispielen angegeben.

Darüber hinaus erforderlich sind:

- ein PC oder Laptop mit USB-Schnittstelle
- das Betriebssystem Windows 10
- ein Internetzugang

## Raspberry Pi und Pico als Python-Controller

Alle Programmieraufgaben sollen auf einem Raspberry Pi umgesetzt werden. In späteren Anwendungen wird dann auch ein Raspberry Pi Pico zum Einsatz kommen.

Der Pi ist für diese Aufgaben besonders geeignet, da Python die bevorzugte Sprache dieses Boards ist. Alle erforderlichen Softwarekomponenten stehen daher standardmäßig zur Verfügung. Zudem weist der Pi eine Reihe von elektrischen Ein- und Ausgängen (I/O-Ports) auf, die den direkten Anschluss elektronischer Komponenten ermöglichen.

Ausführliche Informationen zum Raspberry Pi bzw. zum Pico finden sich z. B. in [2].

## Thonny als Programmierumgebung

Die Thonny Python IDE (Integrated Development Environment) ist der Standard-Editor zum Programmieren mit Python auf dem Raspberry Pi.

Die IDE ist im Raspberry Pi OS bestens integriert und bietet viele Vorteile:

- Der integrierte Text-Editor unterstützt Syntax-Highlighting für Python und MicroPython.
- Für das Datei-Management existiert eine eigene Ansicht. Funktionen wie Speichern und Löschen von Dateien können direkt in der IDE ausgeführt werden.
- Für die Text-Eingabe und -Ausgabe existiert die Ansicht einer Kommandozeile/Konsole/Shell.
- Externe Bibliotheken können über einen integrierten Paket-Manager installiert werden.
- Wird ein neuer Raspberry Pi Pico ohne MicroPython-Firmware erkannt, wird die Installation automatisch angeboten etc.

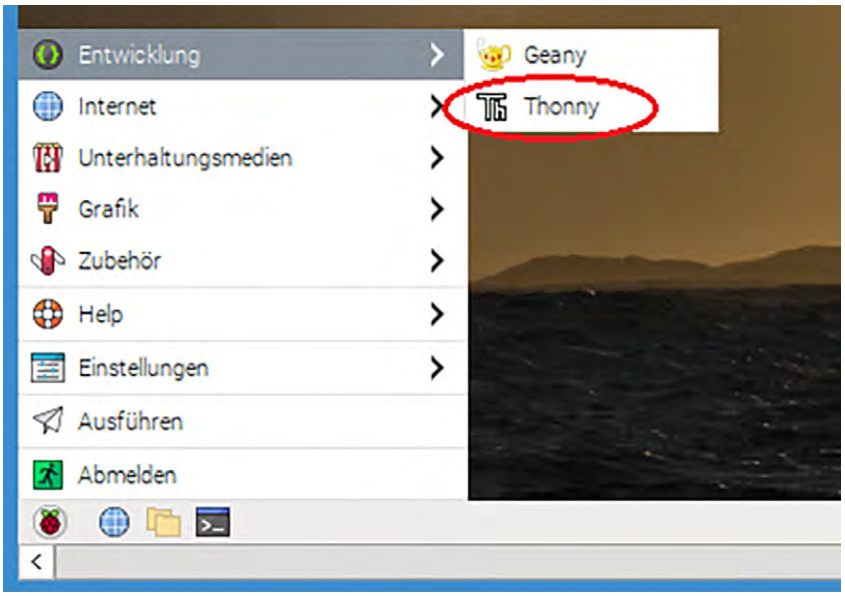



*Bild 2: Starten der Python IDE Thonny*

Bild 2 zeigt die Thonny IDE nach dem Start. Für die Arbeit mit Thonny stehen verschiedene Funktionen zur Verfügung. Die Nummern in den roten Kreisen in Bild 3 zeigen die Funktionseinheiten der IDE:

① Die Menüleiste enthält wichtige Funktionen zur Dateiverwaltung und zum Starten und Stoppen des aktuell sichtbaren Programmcodes.

② Das Textfeld für den Quelltext bzw. Programmcode ist ein Text-Editor mit Syntax-Highlighting.

③ Eingabe- und Ausgabefeld für Text-Informationen während des Programmablaufs, das man als Kommandozeile, Konsole, Shell oder Eingabeaufforderung bezeichnet

