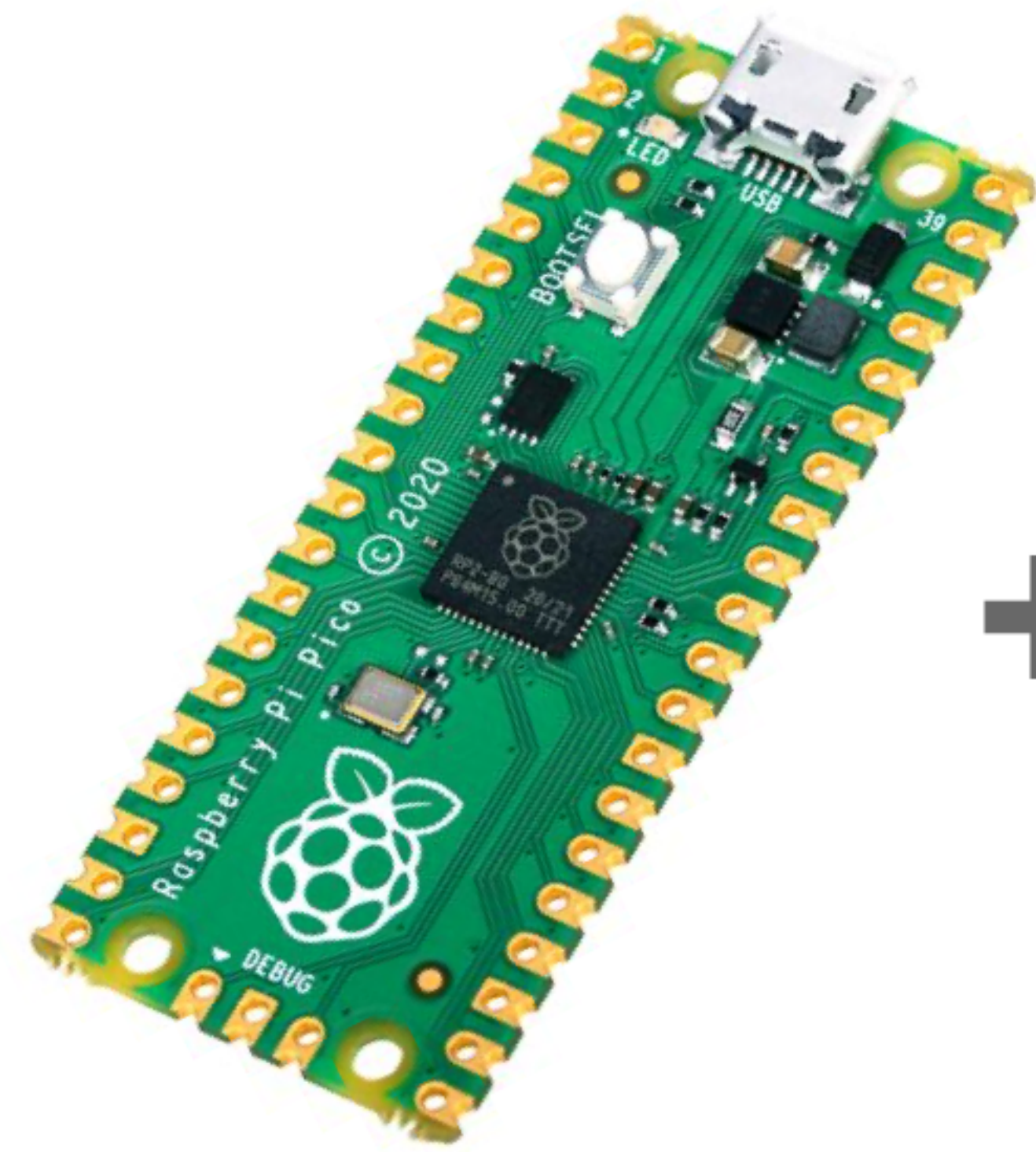
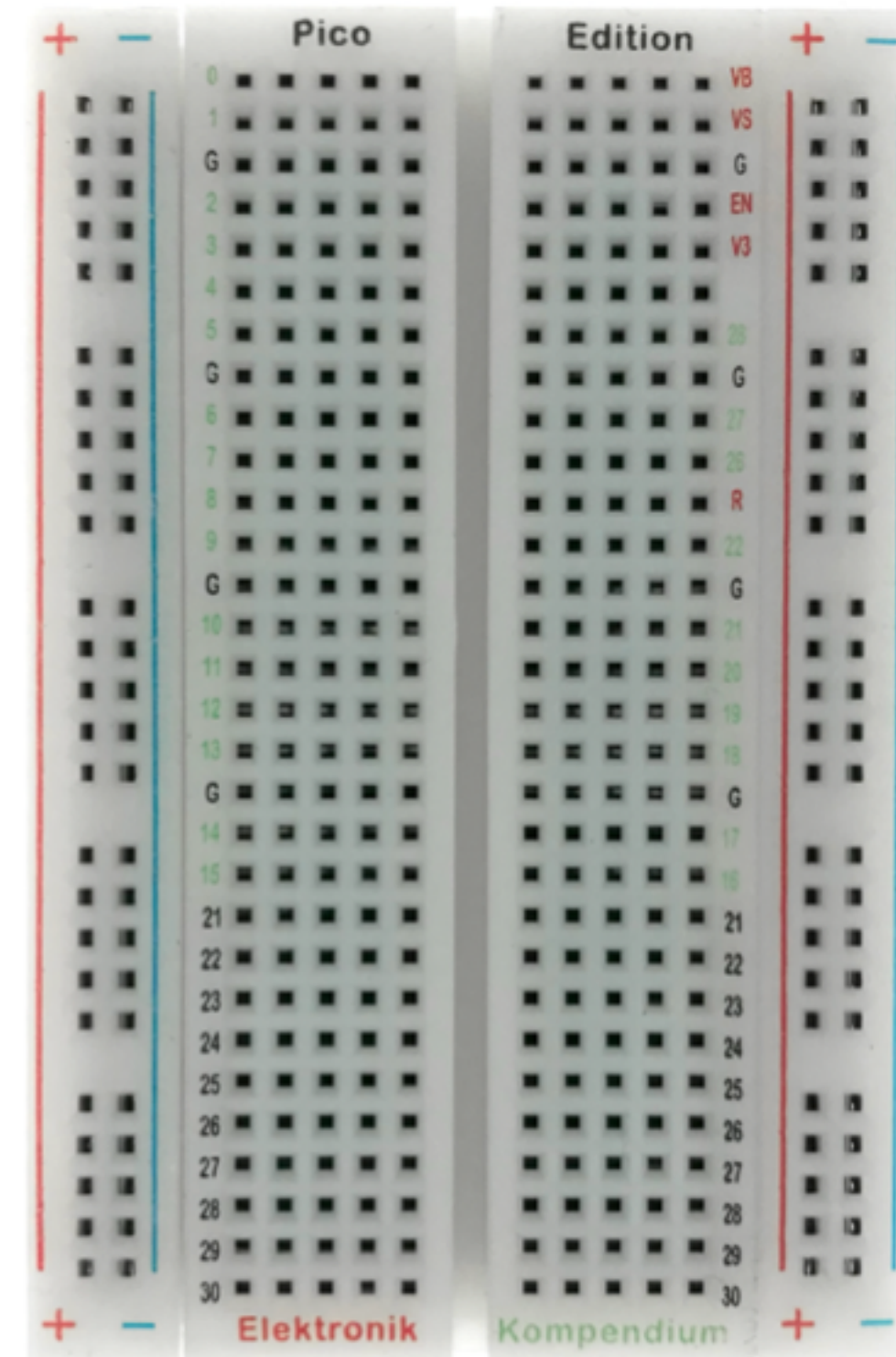


+ **MicroPython**



+



+



Elektronik-Guide Pico Edition

Impressum

Elektronik-Guide Pico Edition

Version: 2023-08-04

Herausgeber:

Patrick Schnabel

Droste-Hülshoff-Str. 22/4

71642 Ludwigsburg

Deutschland

USt-ID-Nr.: DE207734730

WEEE-Reg.-Nr.: DE80632679

<https://www.elektronik-kompendium.de/>



Dieses Elektronik-Set wurde nach den geltenden europäischen Richtlinien entwickelt und hergestellt. Der bestimmungsgemäße Gebrauch aller Bauteile ist in dieser Anleitung beschrieben. Der Nutzer ist für den bestimmungsgemäßen Gebrauch und Einhaltung der geltenden Regeln verantwortlich. Bauen Sie die Schaltungen deshalb nur so auf, wie es in dieser Anleitung beschrieben ist. Das Elektronik-Set darf nur zusammen mit dieser Anleitung weitergegeben werden.



Das Symbol mit der durchkreuzten Mülltonne bedeutet, dass dieses Produkt nicht mit dem Hausmüll entsorgt, sondern als Elektroschrott dem Recycling zugeführt werden muss. Mit dem Kauf dieses Produkts wurden die Gebühren für die Entsorgung entrichtet. Die nächstgelegene kostenlose Annahmestelle für Elektroschrott erfahren Sie von Ihrer regional zuständigen Abfallwirtschaft.

Vorwort (1)

Die Kombination aus Elektronik, Software und Programmierung ist ein spannendes Experimentierfeld. Aber auch nicht ganz so einfach. Denn wenn etwas nicht funktioniert, wo liegt dann der Fehler? In der Verschaltung der Bauteile oder im Programmcode? Bei dieser doppelten Komplexität wird aus Lust schnell Frust. Das weiß ich aus Erfahrung.

Wir brauchen also möglichst einfache Schaltungen und ganz kurze und leicht lesbare Quelltexte. Nach Möglichkeit solltest Du nicht zu viel falsch machen können, oder? Die Idee dieses Elektronik-Sets ist es, die Einstiegshürden möglichst niedrig zu halten. Deshalb programmieren wir den Raspberry Pi Pico mit MicroPython. Das Programmieren mit MicroPython ist vergleichsweise leicht zu erlernen und für Einsteiger in die Hardware-nahe Programmierung optimal geeignet. Außerdem kannst Du Deine eigenen Ideen schnell umzusetzen.

Aus meiner Sicht sollte man immer auch sofort loslegen können. Nicht erst noch irgendwas zusammenbauen, was man aus der Ungeduld heraus nur notdürftig zusammenfrickelt. Und natürlich willst Du nicht auch noch irgendwelches Zubehör besorgen müssen. Deshalb hat im Elektronik-Set Pico Edition der Pico gelötete Stiftleisten.

Außerdem liegt dem Elektronik-Set ein speziell auf den Pico abgestimmtes Steckbrett bei, bei dem die Steckreihen nicht einfach nur durchnummeriert sind, sondern mit der GPIO-Nummer oder der Bezeichnung gekennzeichnet sind. Dadurch ist es viel leichter, den richtigen Pin zu finden.

Das Elektronik-Set Pico Edition ist also nicht nur ein einfaches Bauteile-Set, sondern alle Bauteile sind sinnvoll aufeinander abgestimmt. Dazu gehört der Elektronik-Guide Pico Edition mit vielen verschiedenen Programmcodes und Schaltungen zum Ausprobieren.

Vorwort (2)

Die meisten Aufbauten weichen nur geringfügig voneinander ab, damit Du mehrere Programmcodes nacheinander ausprobieren kann, ohne große Änderungen an den Schaltungen vornehmen zu müssen.

Herausgekommen ist das Elektronik-Set Pico Edition mit Anleitung und Bauteilen zum Experimentieren und Programmieren mit MicroPython. Dieser Elektronik-Guide und das dazugehörige Elektronik-Set soll die Schwelle zum Einstieg noch ein wenig tiefer legen.

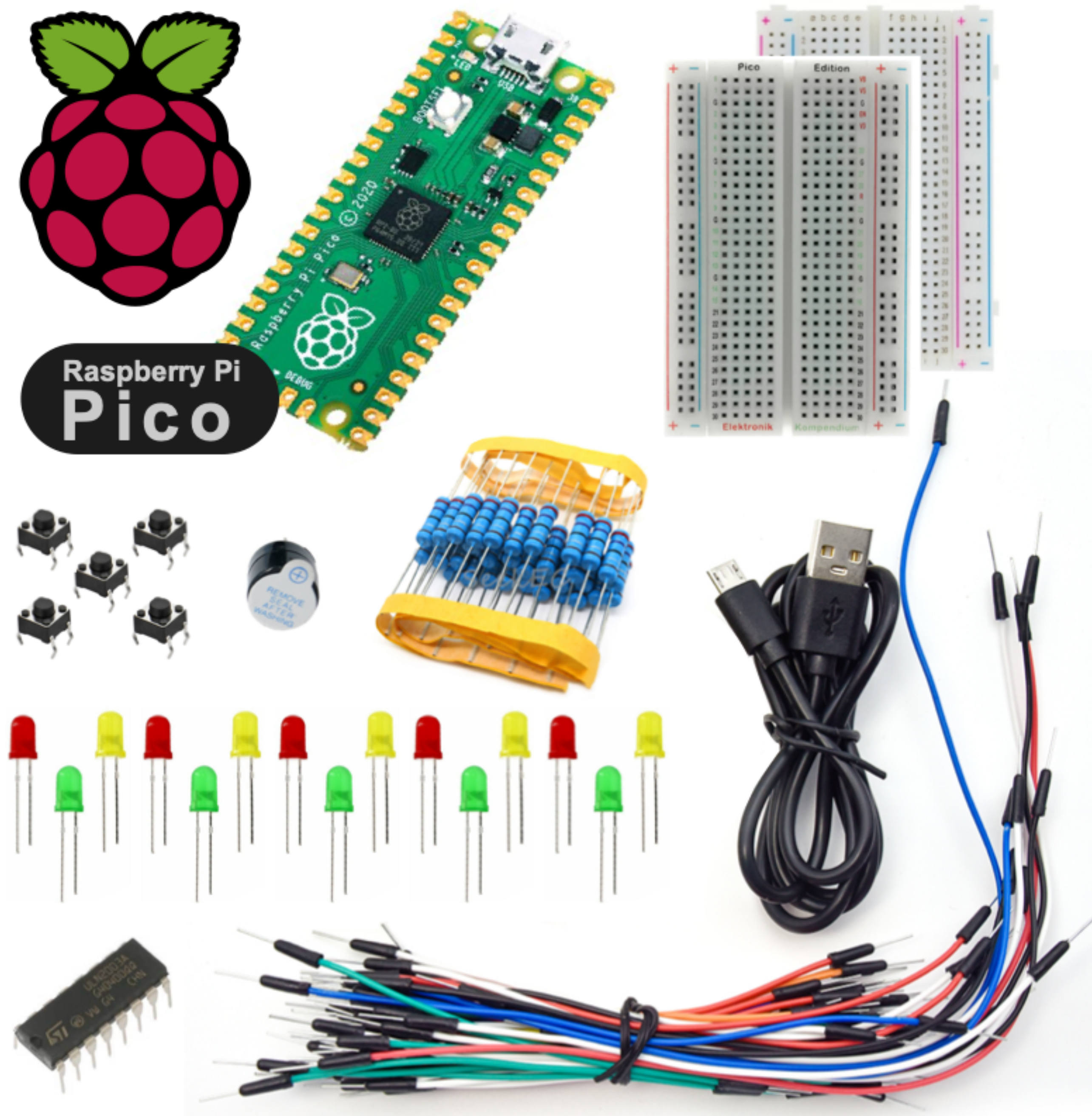
Ich finde, so macht Experimentieren mit dem Raspberry Pi Pico und Elektronik viel mehr Spaß.