④ Übersicht zum aktiven Dateiordner

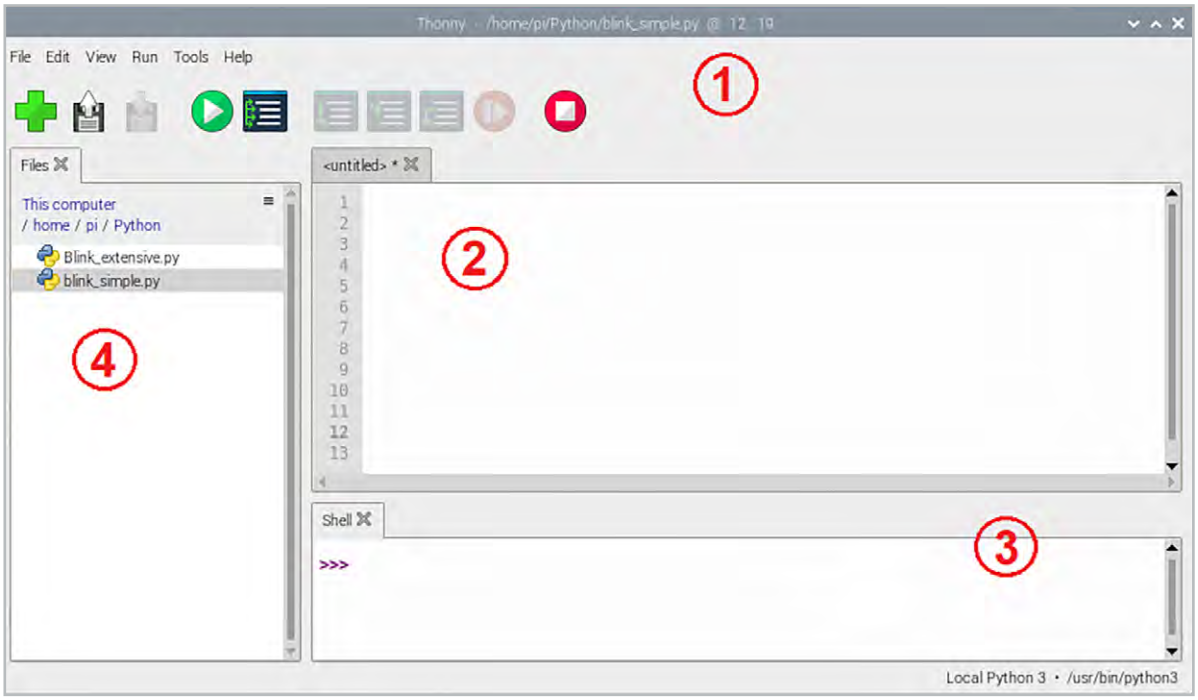



*Bild 3: Die Funktionen der Thonny IDE*

Durch ihren einfachen Aufbau und die klare Struktur wurde die Thonny IDE zum Quasi-Standard für die Python-Programmierung. Deshalb soll sie auch im Rahmen dieser Artikelserie zum Einsatz kommen.

## Experimentiersysteme

Für die praktische Umsetzung der Hardwarebeispiele bietet sich der Einsatz von hochwertigen Breadboards an. Dieses System eignet sich bestens zum Aufbau der erforderlichen Schaltungen. Bild 4 zeigt exemplarisch einen typischen Aufbau mit einer LED aus dem Set von PAD6 als High-/Low-Indikator (CMOS-Logik).

Die einfache Kombination aus Raspberry Pi und Breadboard-Aufbauten ist für viele Praxisanwendungen bereits ausreichend. Allerdings weist diese Variante einige Nachteile auf. Sie ist mechanisch nicht optimal, wenn der Aufbau z. B. einmal bewegt

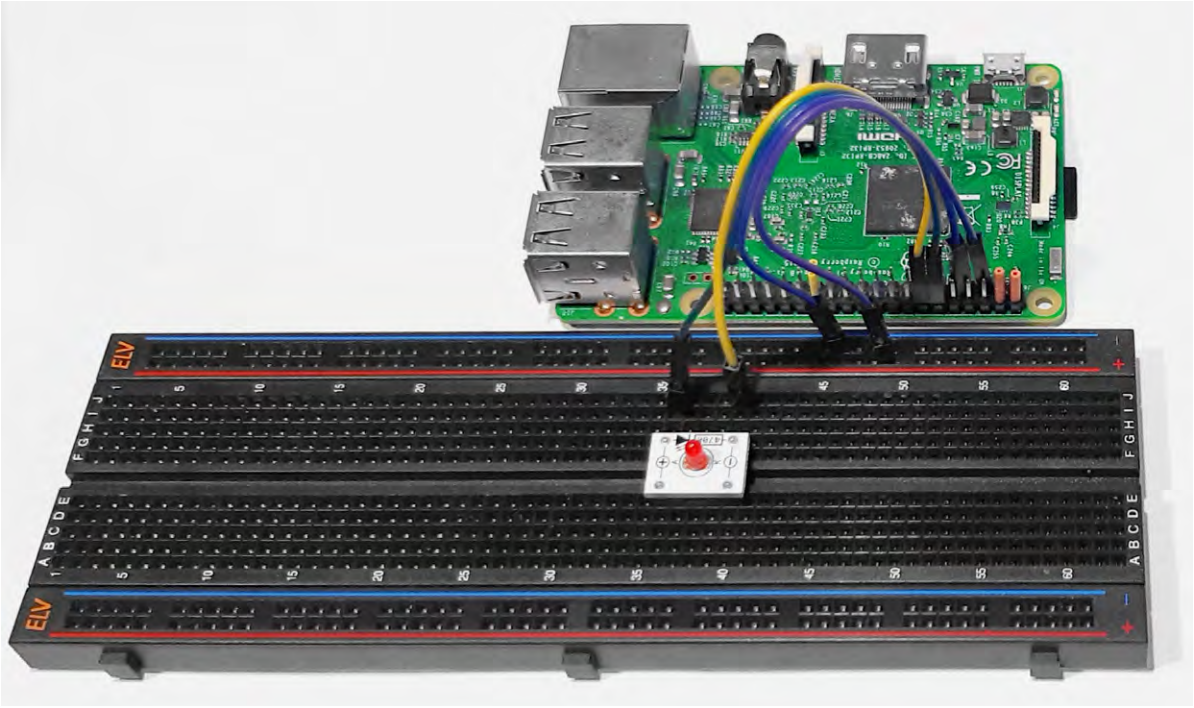


Bild 4: Aufbau mit Raspberry Pi und Breadboard-Schaltung

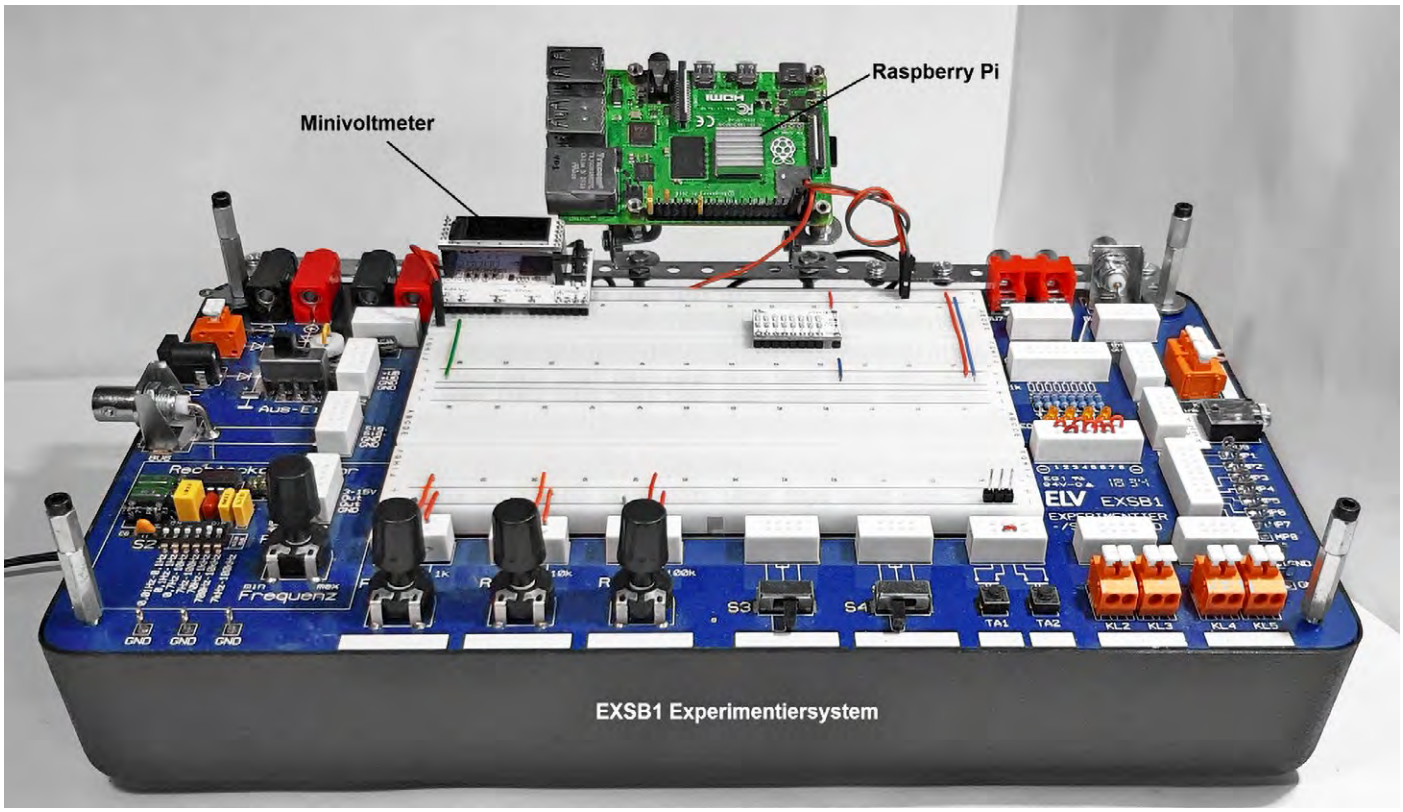



Bild 5: Raspberry Pi am EXSB1-Experimentierboard

werden muss. Zudem stehen keine Bedienelemente wie Schalter, Taster oder Potentiometer etc. direkt zur Verfügung.

Ein komplettes Experimentiersystem wie das EXSB1 löst diese Probleme. Wird zudem der Raspberry Pi mit einigen mechanischen Elementen oben an diesem System fixiert, steht ein nahezu ideales Aufbausystem zur Verfügung, mit dem die elektronische Seite des Python-Kurses zum reinen Vergnügen wird.

Bild 5 zeigt einen entsprechenden Aufbauvorschlag. Natürlich können für die Befestigung des Raspberry Pi auch andere Varianten zum Einsatz kommen.