Patrick Schnabel

PS: So ganz alleine, ohne jede Unterstützung? Das muss nicht sein. Wenn Du Lust hast, in entspannter Atmosphäre mit anderen mit dem Raspberry Pi Pico zu experimentieren, dann sind vielleicht unsere Online-Workshops etwas für Dich. Du bekommst während dem Workshop persönliche Unterstützung bei Problemen oder Fragen.

<https://www.elektronik-kompendium.de/service/events/>

Elektronik-Set Pico Edition



Das Elektronik-Set Pico Edition enthält viele elektronische Bauteile, um Hardware-nahes Programmieren zu lernen, Steuerungen selber zu programmieren und ohne LötKolben zu experimentieren.

Eine strukturierte Einführung berücksichtigt die Besonderheiten von Elektronik und Programmierung ohne Vorkenntnisse.

- Hardware-nahes Programmieren mit MicroPython ohne Vorkenntnisse.
- Grundlagen lernen, um Steuerungen selber zu programmieren.
- Ohne LötKolben experimentieren. Bauteile einfach stecken.

<https://www.elektronik-kompendium.de/shop/elektronik-set/pico-edition>

Inhaltsverzeichnis



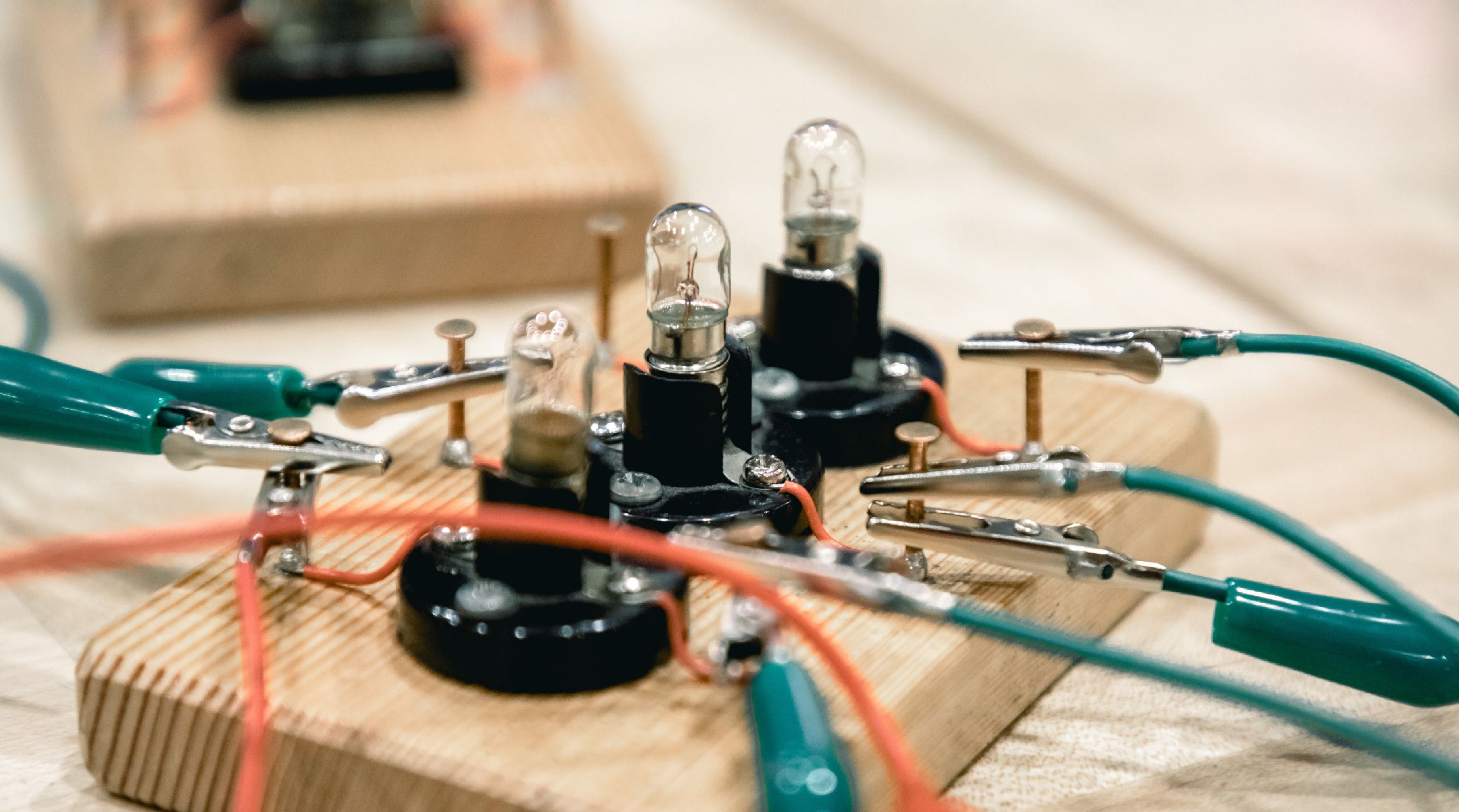
Einführung und Grundlagen



Bauteile

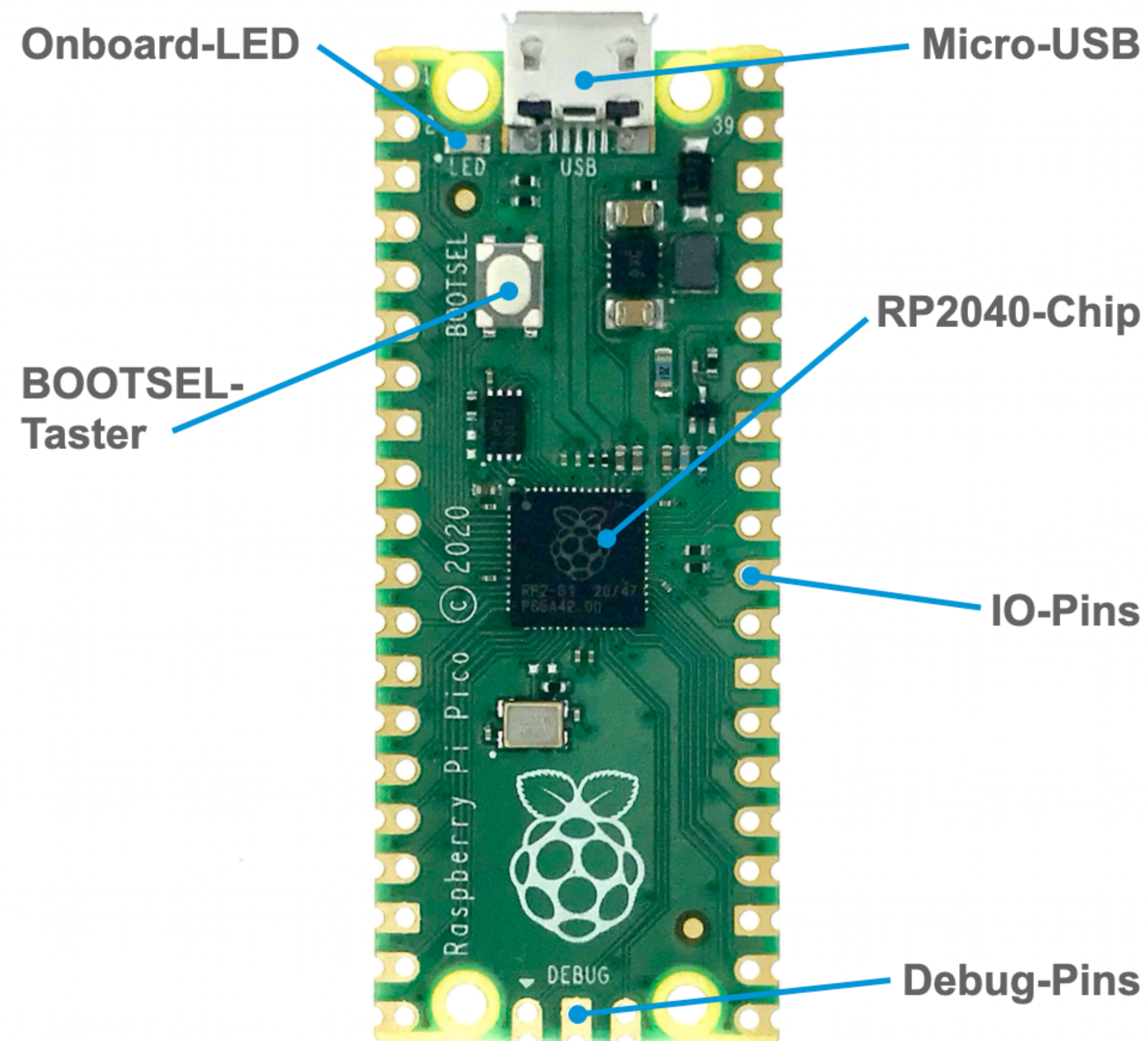


Experimente und Anwendungen



Einführung und Grundlagen

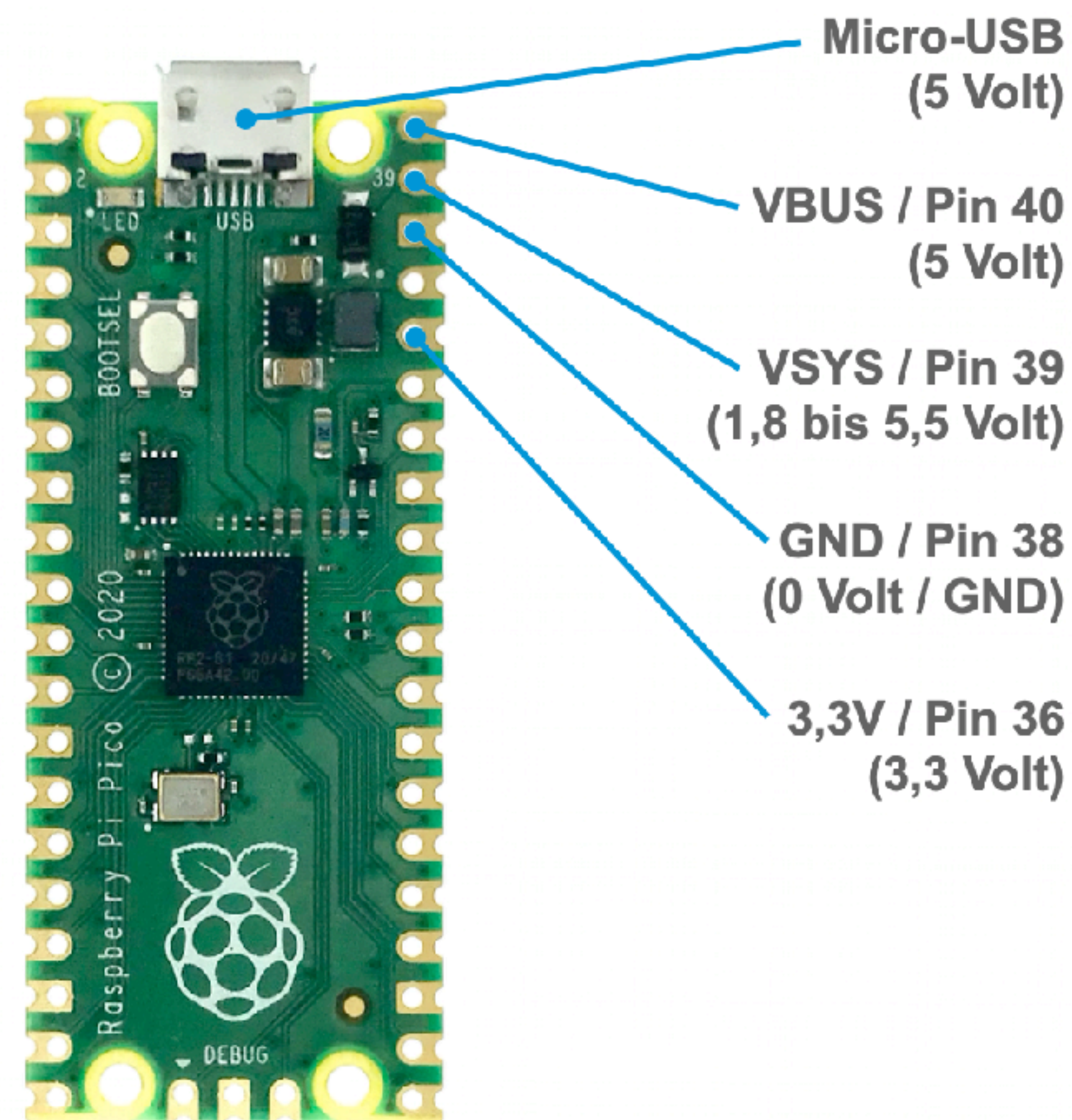
Raspberry Pi Pico: Was ist das?



Ein Raspberry Pi Pico ist ein Mikrocontroller-Board von der Raspberry Pi Foundation. Oft wird er liebevoll einfach nur „Pico“ genannt. Er hat einige nützliche Funktionen für Steuerungen, wie man es von anderen Mikrocontrollern kennt. Er lässt sich mit MicroPython, C/C++ und Visual Studio Code programmieren. Die Platine hat das Arduino-Nano-Format und lässt sich über eine Micro-USB-Buchse mit Strom versorgen und programmieren. Der Mikrocontroller ist ein RP2040. Zum Speichern des Programmcodes ist ein 2 MByte großer Flash-Speicher vorhanden. Eine Besonderheit, im Vergleich zu anderen Mikrocontrollern, sind die Programmable IOs (PIO), die unabhängig programmierbar sind.

Ein Mikrocontroller, wie der Pico, ist dafür gedacht, einzelne Aufgaben zu übernehmen. Typischerweise aus dem Bereich der Steuerung, Regelung und Automation. Dafür verfügt er über zahlreiche analoge und digitale Eingänge und Ausgänge mit unterschiedlichen Funktionen. Wir bewegen uns also im Bereich der hardwarenahen Programmierung und Physical Computing.