In Bild 5 ist zusätzlich noch ein Mini-Voltmeter-Modul zu sehen. Dieses ist für die ersten Experimente nicht unbedingt erforderlich, es kann allerdings für viele Anwendungen vorteilhaft eingesetzt werden (siehe Abschnitt „Mini-Voltmeter als Logic-Analyzer“).

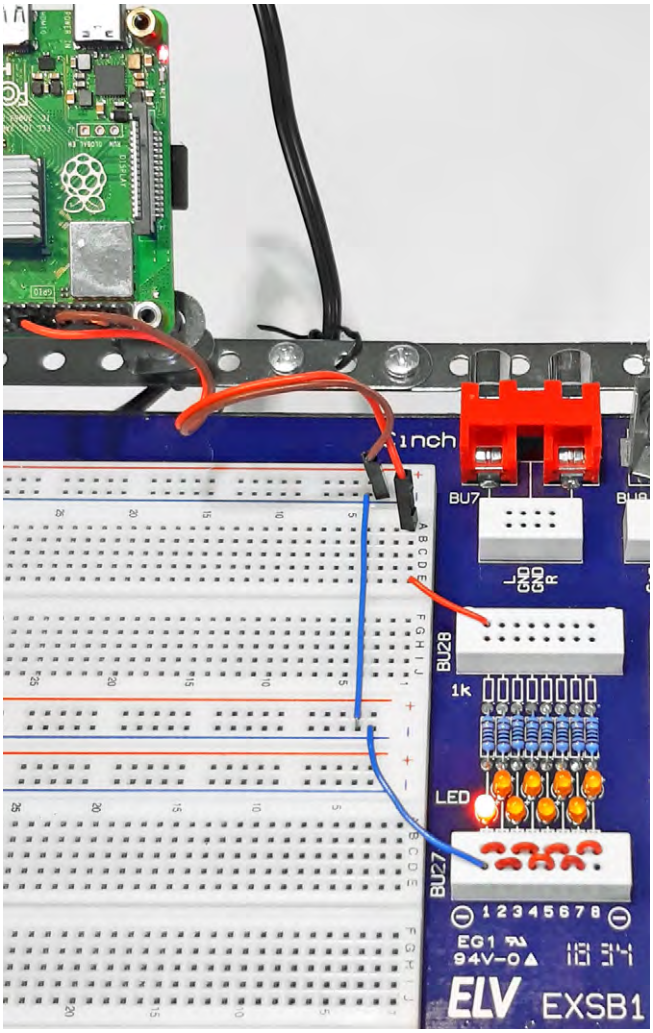



*Bild 6: Hardware-Aufbau zur Ansteuerung einer LED mit Python*

## LEDs und Logik-Level-Anzeige im Einsatz

In einem ersten Praxisbeispiel sollen die Funktion der I/O-Ports des Raspberry Pi überprüft und erprobt werden. Wenn das EXSB1-System zur Verfügung steht, kann dazu eine der auf dem Board vorhandenen LEDs verwendet werden. Dies hat den Vorteil, dass der erforderliche Vorwiderstand ebenfalls bereits vorhanden ist. Bild 6 zeigt den entsprechenden Aufbau dazu.

Falls die Schaltung auf einem einzelnen Breadboard aufgebaut werden soll, ist zu beachten, dass die LED mit einem strombegrenzenden Widerstand zwischen mindestens 150 Ω bis 1 kΩ geschützt werden muss. Alternativ können auch LED-Module aus den PAD-Adaptersets verwendet werden. Diese verfügen über einen bereits integrierten Vorwiderstand auf der Platine.

Zur Verdeutlichung zeigt Bild 7 den Aufbau nochmals als realitätsnahes Schaltbild.

## Ein erstes Python-Programm

Das folgende Python-Programm sorgt dafür, dass die LED im Sekundentakt blinkt (blink_LED_1_Hz.py):

```
import RPi.GPIO as GPIO
import time

led_pin = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)

while True:
  GPIO.output(led_pin, GPIO.HIGH)
  time.sleep(0.5)
  GPIO.output(led_pin, GPIO.LOW)
  time.sleep(0.5)
```

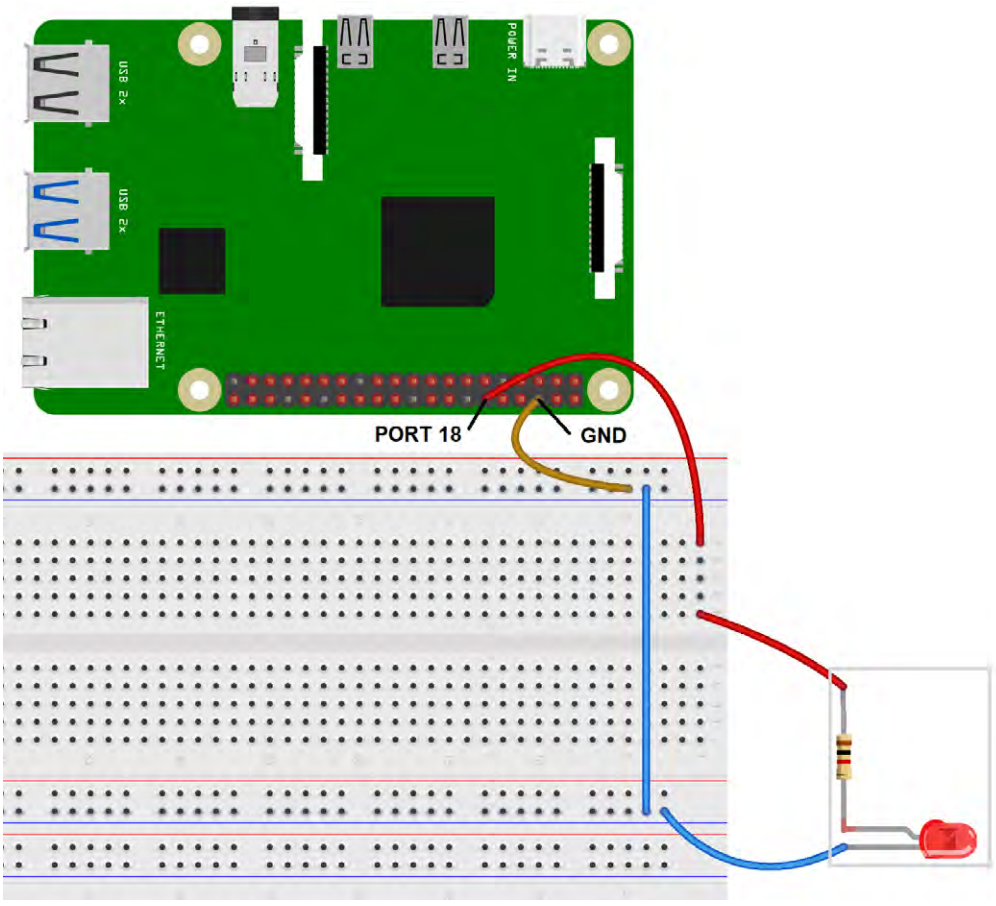
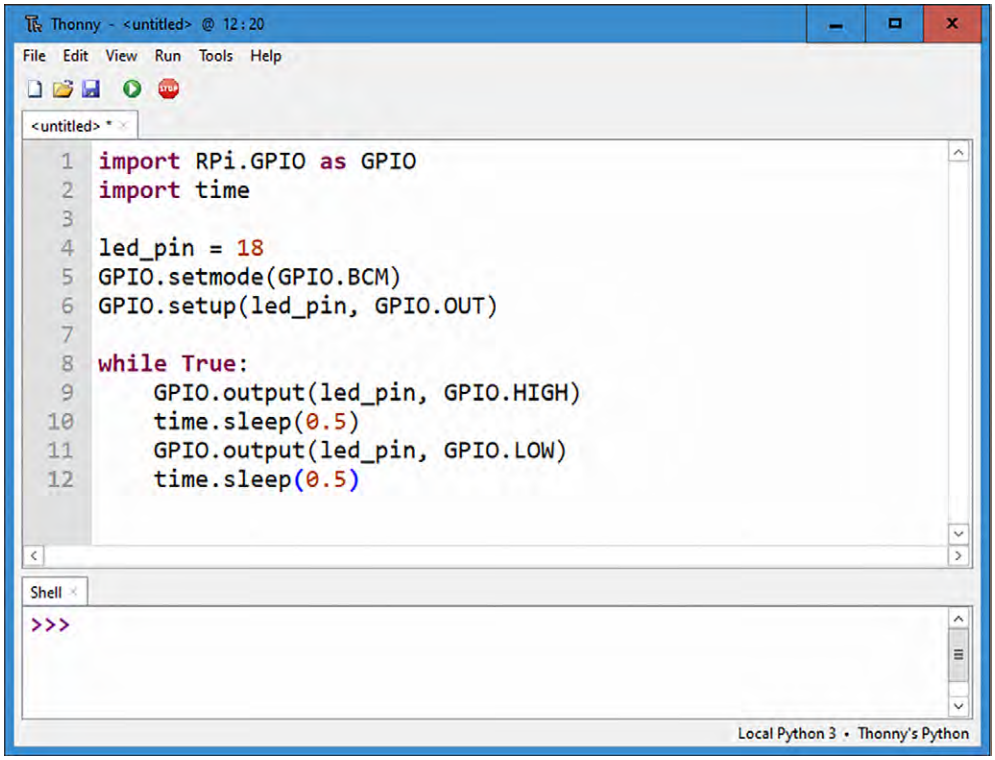



*Bild 7: LED-Steuerung als realitätsnahes Schaltbild*

Bild 8: Python-Programm in der Thonny IDE




Alle Programme und Skripts etc. sind im Download-paket zu diesem Artikel enthalten. Das Programm kann also entweder (zu Übungszwecken) abgetippt werden oder man lädt es direkt via Internet-Download auf den Raspberry Pi. Alternativ kann das Programm z. B. auch als Textdatei mit einem USB-Stick auf den Raspberry Pi übertragen werden. In der Thonny IDE sieht das Programm so aus, wie in Bild 8 zu sehen.

Nach dem Starten des Programms (über das grüne Start-Symbol mit dem weißen Pfeil) beginnt die angeschlossene LED zu blinken.