Energieversorgung



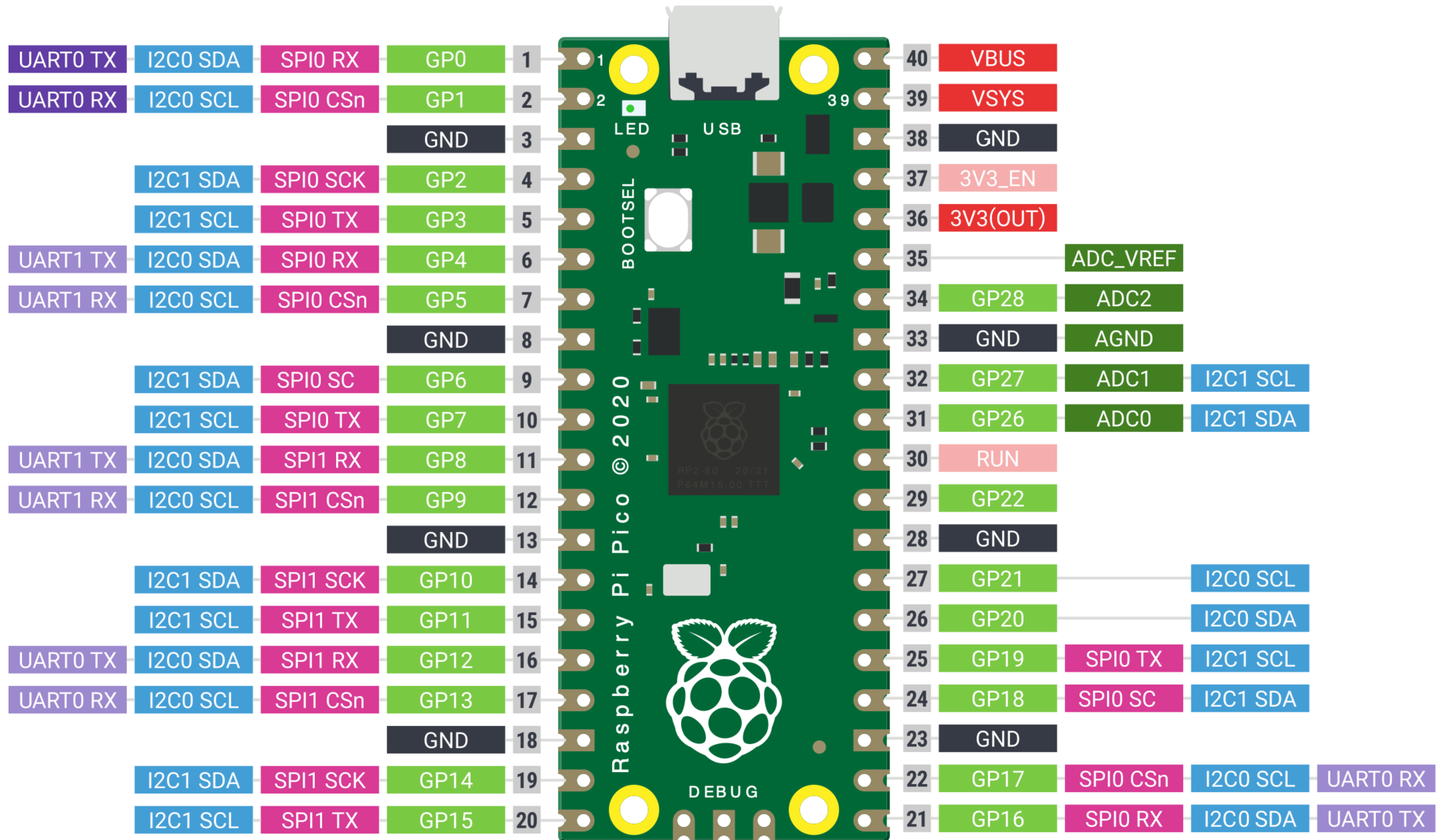
- 5 Volt über den Micro-USB-Anschluss
- 5 Volt über den VBUS-Pin (Pin 40), ohne USB-Speisung oder wenn Pico im USB-Host-Betrieb
- 3,3 Volt über den 3V3-Pin (Pin 36)
- 1,8 bis 5 Volt über den VSYS-Pin (Pin 39)
- Der Anschluss von 0 Volt bzw. GND kann über einen freien GND-Pin erfolgen. Zum Beispiel Pin 38.

Der Raspberry Pi Pico kann auf mehrere Arten mit Strom versorgt werden. Die einfachste Methode ist, den Micro-USB-Anschluss über ein USB-Kabel mit einem Computer zu verbinden. Alternativ besteht die Möglichkeit im Stand-alone-Betrieb, also ohne Computer, ein normales USB-Netzteil zu verwenden.

Weitere Methoden bestehen im direkten Anschluss einer externen, festen Spannung über den 3,3V-Pin der IO-Pins. Oder, wenn eine andere Spannung verwendet werden muss, dann kann man diese Spannungsquelle mit dem VSYS-Pin der IO-Pins verbinden. Hier darf die Spannung zwischen 1,8 und 5,5 Volt liegen. Auf der Platine ist ein Spannungswandler, der aus einer Eingangsspannung einer Versorgung mit +3,3 Volt macht. Das ist praktisch, wenn man mit 2 oder 3 AA-Batterien eine mobile Energieversorgung realisieren will.

Der Stromverbrauch ist abhängig von den angeschlossenen Bauteilen und liegt bei einfachen Experimenten mit ein paar Leuchtdioden bei maximal 20 mA.

GPIO-Belegung (Pinout)



Entwicklungsumgebung zum Programmieren

Was ist eine Entwicklungsumgebung?

Mit Entwicklungsumgebung ist meist ein Programm gemeint, in dem der Programmcode oder Quelltext für ein anderes Programm geschrieben wird. Also das Programm, mit dem programmiert wird. Im einfachsten Fall ist das ein einfacher Text-Editor. Programmierer verwenden gerne spezielle Text-Editoren die über Syntax-Highlighting verfügen. Das heißt, die Grammatik der Programmiersprache erkennen und zur besseren Lesbarkeit bestimmte Befehle, Kommandos, Optionen, Parameter usw. farblich unterschiedlich darstellen. Hilfreich sind Editoren, die auch noch Eingabefehler erkennen und kennzeichnen können. Desweiteren vereinfacht es die Software-Entwicklung, wenn man direkt im Editor das geschriebene Programm starten und stoppen kann.

Zu einer vollständigen Entwicklungsumgebung gehört meist eine bestimmte Hardware und zusätzliche Software, was aber davon abhängig ist, in welcher Programmiersprache geschrieben werden soll und was genau programmiert wird. Beispielsweise, ob eine bestimmte Hardware berücksichtigt werden muss.

- Hardware: Beliebiger Computer mit Zubehör (Maus, Tastatur, Bildschirm)
- Elektronik: Raspberry Pi Pico mit verschiedenen Bauteilen, Steckbrett, Verbindungskabel und Micro-USB-Kabel
- Programmiersprache: MicroPython (muss zuerst auf dem Pico installiert werden)
- Editor: Thonny Python IDE (muss auf dem Computer installiert sein)
- Internet: von Vorteil, aber nicht zwingend erforderlich

Raspberry Pi Pico programmieren

Editor: Thonny Python IDE

Das Schreiben und Programmieren eines Programms erfolgt in einem Editor. Empfohlen wird die Thonny Python IDE.

Diesen Editor gibt es für verschiedene Betriebssysteme. Wenn Du mit der Desktop-Version von Raspberry Pi OS arbeitest, dann erreichst Du das Programm im Menü über das Untermenü „Entwicklung“. In anderen Betriebssystemen muss Thonny erst noch installiert werden.

Stelle zuerst sicher, dass die Thonny Python IDE installiert ist. Start das Programm testweise und beende es anschließend wieder.

<https://thonny.org/>

Programmiersprache: MicroPython

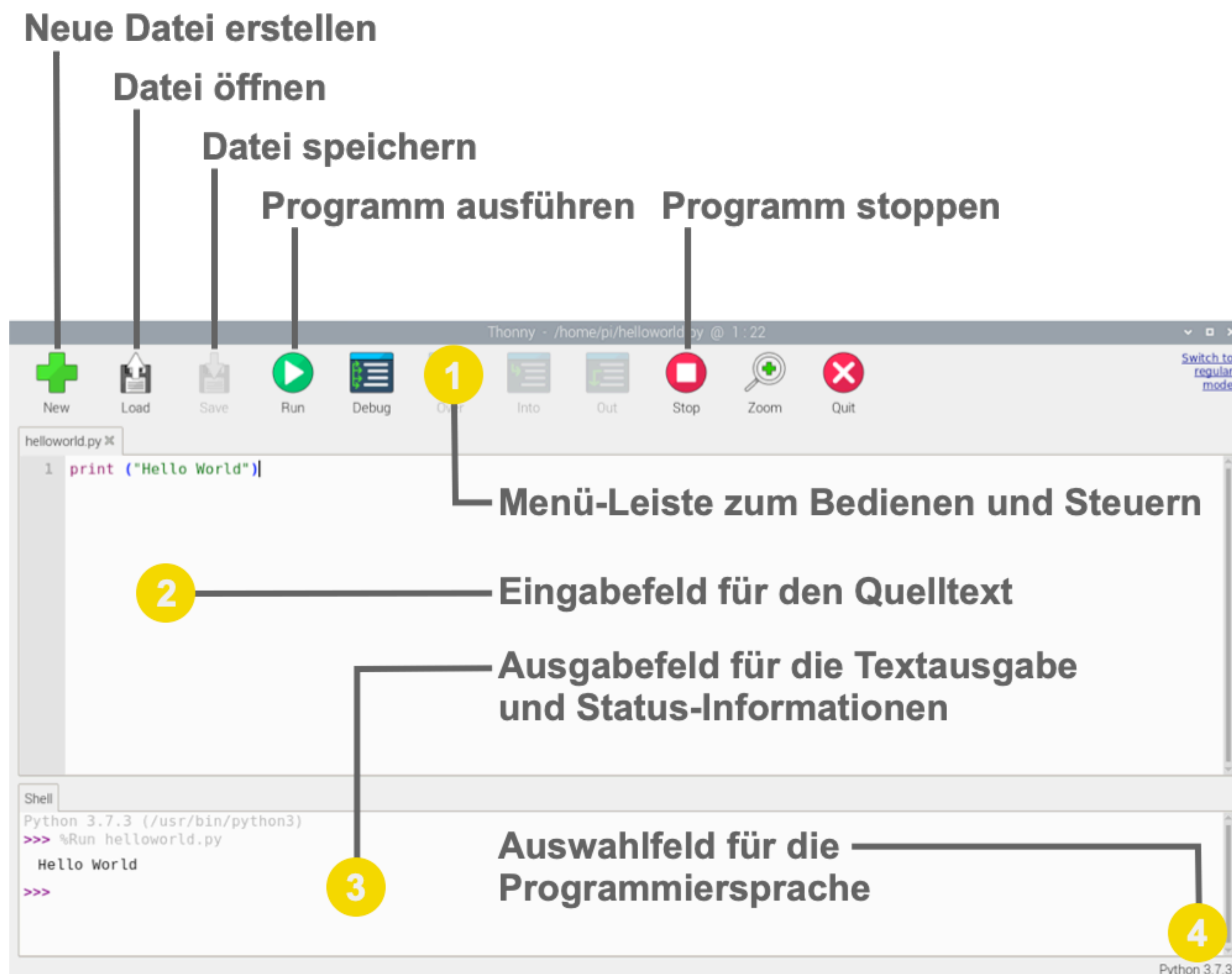
Standardmäßig ist der Raspberry Pi Pico für die Programmierung mit C und C++ eingerichtet.

Für Anfänger ist MicroPython zu empfehlen. Das ist eine einfachere Version der Programmiersprache Python, die speziell für Mikrocontroller entwickelt wurde.

Wenn der Pico mit MicroPython programmiert werden soll, dann muss MicroPython zuerst auf dem Board installiert werden. Der Vorgang ist einmalig und besteht nur darin, eine Datei aus dem Internet herunterzuladen und auf den Pico zu kopieren.

<https://docs.micropython.org/en/latest/rp2/quickref.html>

Bedienen der Thonny Python IDE



Die Thonny Python IDE verfügt über alle erforderlichen Funktionen und Bedienelemente, um effektiv mit dem Pico und MicroPython programmieren zu können.

Aufteilung der Entwicklungsumgebung

1. Menü-Leiste zum Bedienen und Steuern
2. Eingabefeld für den Quelltext
3. Ausgabefeld für die Textausgabe und Status-Infos

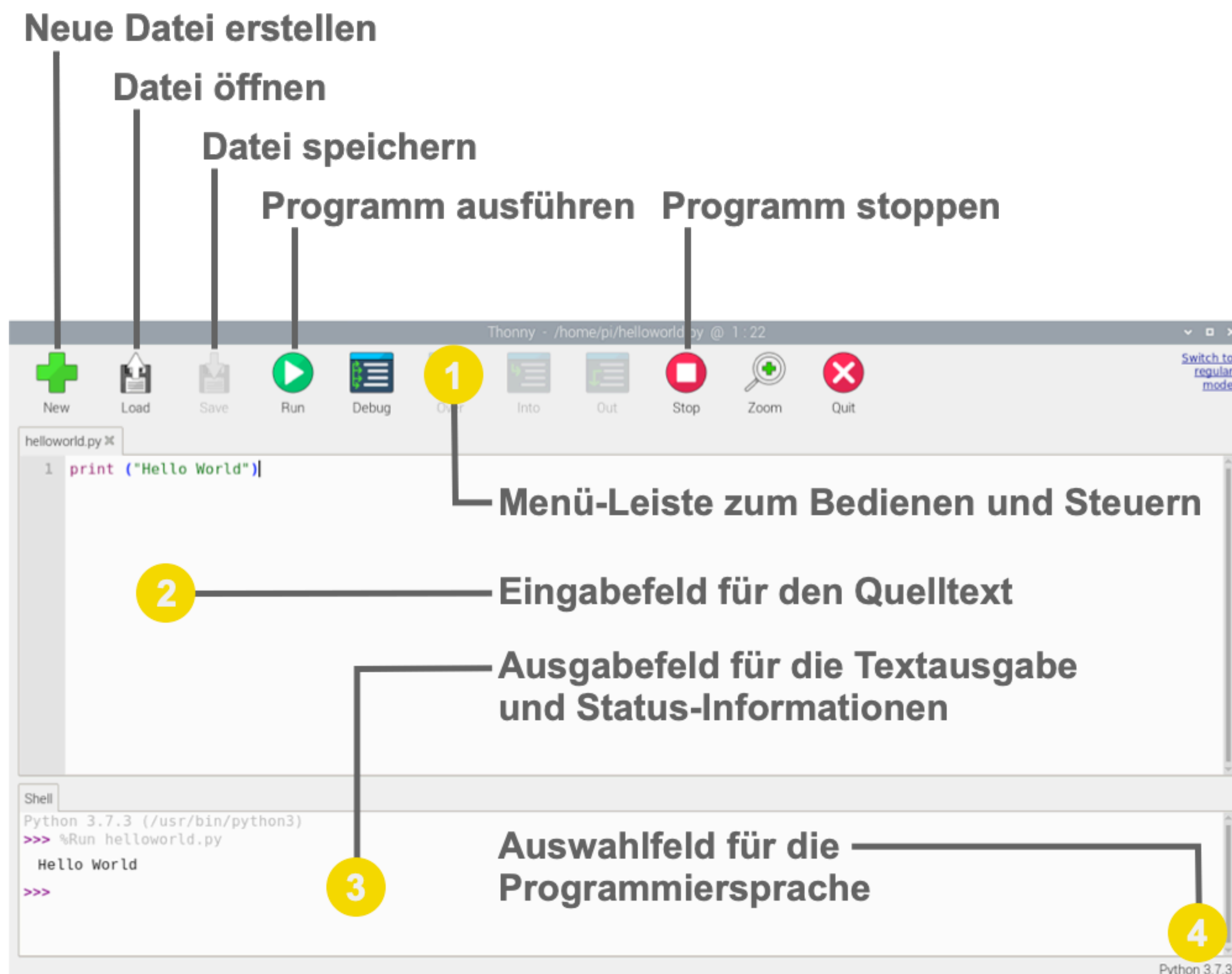
Wichtige Funktionen der Menü-Leiste

- Neue Datei erstellen (New)
- Datei öffnen (Load)
- Datei speichern (Save)
- Programm ausführen (Run)
- Programm stoppen (Stop)

Auswahl der Programmiersprache

Rechts unten befindet sich die Anzeige der aktuell ausgewählten Programmiersprache. Mit einem Klick auf diese Status-Anzeige öffnet sich ein Menü mit weiteren Programmiersprachen und Python-Versionen. Für die Programmierung des Raspberry Pi Pico ist hier „MicroPython“ auszuwählen.