## Wie arbeitet der Code?

Die erste Zeile des Programms importiert das RPi.GPIO-Modul, das benötigt wird, um auf die GPIO-Pins zuzugreifen. Dann erfolgt der Import des time-Moduls, das u. a. Programmpausen ermöglicht.

Die nächsten Zeilen setzen die Variable „led_pin" auf 18 und legen den Modus der GPIO-Pins auf BCM („Broadcom"-Nummerierung) fest. Die Zeile GPIO.setup(led_pin, GPIO.OUT) konfiguriert den led_pin als Ausgangspin (Output).

Die folgende „while"-Schleife wird ohne Ende durchlaufen, da „True" als Schleifenbedingung angegeben ist. Innerhalb der Schleife wird zuerst der led_pin auf HIGH gesetzt, um die LED einzuschalten, dann wird time.sleep(0.5) aufgerufen, um eine Pause von 0,5 Sekunden einzulegen.

Anschließend wird der led_pin auf LOW gesetzt, um die LED auszuschalten, dann folgt erneut eine Pause von 0,5 Sekunden. Die Schleife wird dann von vorne gestartet, und der Zyklus wiederholt sich, wodurch die LED abwechselnd ein- und ausgeschaltet wird. Das Programm erzeugt also eine Blinksequenz, bei der die LED alle 0,5 Sekunden ein- und ausgeschaltet wird.

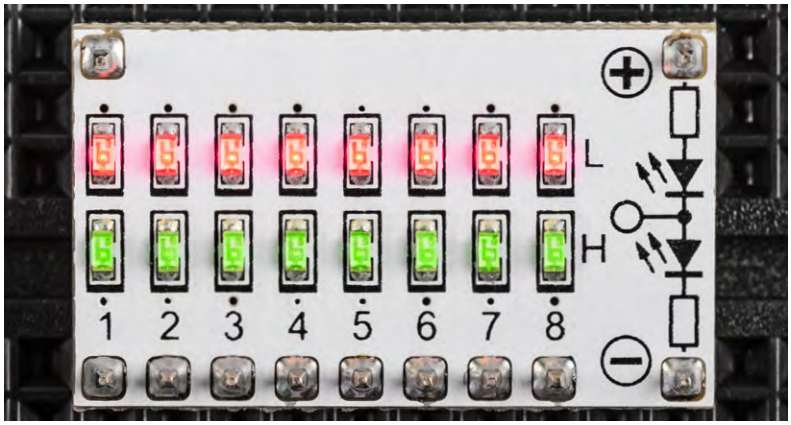



Bild 9: Undefinierter Pegel am Logic-Level-Modul

Die Blinkfrequenz der LED beträgt dabei:

$$f = 1/T = 1/(0{,}5\ s + 0{,}5\ s) = 1/1\ s = 1\ Hz$$

Nach dem Programmstart blinkt die LED daher im gut erkennbaren Sekundentakt.

Obwohl das Programm einwandfrei arbeitet, wird in der Konsole eine Warnmeldung ausgegeben:

**... RuntimeWarning: This channel is already in use ...**

Diese weist darauf hin, dass die IO-Ports eventuell bereits in früheren Anwendungen zum Einsatz gekommen sind. In dieser einfachsten Programmversion wurde zunächst auf eine klare Definition des Pin-Zustands beim Programmstart verzichtet. Das Programm arbeitet jedoch trotz der Warnung problemlos. Weitere Hinweise hierzu finden sich im nächsten Abschnitt.

Anstelle der einfachen LED kann auch eine Logic-Level-Anzeige verwendet werden. Ein entsprechendes Modul ist im PAD6 enthalten. Diese Pegelanzeige wird sowohl mit der Betriebsspannung des Raspberry Pi von 3,3 V als auch mit Ground (GND) verbunden. Ohne weitere Beschaltung leuchten die acht roten und acht grünen LEDs des Moduls schwach auf und zeigen damit einen „undefinierten" Logikpegel an (Bild 9).