MicroPython auf dem Pico installieren (1)



Verbinde Deinen Pico mit Deinem Computer

Hilfreich ist es, wenn Du Deinen Pico immer zuerst über das Micro-USB-Kabel mit Deinem Computer verbindest und erst danach die Thonny Python IDE startest.

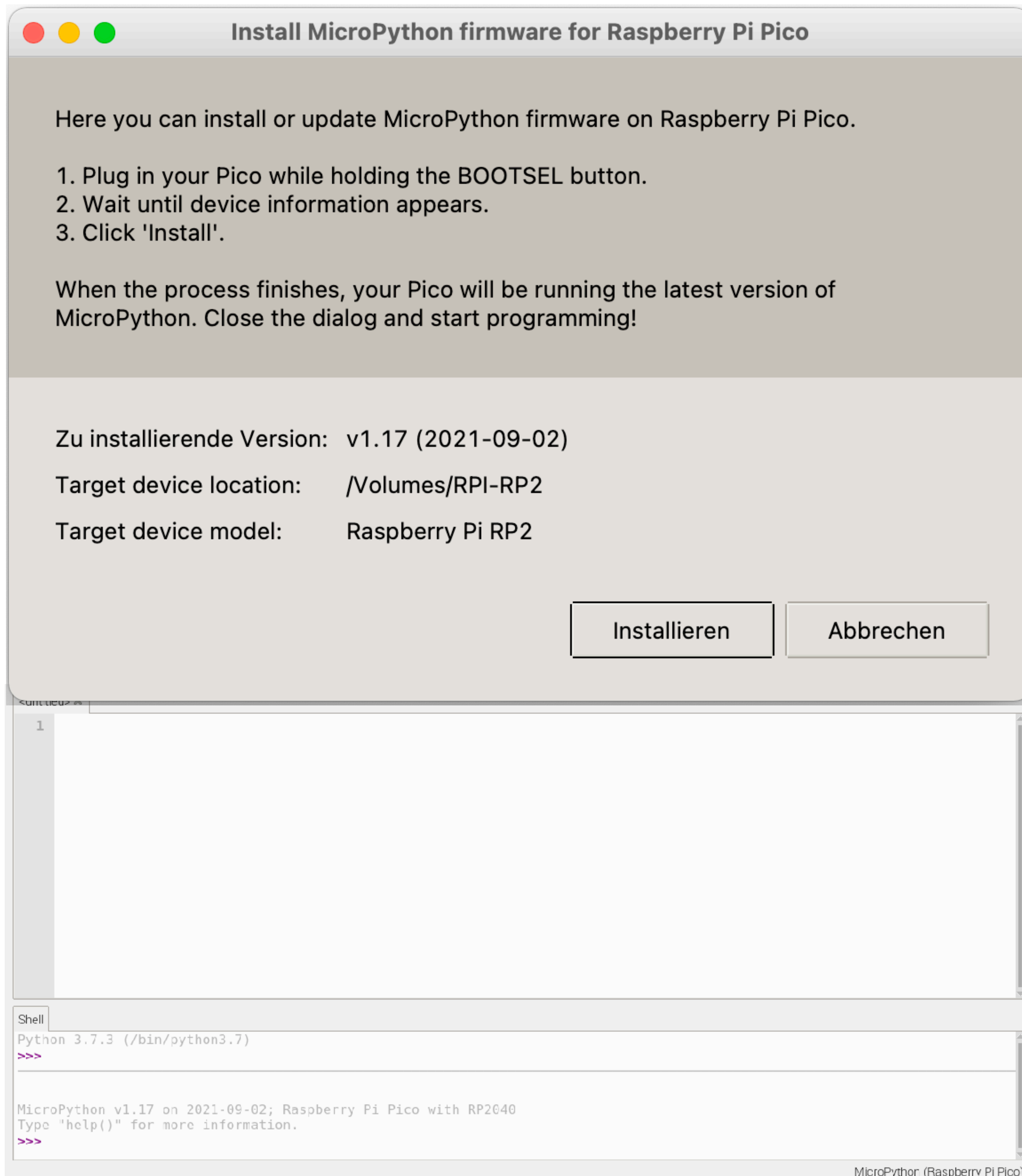
Es kann sein, dass Du keine Rückmeldung beim Einstecken des Picos von Deinem Betriebssystem bekommst. Das ist in Ordnung.

Thonny-Editor starten und MicroPython auswählen

Wenn der Pico mit Deinem Computer verbunden ist, kannst Du den Thonny-Editor starten. Diese Reihenfolge hat den Vorteil, dass der Thonny-Editor den Pico automatisch erkennt. Hier ist zu beachten, dass die Thonny Python IDE normalerweise auf Python eingestellt ist. Um den Pico mit MicroPython programmieren zu können, muss in der Thonny Python IDE noch die richtige Programmiersprache und Version ausgewählt werden.

Klicke dazu rechts unten auf die Bezeichnung „Python“ oder ähnlich und wähle dort „MicroPython“ für Raspberry Pi Pico aus. Danach ist das Programm umgestellt und der Pico wird dabei automatisch erkannt, wenn er per USB mit Deinem Computer verbunden ist.

MicroPython auf dem Pico installieren (2)



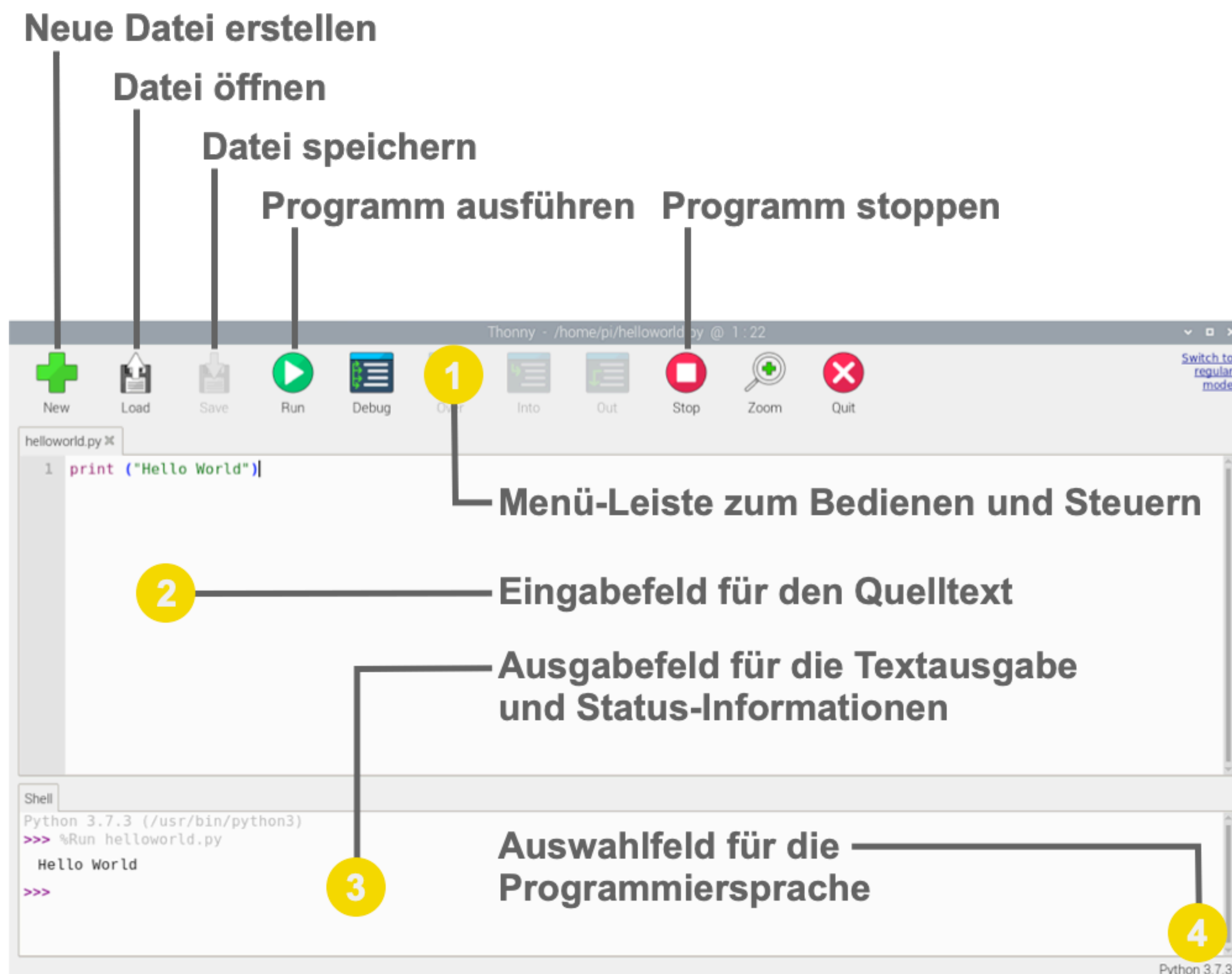
MicroPython auf dem Raspberry Pi Pico installieren

Beim erstmaligen Erkennen eines Raspberry Pi Picos wird die Thonny Python IDE eine Firmware aus dem Internet nachladen wollen. Das kannst Du durch Klicken auf „Install“ bestätigen. Der Vorgang ist in der Regel innerhalb weniger Sekunden erledigt. Danach ist Dein Raspberry Pi Pico zum Programmieren mit MicroPython und dem Thonny-Editor bereit.

Wichtig

Der Thonny-Editor ist mit dem Pico dann zum Programmieren bereit, wenn im Feld „Shell“ bzw. „Kommandozeile“ eine Meldung mit „MicroPython“ und „Raspberry Pi Pico with RP2040“ erscheint. Wenn nicht, dann gibt es dafür eine einfache Lösung: Wenn offensichtlich keine Verbindung besteht, dann kannst Du durch Klicken auf „STOP“ in der Menü-Leiste die Verbindung manuell herstellen.

Programmieren mit der Thonny Python IDE



Neue Datei erstellen

Am Anfang wirst Du ein neues Programm immer mit einer neuen Datei beginnen indem Du auf „New“ klickst (grünes Plus). Anschließend kannst Du einen fertigen Quelltext in das Textfeld kopieren oder Dein Programm von Grund auf neu schreiben.

Programm speichern

Zum Speichern Deines Programms dient im Thonny-Editor die „Save“-Funktion (Diskette mit Pfeil nach unten). Nachdem Du darauf geklickt hast, hast Du die Möglichkeit das Programm entweder auf Deinem Computer oder auf dem Pico zu speichern. Zum Ausführen auf dem Pico musst Du Dein Programm auf dem Pico speichern. Wichtig ist, dass Du die Datei mit der Dateiendung „.py“ speicherst.

Wichtig: Der Thonny-Editor merkt sich beim Öffnen und Speichern von Dateien den Speicherort.

Programm ausführen und stoppen

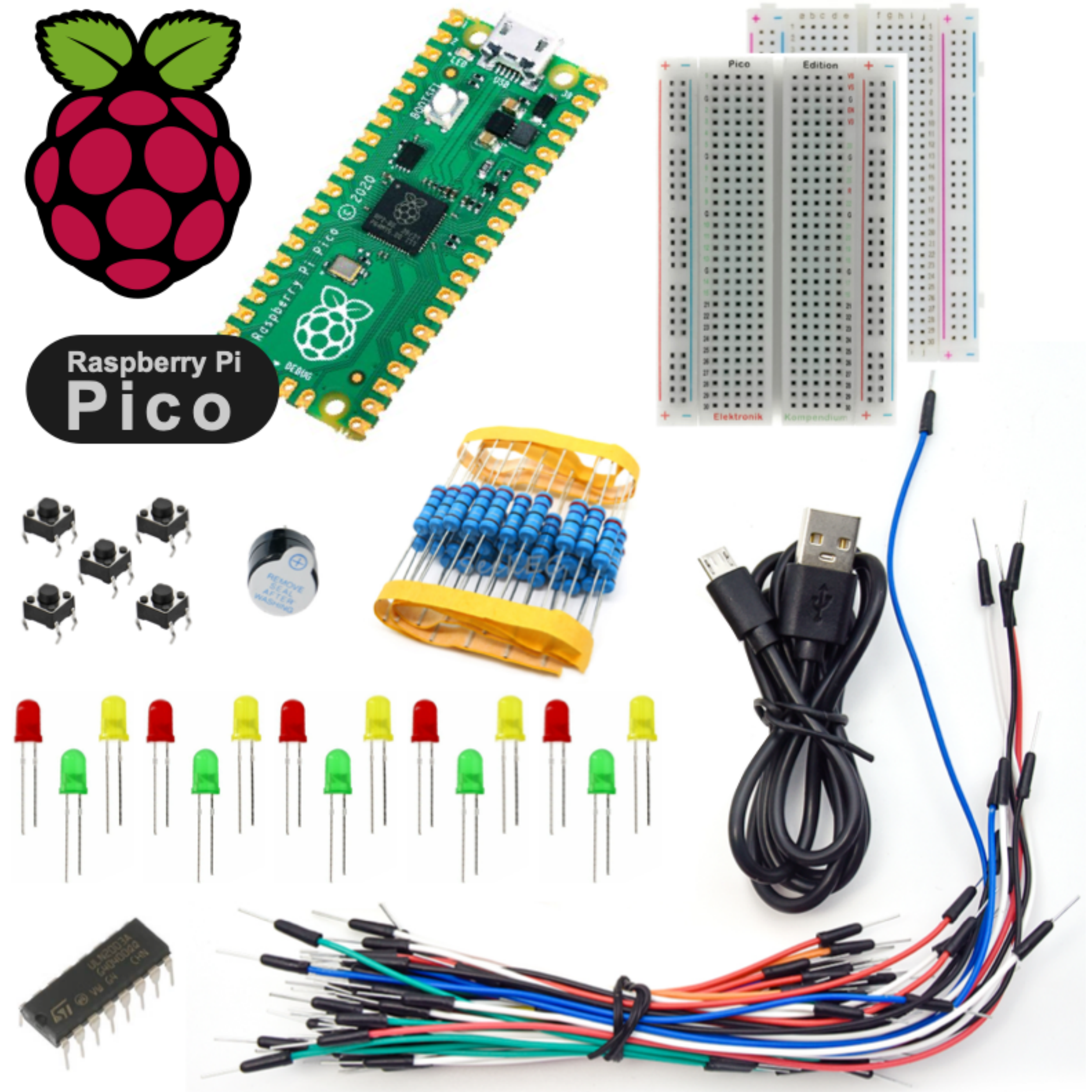
Das Programm startet dann, wenn Du auf „Run“ klickst (grüner Kreis mit weißem Dreieck). Wenn Du das Programm beenden willst, klickst Du auf „Stop“ (roter Kreis mit weißem Rechteck).