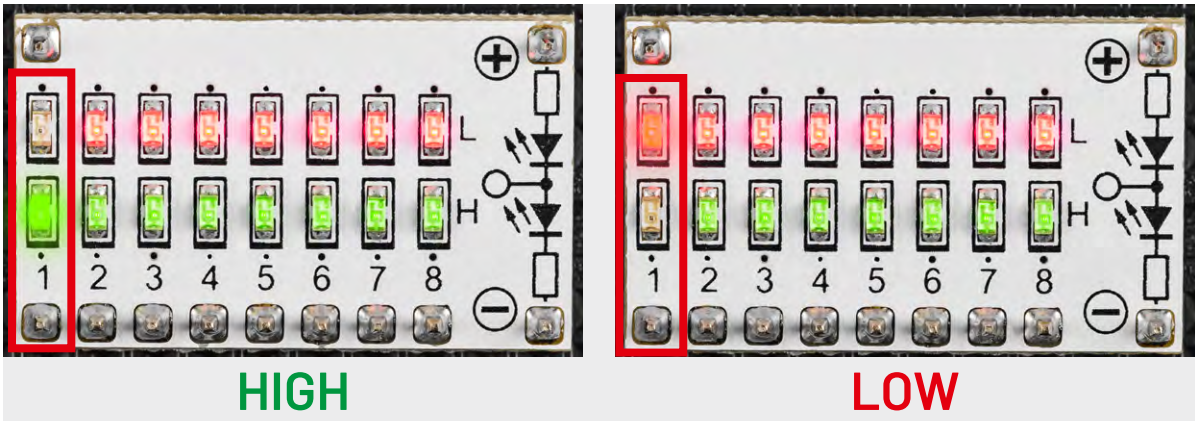



*Bild 10: Pegel HIGH (3,3 V) und LOW (0 V) am Logic-Level-Modul*

Wird der aktive Ausgangspin des Raspberry Pi (I/O Nummer 18) mit dem ersten Kanal der Logic-Level-Anzeige verbunden, blinken die rote und grüne LED dieses Kanals im Wechseltakt. Damit wird der Pegelwechsel des Pins (0 V: LOW-Pegel, 3,3 V: HIGH-Pegel) sehr anschaulich demonstriert (Bild 10).

## Programme definiert beenden

Im Gegensatz zu den anderen sieben Kanälen hat der Eingang nun ein definiertes HIGH/LOW-Signal, das entsprechend angezeigt wird. Wird das Programm über das Stopp-Symbol (roter Kreis mit weißem Quadrat in der Thonny IDE) oder über die Tastenkombination Ctrl-C (bzw. Strg-C) angehalten, bleibt entweder die rote oder die grüne LED dauerhaft eingeschaltet. Welche LED aktiv bleibt, hängt davon ab, in welchem Zustand der Port beim Beenden aktiv war.

Dieses Verhalten ist in vielen Anwendungsfällen unerwünscht. Wenn auf diese Weise z. B. eine Maschine gesteuert wird, ist nach einem Programmabbruch der genaue Zustand des Geräts nicht definiert. Dies kann bei größeren Systemen zu gefährlichen Situationen führen.

Es ist daher sinnvoll, das Programm (try_except.py) folgendermaßen zu erweitern:

```
import RPi.GPIO as GPIO
import time

LedPin = 18

GPIO.setmode(GPIO.BCM)          # Numbers GPIOs by physical location
GPIO.setup(LedPin, GPIO.OUT)    # Set LedPin's mode is output

try:
 while True:
 print (, ...led on')
 GPIO.output(LedPin, GPIO.LOW)  # led on
 time.sleep(0.5)
 print (, led off...')
 GPIO.output(LedPin, GPIO.HIGH) # led off
 time.sleep(0.5)

except KeyboardInterrupt:       # When , Ctrl+C' is pressed, the following code will be executed.
 GPIO.output(LedPin, GPIO.LOW)  # led off
   GPIO.cleanup()           # Release resource
   print (, bye...')
```

Hier wird die Hauptschleife in eine „try/except"-Konstruktion eingeschlossen. Im „except"-Zweig wird ein Keyboard-Interrupt definiert, der einen kontrollierten Programmabbruch ermöglicht. Wenn das Programm über Ctrl-C beendet wird, wird der Port in den LOW-Zustand versetzt. Die Anweisung „GPIO.cleanup()" sorgt zudem dafür, dass die verwendeten Ressourcen, also die GPIO-Pins, programmtechnisch wieder freigegeben werden.

Zusätzlich werden in der Konsole weitere Informationen ausgegeben. Über die „print"-Anweisung wird angezeigt, ob die LED gerade ein- oder ausgeschaltet ist. Am Ende liefert die Ausgabe „bye..." in der Konsole die Information, dass das Programm ordnungsgemäß beendet wurde.

Auf der Hardwareseite ist dies daran erkennbar, dass alle LEDs des Logic-Pegel-Moduls nach Programmende via Ctrl-C nur noch schwach leuchten. Das bedeutet, dass der verwendete Ausgang nun hochohmig, also inaktiv ist. Dies ist typischerweise der erwünschte Zustand nach dem Beenden eines Programms. Wird das Programm dagegen über das Stopp-Symbol abgebrochen, entsteht wieder der bereits bekannte undefinierte Zustand. Eine weitere Wirkung einer definierten Programmbeendigung zeigt sich beim Neustart. Wurde das Programm mit Ctrl-C beendet, wird kein Warnhinweis mehr ausgegeben, da die Systemressourcen in diesem Fall ordnungsgemäß freigegeben wurden. Beim Abbruch über das Stopp-Symbol wird dagegen nach einem Neustart wieder die bekannte Warnung angezeigt.

## Mini-Voltmeter als Logic-Analyzer

Das Mini-Voltmeter ist ein äußerst nützliches Modul, wenn es darum geht, Spannungspegel schnell und präzise zu erfassen. Bild 11 zeigt das Modul mit den Betriebsspannungen (5 V und 3,3 V) des Raspberry Pi.

Neben den Betriebsanzeigen kann das Modul jedoch auch noch weitere wichtige Aufgaben übernehmen. So ist es bei der Arbeit mit dem Raspberry Pi oftmals erforderlich, auch den zeitlichen Verlauf von Spannungspegeln zu erfassen. Auch hier kann das Mini-Voltmeter nutzbringend eingesetzt werden.

Bild 12 zeigt den Spannungsverlauf am Port 18, wenn das obige Programm aktiv ist. Zur Verdeutlichung wurden die Schaltzeiten auf 1,5 Sekunden erhöht. Nun kann man den zeitlichen Spannungsverlauf, also das Umschalten zwischen 0 V und 3,3 V sehr gut erkennen. Das Modul dient hier als einfacher Logic-Analyzer. Der erste Kanal (Input 1) zeigt dagegen nach wie vor die konstante Eingangsspannung des Raspberry Pi von 5,0 V an.

## Fazit und Ausblick

Im ersten Teil dieses Python-Grundkurses wurde der Start der Programmierumgebung vorgestellt und erläutert. Die ersten Programme ermöglichten bereits den Zugriff auf einfache Hardwarekomponenten wie LEDs.

Im nächsten Beitrag werden die I/O-Ports genauer unter die Lupe genommen. Es wird gezeigt, dass die Pins des Raspberry Pi nicht nur als Ausgänge, sondern auch als Eingänge deklariert werden können. Damit lassen sich Tastensteuerungen oder Reed-Relais-Schaltungen aufbauen.

Es werden auch die Probleme des Tastenprellens genauer betrachtet und Lösungsvorschläge diskutiert. Zudem wird die Strombelastbarkeit der Pins untersucht. Genauere Kenntnisse zu diesem Thema können helfen, Überlastungen des Raspberry Pi zu verhindern. ELV

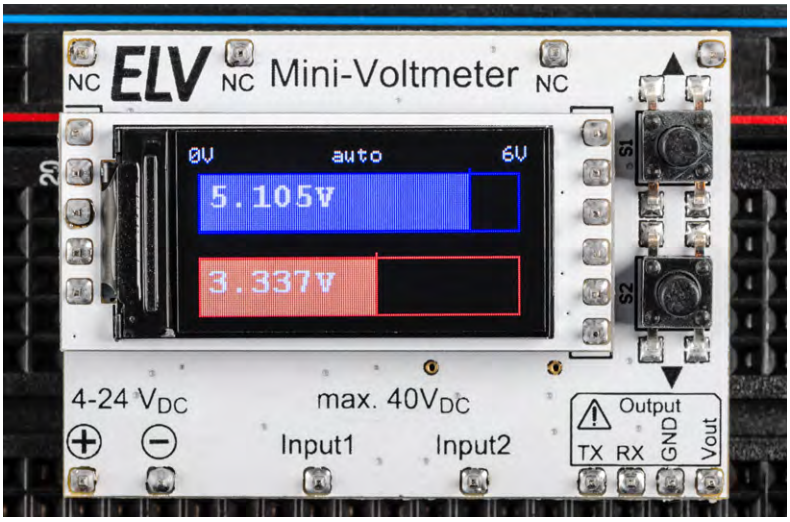



*Bild 11: Die Betriebsspannungen des Raspberry Pi auf dem Mini-Voltmeter*

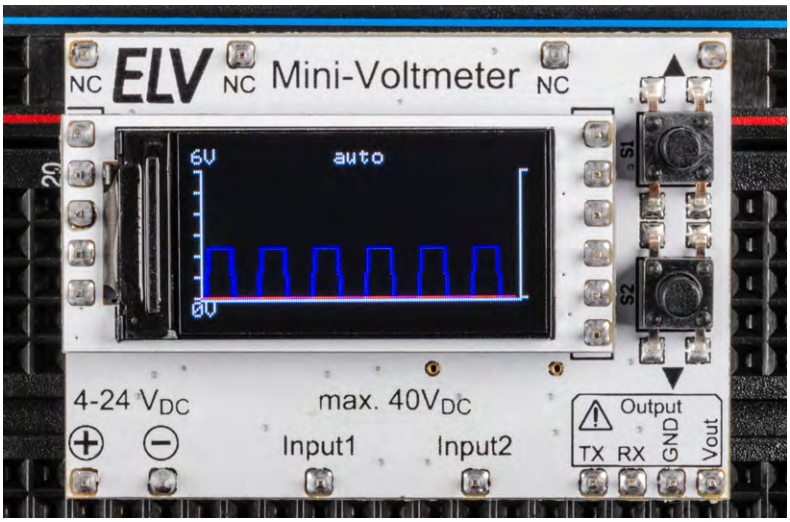



*Bild 12: Logikpegelverlauf an Port 18 auf dem Mini-Voltmeter*

## Weitere Infos

[1] G. Spanner, MicroPython, Elektor Verlag (2020)
[2] G. Spanner, Raspberry Pi 4 und Pico – Pfiffige Projekte zum Messen, Steuern und Regeln, Elektor Verlag (2023)

Download-Paket

| Material | Artikel-Nr. |
|---|---|
| z. B. Raspberry Pi 4 Model B, 8GB RAM | 250567 |
| Experimentier-/Steckboard EXSB1 | 153753 |
| Mini-Voltmeter für Steckboards mit sTFT-Display, MVM1 | 156596 |
| LEDs mit Vorwiderständen sind z. B. im PAD2 (linear) enthalten | 154712 |
| Logik-Pegel-Module sind z. B. im PAD6 (CMOS-Logik) enthalten | 155858 |
| Kabelset | 129019 |