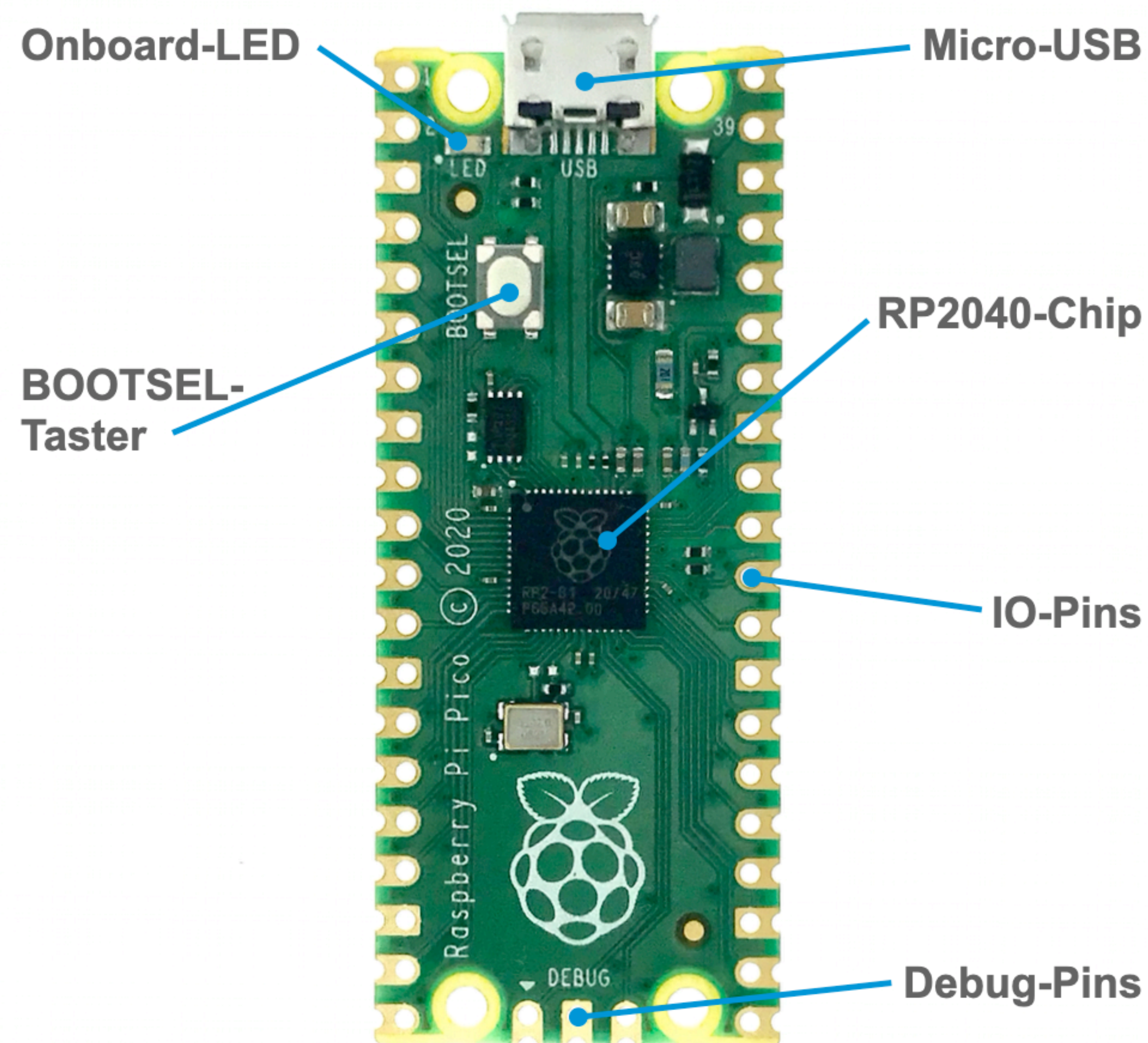
Bauteile

Bauteile im Elektronik-Set Pico Edition

- Raspberry Pi Pico (Mikrocontroller-Modul)
- USB-Kabel
- Steckbrett „Pico Edition“
- Verbindungskabel
- Taster
- Widerstände
- Leuchtdioden
- Potentiometer
- (Summer)
- ULN2003A (IC) für zukünftige Anwendungen



Raspberry Pi Pico (Modul)



Der Pico wird über ein USB-Kabel mit einem Computer verbunden. Der Pico wird dabei mit Strom versorgt und meldet sich dann am Computer an. Die Programmierung erfolgt über eine virtuelle serielle Schnittstelle.

- programmierbarer Mikrocontroller
- Modul mit Chip, Speicher und Anschlüssen

Der Raspberry Pi Pico ist eine Platine mit einem Mikrocontroller drauf, zusätzlichen Bausteinen für die externe Beschaltung und Programmierung. Mit zwei 20-poligen Stiftleisten ist die Platine Steckbrett-kompatibel, wodurch die GPIO-Pins für die externe Beschaltung leichter zugänglich sind. Auf einem Steckbrett lassen sich Kabel und andere Bauteile durch Stecken leichter hinzufügen oder entfernen.

An einem Ende der Platine befindet sich ein Micro-USB-Anschluss (Typ B) zur Stromversorgung und zur Programmierung über einen Computer.

Neben dem USB-Anschluss befindet sich eine Onboard-LED, die für die eigene Programmentwicklung genutzt werden kann. Daneben wiederum befindet sich ein BOOTSEL-Taster, der während der Programmierung des Mikrocontrollers benötigt wird.

USB-Kabel



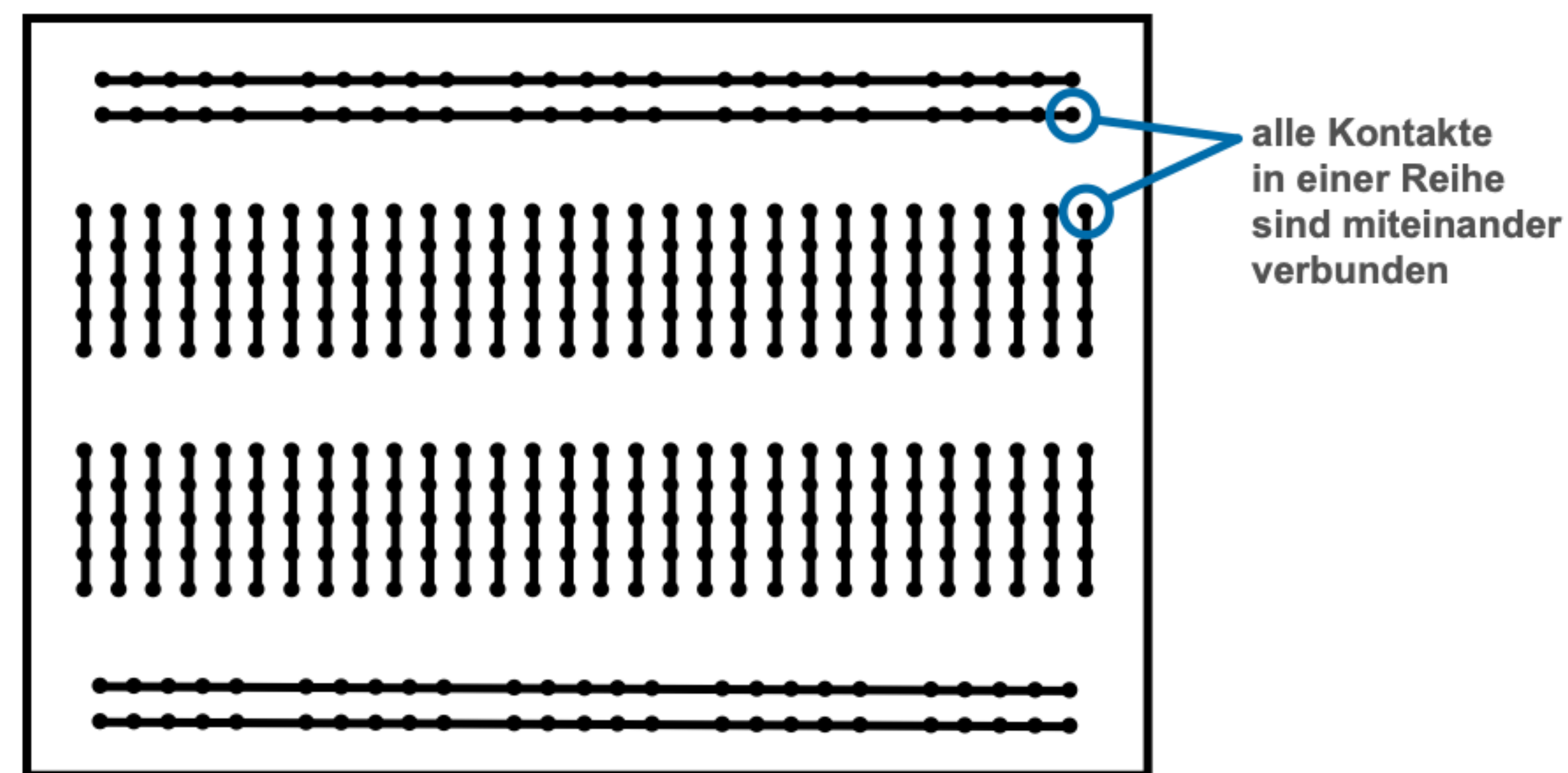
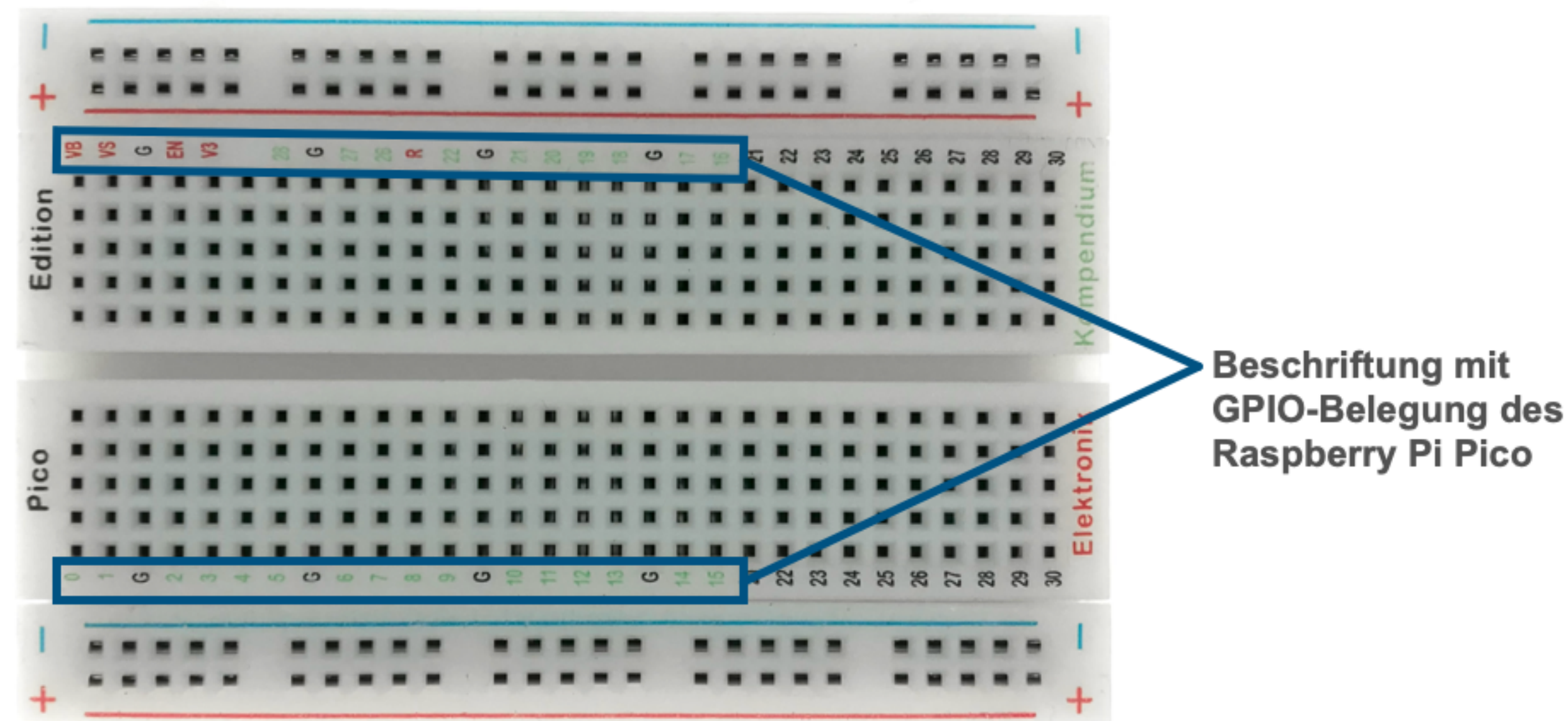
- Verbindungskabel zwischen Computer (Host) und Raspberry Pi Pico (Endgerät)
- Verbindungskabel mit Datenleitungen
- Stromversorgung für den Pico oder ein anderes Endgerät mit 5 Volt

USB steht für Universal Serial Bus. Es ist eine universelle Schnittstelle, um Endgeräte und Peripherie mit einem Computer zu verbinden. Typischerweise Tastatur, Maus, USB-Sticks, Drucker und ähnliches. Eine Besonderheit dieser Schnittstelle ist, dass sie nicht nur Datenleitungen hat, sondern auch die Energieversorgung für das Endgerät übernehmen kann.

An einem Ende der Platine des Pico befindet sich ein Micro-USB-Anschluss (Typ B) zur Stromversorgung und zur Programmierung über einen Computer.

Der Pico wird über den USB-Kabel mit einem Computer verbunden. Der Pico meldet sich dabei als Laufwerk am Computer an und kann durch Speichern einer Datei mit dem Programmcode auf diesem Laufwerk programmiert werden.

Steckbrett für den Raspberry Pi Pico



Die durchgezogenen Linien kennzeichnen die Steckkontakte, die eine elektrische und damit leitende Verbindung aufweisen.

- lötfreies Experimentieren durch Steckverbindungen
- aufgedruckte GPIO-Nummern des Pico

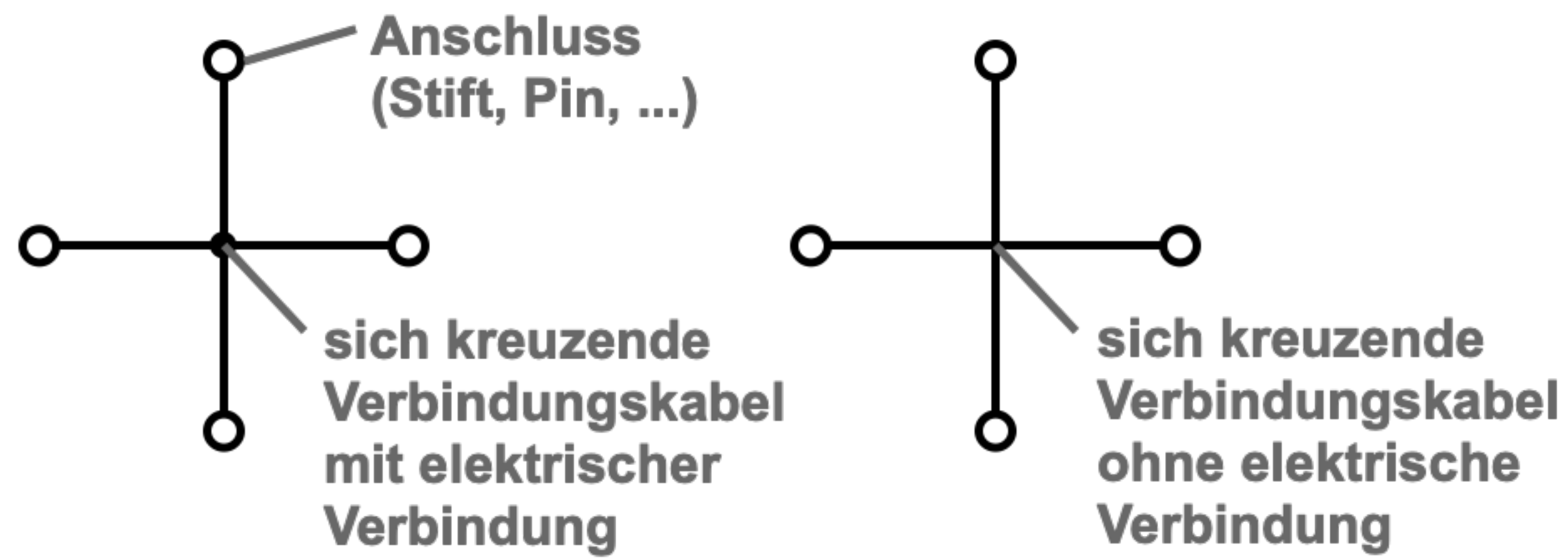
Ein Steckbrett oder Steckboard ist eine Experimentier-Platine, auf der sich elektronische Bauteile durch Stecken mechanisch befestigen und zu Schaltungen elektrisch verbinden lassen.

Das Steckbrett wird im Englischen oft „Solderless Breadboard“ genannt, womit angedeutet wird, dass das Verbinden mit dem Steckbrett lötfrei erfolgt. Das Steckbrett hat 400 Kontakte im 2,54-mm- bzw. 1/10-Zoll-Raster, die reihenweise miteinander verbunden sind. Über diese Kontaktreihen und zusätzlich steckbare Drahtbrücken werden die einzelnen Bauteile miteinander verbunden.

In der Mitte des Steckbretts befindet sich ein breiter Steg, damit die Anschlüsse von integrierten Schaltungen (ICs) in DIL-Bauform voneinander getrennt sind.

Eine spezielle Beschriftung an den Steckreihen kennzeichnet die GPIO-Nummern des Raspberry Pi Pico.

Steckverbindungskabel

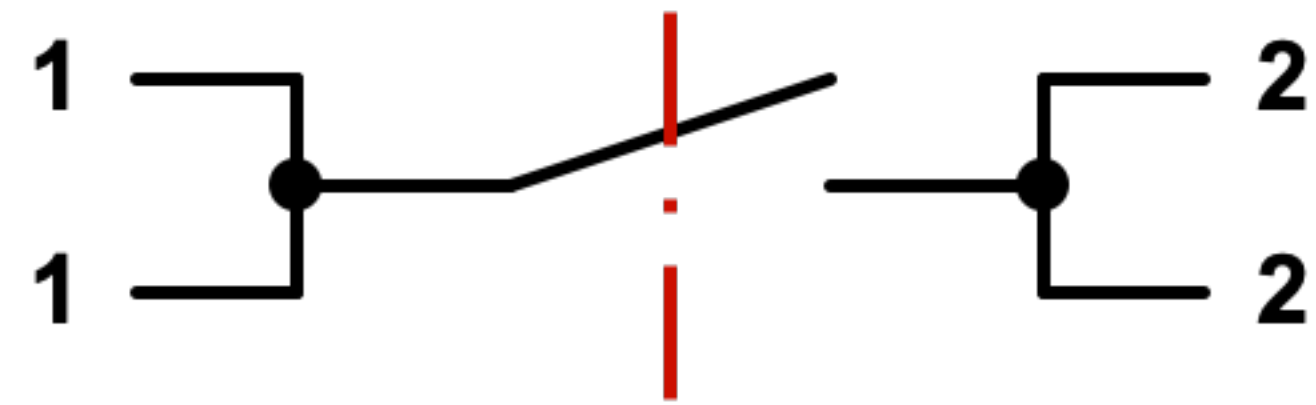


- elektrische Verbindungen zwischen Bauelementen, Schaltungen und Geräten

Steckverbindungskabel oder auch nur Verbindungskabel dienen zum Trennen und Verbinden von Bauelementen, Schaltungen und Geräten.

Bei elektrischen Steckverbindungen unterscheidet man den männlichen Teil einer Steckverbindung (mit nach außen weisenden Kontaktstiften) und den weiblichen Teil (mit nach innen weisenden Kontaktöffnungen). Den männlichen Teil bezeichnet man als Stecker, wenn er sich an einem Kabelende befindet. Den weiblichen Teil bezeichnet man richtigerweise als Kupplung, wenn er sich an einem Kabelende befindet. Manchmal sagt man auch Buchse dazu.

Taster



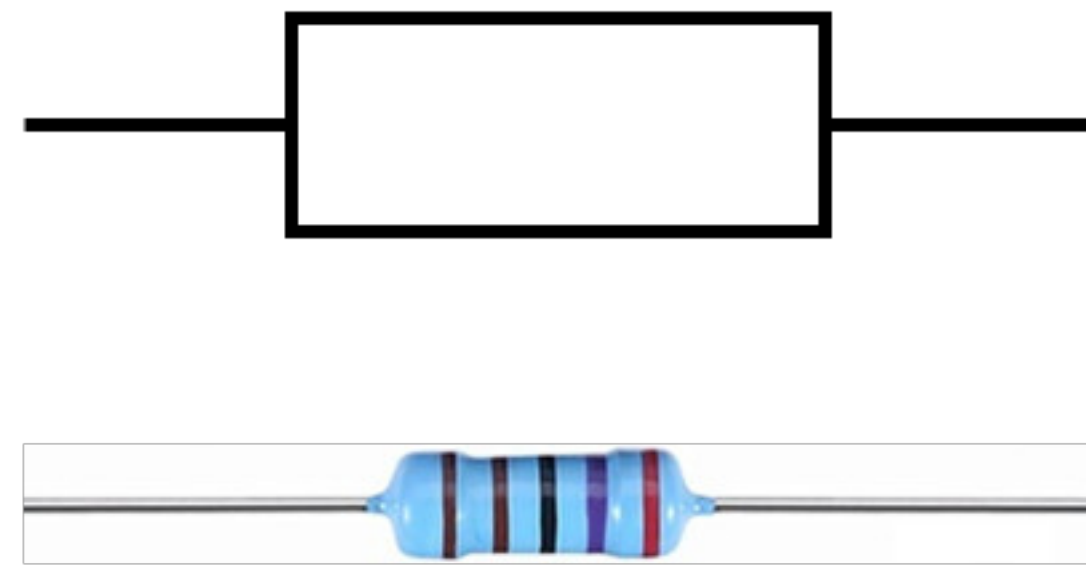
Die im Bild mit 1 und 2 gekennzeichneten Anschlüsse können je nach Bauform an unterschiedlichen Positionen liegen. Welche Anschlüsse zusammengehören, muss man mit einem Durchgangsprüfer oder Widerstandsmessgerät herausfinden.

- Stromkreis schließen
- Stromkreis unterbrechen
- manuelles Ein- und Ausschalten

Durch einen Taster kann ein Stromkreis geschlossen und unterbrochen werden. Damit steuert man, ob im Stromkreis ein Strom fließt oder nicht. Bei dem dargestellten Taster handelt es sich um einen 2-poligen Taster mit 4 Anschlüssen. Je zwei Anschlüsse stellen einen Kontakt des Tasters dar. Bei einem Taster ist der geöffnete Zustand der Normalzustand. Der Kontakt wird nur durch Drücken des Tasters geschlossen. Und der Kontakt bleibt nur solange geschlossen, wie der Taster gedrückt bleibt. Lässt man den Taster los, dann öffnet sich der Kontakt wieder.

Wenn man einen Stromkreis richtig ein- und ausschalten will, dann sollte man dafür einen Schalter verwenden, der nach der Betätigung in seiner Position bleibt. Ein Taster schaltet sich automatisch in seinen Normalzustand zurück.

Widerstand



Ein Widerstand ist zwar ungepolt, doch auch er kann kaputt gehen, wenn er zu viel Spannung oder zu viel Strom ausgesetzt wird. Die relevante Größe ist die elektrische Leistung in Watt (W) oder Milliwatt (mW). Das ist mathematisch ein Produkt aus Spannung und Strom. In der Regel verwendet man Widerstände mit maximal 250 mW bzw. 0,25 W Verlustleistung. In der Regel ist das bei kleinen Spannungen bis 9 V und kleine Ströme bis 25 mA unproblematisch.

Hinweis: Widerstände, die mit einer zu großen Leistung betrieben werden, werden heiß und brennen durch. Dann qualmt es etwas und der Widerstand färbt sich braun bis schwarz.

- Begrenzen des elektrischen Stroms
- Begrenzen der elektrischen Spannung
- Einstellen von Strom und Spannung

Mit einem Widerstand kann man Strom und Spannung in einer Schaltung begrenzen und bestimmte Spannungs- und Stromwerte an bestimmten Punkten in einer Schaltung einstellen.


Um einen bestimmten Widerstandswert zu errechnen, verwendet man das Ohmsche Gesetz.

In der praktischen Elektronik unterscheidet man zwischen Kohleschichtwiderständen und Metallfilmwiderständen.

Kohleschichtwiderstände haben meist einen hell gefärbten Widerstandskörper und weisen typischerweise 4 Ringe auf. Metallfilmwiderstände haben meist einen Hellblau gefärbten Widerstandskörper und weisen typischerweise 5 Ringe auf.

Kennzeichnung von Widerständen (1)


4 Ringe



**1.000 Ω
± 5 %**

Farbe	1. Ring	2. Ring	3. Ring	Multiplikator	Toleranz
Schwarz	0	0	0	× 1 Ω	
Braun	1	1	1	× 10 Ω	± 1 %
Rot	2	2	2	× 100 Ω	± 2 %
Orange	3	3	3	× 1.000 Ω (1 kΩ)	
Gelb	4	4	4	× 10.000 Ω (10 kΩ)	
Grün	5	5	5	× 100.000 Ω (100 kΩ)	± 0,5 %
Blau	6	6	6	× 1.000.000 Ω (1 MΩ)	± 0,25 %
Lila	7	7	7	× 10.000.000 Ω (10 MΩ)	± 0,1 %
Grau	8	8	8		± 0,05 %
Weiß	9	9	9		
Gold				× 0,1 Ω	± 5 %
Silber				× 0,01 Ω	± 10 %

5 Ringe



**2.700 Ω
± 1 %**

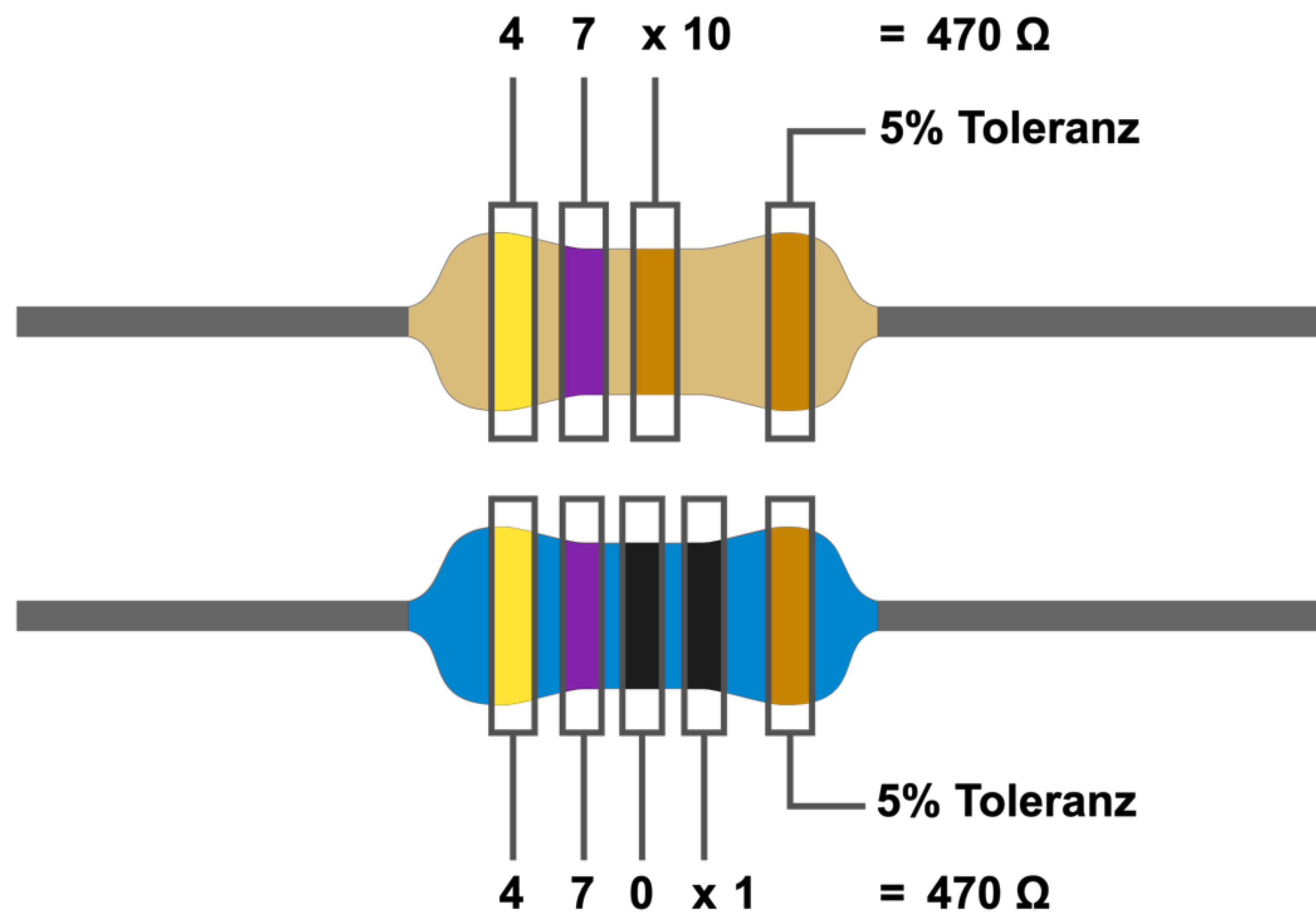
Widerstände werden in der Regel durch Farbringe gekennzeichnet. Je nach Art des Widerstandsmaterials, werden 4 oder 5 Ringe verwendet.

Kohleschichtwiderstände haben meist einen Beige (Mischung aus Weiß und Braun) gefärbten Widerstandskörper und weisen typischerweise 4 Ringe auf. Metallfilmwiderstände haben meist einen Hellblau gefärbten Widerstandskörper und weisen typischerweise 5 Ringe auf.

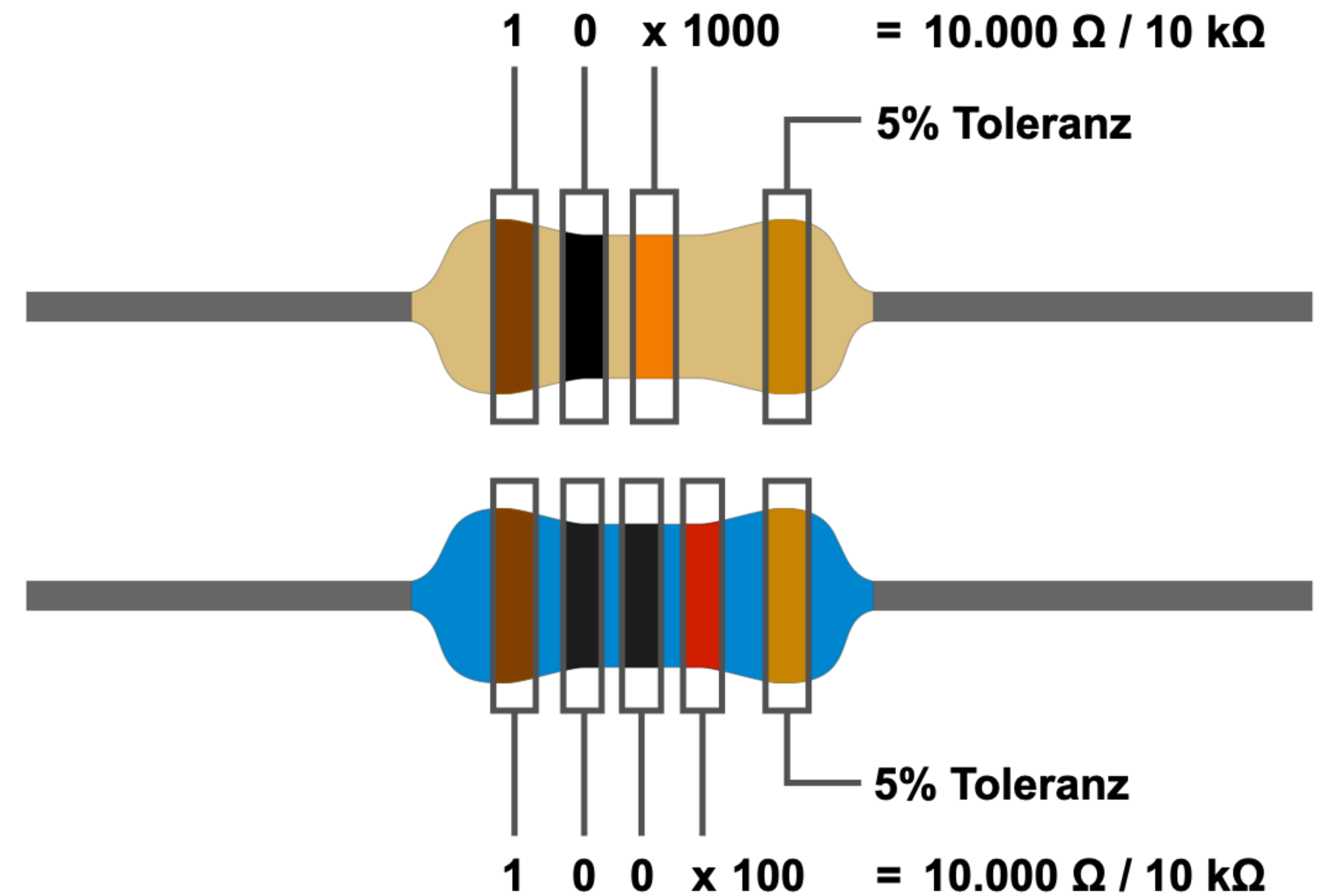
Die Ringe entsprechen einem internationalen Farbcode zur Kennzeichnung und Bestimmung des Widerstandswertes in Ω, sowie der Toleranz dieses Wertes. Um die Werte abzulesen, behilft man sich mit einer Tabelle.

Kennzeichnung von Widerständen (2)

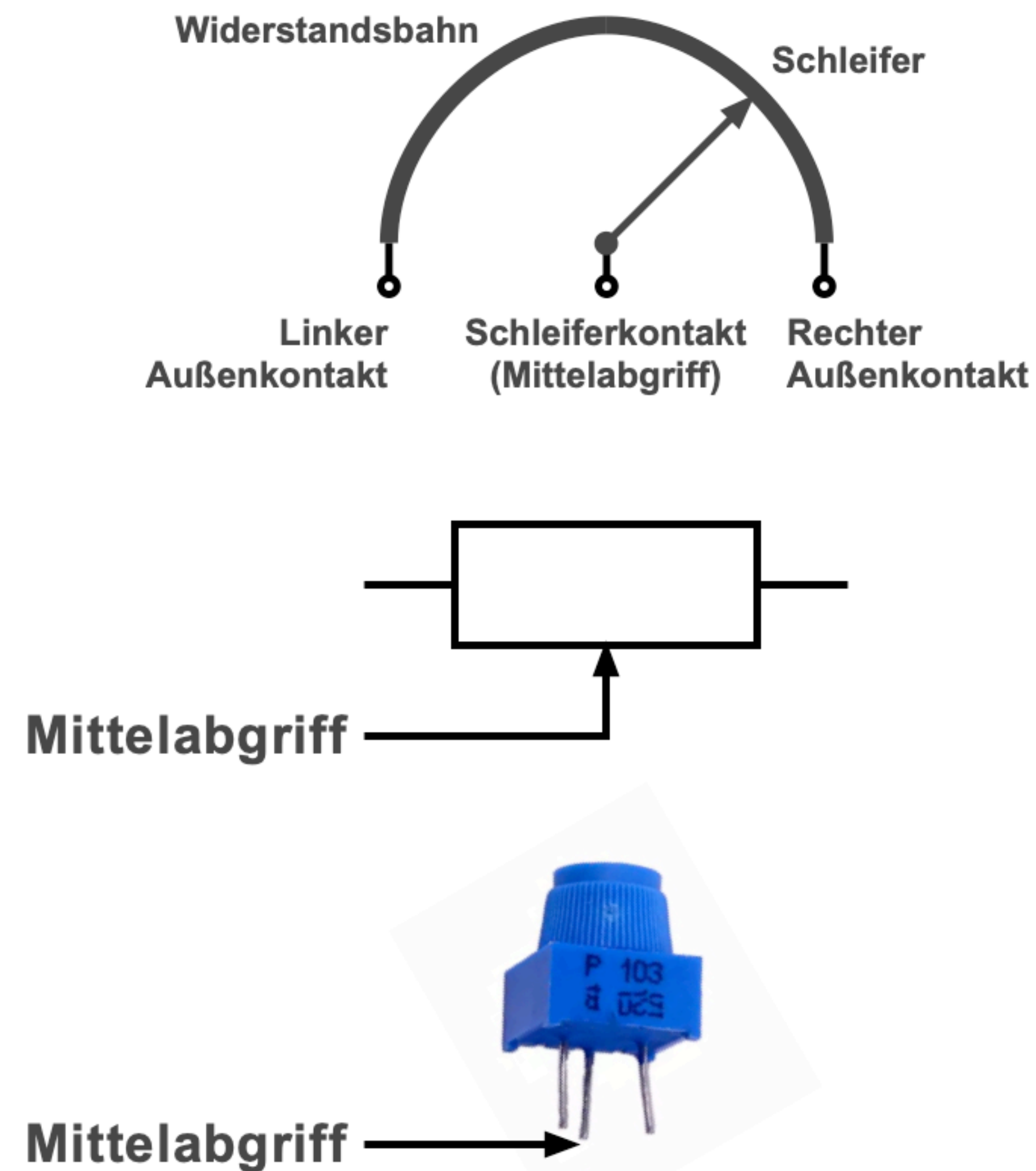
Widerstand mit 470 Ohm



Widerstand mit 10 kOhm



Potentiometer (Poti)



- einstellbarer Widerstand
- Einstellen des Stroms
- Einstellen der Spannung

Ein Potentiometer, kurz Poti, ist ein passives Bauelement dessen Widerstandswert sich stufenlos einstellen lässt. Über den Widerstandskörper wird ein Kontakt (Schleifer) geführt, über dessen Position man einen bestimmten Widerstand einstellen und abgreifen kann. Dazu hat das Potentiometer nicht nur zwei, sondern drei Anschlüsse. Zwei für den Widerstand und den dritten für den Abgriff.

Ein Potentiometer kann man sich wie einen Spannungsteiler aus zwei in Reihe geschalteten Widerständen vorstellen. Mit einem Drehregler stellt man das Verhältnis der beiden Widerstände ein.

Kennzeichnung von Potentiometern (Potis)



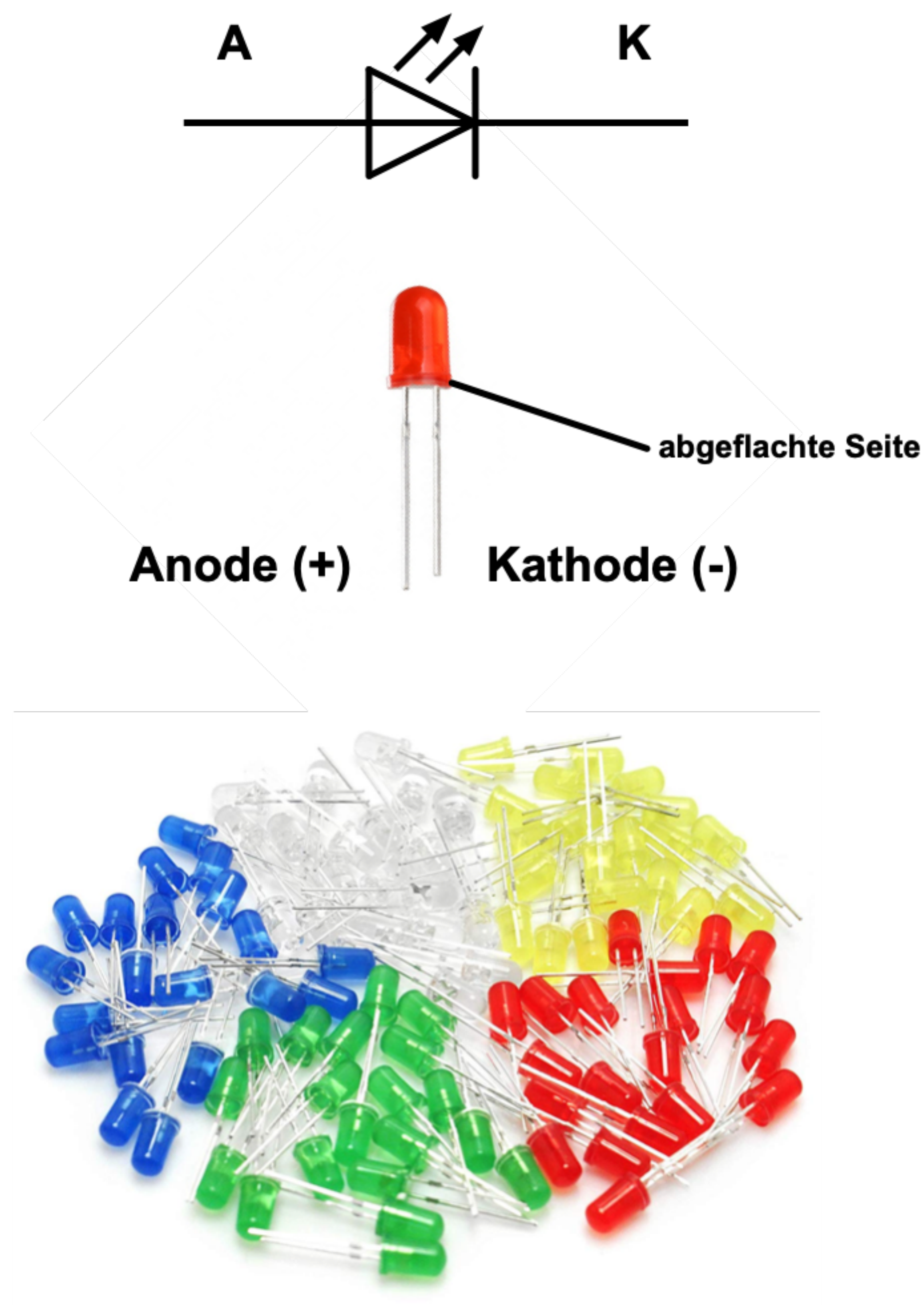
$$\begin{aligned} &= 10.000 \Omega \\ &= 10 \text{ k}\Omega \end{aligned}$$

Manchmal ist auf dem Potentiometer der Widerstandswert aufgedruckt. Manchmal ist die Kennzeichnung kodiert, also eine Kurzform. Dann sind die ersten zwei Ziffern der Wert in Ohm. Eine dritte Ziffer ist der Multiplikator, also die Anzahl der Nullen, die man dem Wert anfügt. Ab drei Nullen rechnet man in Kiloohm (kOhm) um.

Beispiele:

101:	10	x	10	=	100	Ohm	
102:	10	x	100	=	1.000	Ohm	= 1 kOhm
103:	10	x	1.000	=	10.000	Ohm	= 10 kOhm
104:	10	x	10.000	=	100.000	Ohm	= 100 kOhm
105:	10	x	100.000	=	1.000.000	Ohm	= 1 MOhm
151:	15	x	10	=	150	Ohm	
152:	15	x	100	=	1.500	Ohm	= 1,5 kOhm
153:	15	x	1.000	=	15.000	Ohm	= 15 kOhm
154:	15	x	10.000	=	150.000	Ohm	= 150 kOhm
201:	20	x	10	=	200	Ohm	
202:	20	x	100	=	2.000	Ohm	= 2 kOhm
203:	20	x	1.000	=	20.000	Ohm	= 20 kOhm
204:	20	x	10.000	=	200.000	Ohm	= 200 kOhm
501:	50	x	10	=	500	Ohm	
502:	50	x	100	=	5.000	Ohm	= 5 kOhm
503:	50	x	1.000	=	50.000	Ohm	= 50 kOhm
504:	50	x	10.000	=	500.000	Ohm	= 500 kOhm

Leuchtdiode (LED)



Leuchtdioden werden typischerweise mit einem Vorwiderstand in Reihe betrieben, der eine spannungs- und strombegrenzende Wirkung hat. Die Strombegrenzung ist deshalb notwendig, weil der LED-Halbleiter bei zu viel Strom kaputt gehen kann.

- optischer Signalgeber
- optischer Sensor

Eine Leuchtdiode, auch Light Emitting Diode, kurz LED genannt, erzeugt ein Licht in einer bestimmten Farbe, wenn sie von einem Strom durchflossen wird. Dabei verhält sich die LED wie jede andere Halbleiterdiode auch.

Die beiden Anschlüsse werden als Kathode und Anode bezeichnet. Typischerweise sind die beiden Anschlussdrähte einer LED unterschiedlich lang. Der längere von beiden ist die Anode. Der kürzere die Kathode. Außerdem sind die meisten LEDs auf der Kathodenseite abgeflacht. Um das zu erkennen, musst Du ganz genau hinschauen.

Leuchtdioden gibt es in vielen verschiedenen Farben. Am häufigsten kommen Rot, Grün, Gelb, Blau und Weiß vor. Die Farbe wird durch das Halbleitermaterial vorgegeben und zusätzlich ein entsprechend gefärbtes und lichtdurchlässiges Gehäuse verwendet.

Kennzeichnung und Anschlussbelegung von Leuchtdioden



Je nach Hersteller, Farbe und Typ können Leuchtdioden sehr unterschiedliche elektrische Werte aufweisen. Gemeint sind die Durchlassspannung und der Durchlassstrom. Beide Werte sind wichtig zur Berechnung des strombegrenzenden Vorwiderstands.

Ein Vorwiderstand begrenzt den Strom in Durchlassrichtung typischerweise auf etwa 10 mA. Die Spannung je nach LED-Typ auf 1,8 bis 2,2 Volt, oder höher. Manche LEDs haben eine Durchlassspannung von 3 oder sogar 4 Volt.

An 9 Volt empfiehlt sich ein Widerstandswert zwischen 1 kOhm bis 20 kOhm.

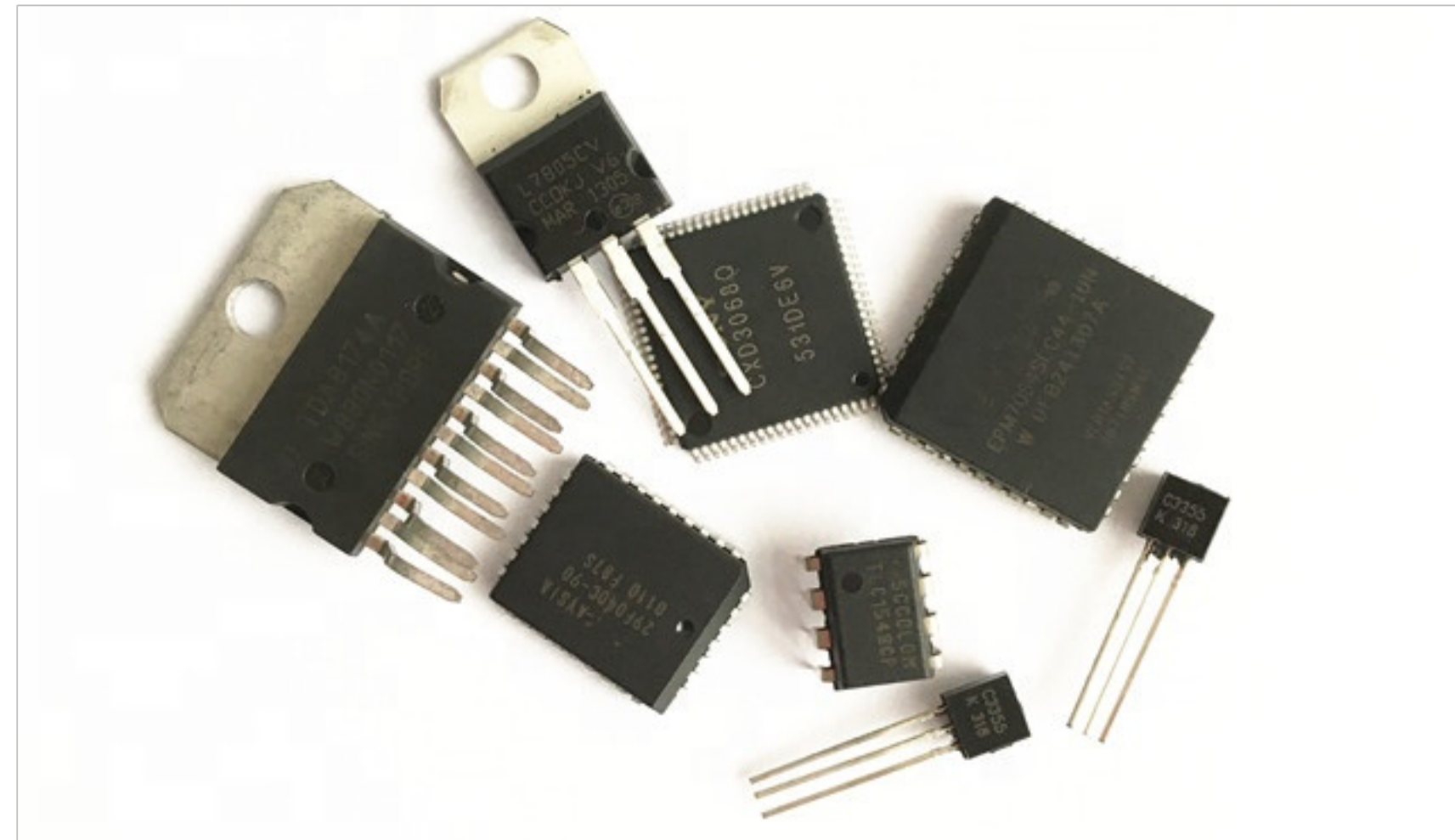
Bei einem deutlich kleineren Widerstand als 1 kOhm kann die LED kaputt gehen. Bei einem deutlich zu großen Widerstand als 20 kOhm leuchtet die LED zu schwach oder gar nicht.

Weil sich eine LED wie jede andere Halbleiterdiode verhält, gibt es eine Sperrrichtung und eine Durchlassrichtung. Soll eine LED leuchten, muss sie in Durchlassrichtung angeschlossen sein. Also die Kathode (K) an Minus (-) und die Anode (A) an Plus (+).

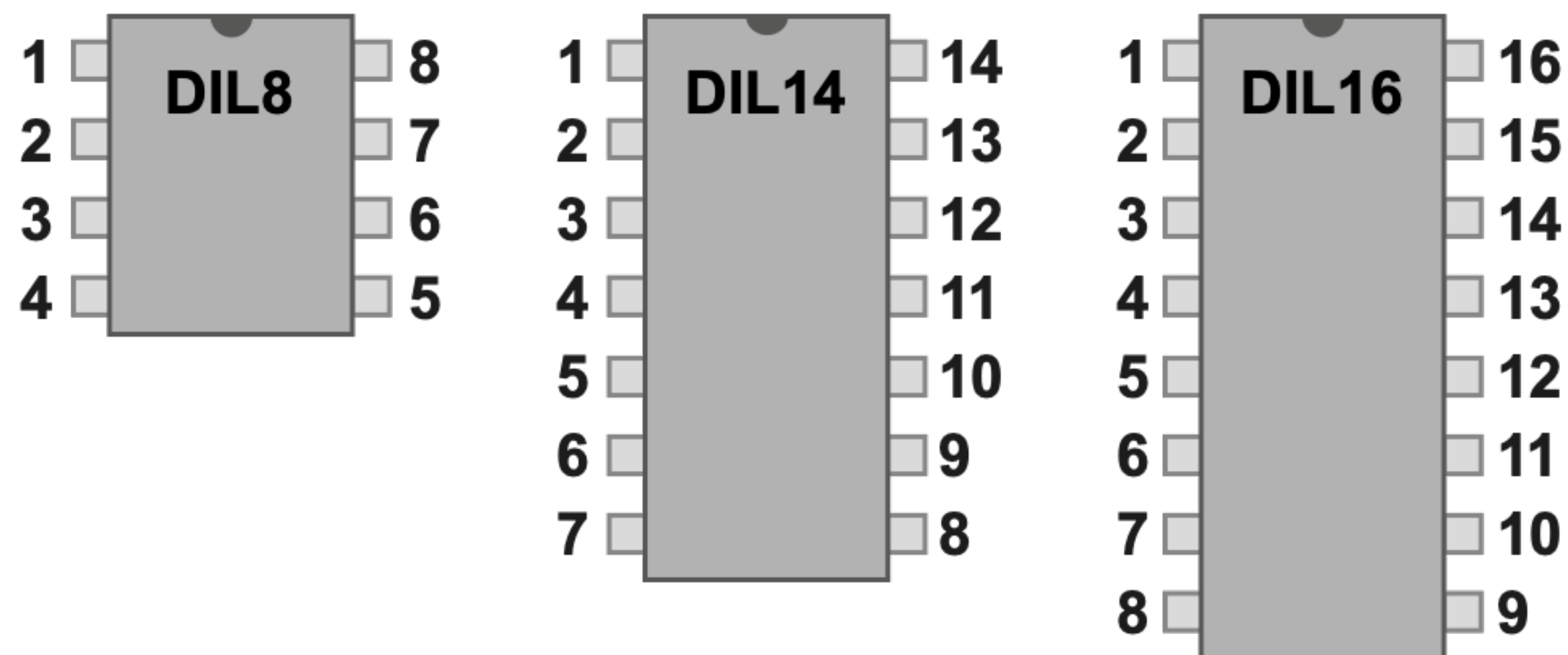
Einfach zu merken: Das Pluszeichen hat einen Strich mehr als das Minuszeichen und deshalb ist der Anschlussdraht der Anode etwas länger. Außerdem sind die meisten LEDs auf der Minusseite abgeflacht, wie eben ein Minus, oder das "K" der Kathode.

Beim Schaltzeichen kann man sich das so merken: Das Schaltzeichen hat wegen dem Querbalken die Form des Buchstabens "K". Das Dreieck hat eine Ähnlichkeit mit dem Buchstaben "A". Beim Querbalken ist der Anschluss die Kathode und auf der anderen Seite die Anode. Die Anode zeigt vom Pluspol weg und zum Minuspol hin, was der technischen Stromrichtung entspricht. Und somit wird die Anode am Pluspol und die Kathode am Minuspol angeschlossen.

Integrierte Schaltkreise / Integrated Circuits (IC)



- komplexe Standard-Schaltung als ein Bauteil
- Platz und Kosten sparen durch mehrfach integrierte Funktionen
- geringerer Schaltungsaufwand

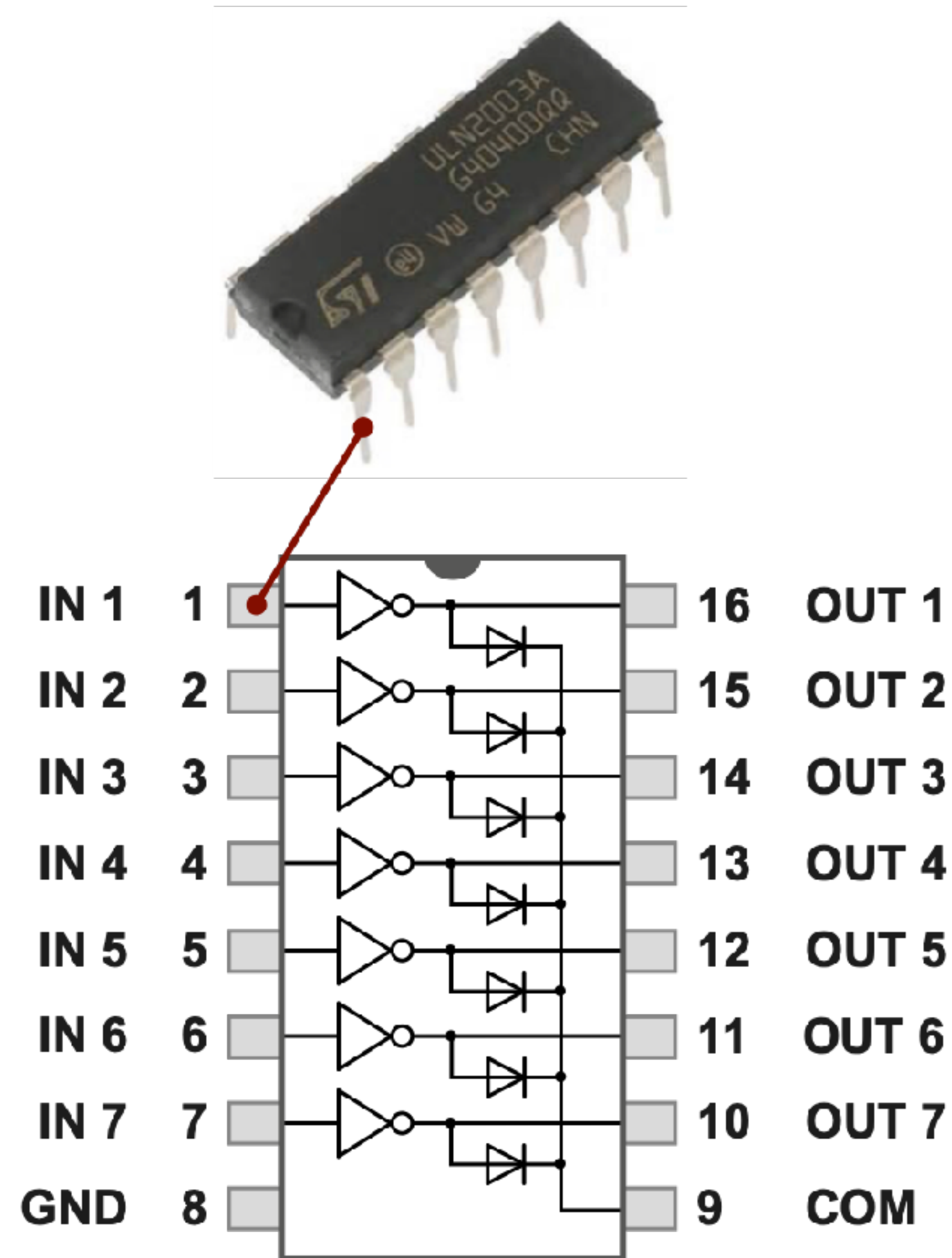


Wenn man auf die beschriftete Seite des ICs schaut, kann man an einer kurzen Seite eine Einkerbung entdecken. Diese Einkerbung kennzeichnet „oben“. Links von der Einkerbung beginnt die Nummerierung der Pins von 1 an nach unten gegen den Uhrzeigersinn zu zählen.

Viele Schaltungen oder Schaltungsteile kommen in der praktischen Elektronik immer wieder vor. Um diese teilweise komplexen Schaltungen nicht immer wieder neu aufbauen oder erfinden zu müssen, werden sie in eine integrierte Schaltung (IS = Integrierter Schaltkreis) zusammengefasst und in einem Gehäuse vergossen.

Fast alle ICs gibt es in unterschiedlichen Gehäuseformen. Die unterscheiden sich nicht nur in der Größe, sondern hauptsächlich in der Art der Anschlusspins und wie die mit der restlichen Schaltung verbunden werden. Also beispielsweise, ob das IC lötl- oder steckbar ist. In der Hobby-Elektronik werden ICs gerne mit DIL- bzw. DIP-Gehäuse verwendet. Diese eignen sich zum Stecken in eine Fassung und zum Verlöten auf eine Platine.

ULN2003A - Darlington Transistor Array (IC)



- Eingänge für TTL-Signale (5 Volt)
- Ausgänge für bis 50 Volt
- Ausgänge mit Freilaufdiode für induktive Lasten (z. B. Motor und Relais)

Ein ULN2003A ist ein integrierter Schaltkreis mit 7 bipolaren NPN-Darlington-Transistoren mit offenem Kollektor und gemeinsamen Emitter. Damit kann man eine Spannung bis 50 Volt (V) und einen Strom bis 500 Milliampere (mA) pro Ausgang schalten. Eine Besonderheit ist die bereits integrierte Freilaufdiode an den Ausgängen. Dadurch kann man problemlos Relais, Motoren und andere induktive Lasten schalten.

Wenn Du an einem Ausgang des ULN2003A ein Relais oder einen Motor anschließt, dann verbinde auch den COM-Anschluss (Pin 9) mit der Versorgungsspannung der angeschlossenen Verbraucher. Dadurch aktivierst Du die integrierten Freilaufdioden, die die Darlington-Stufen vor der Rückwirkung induktiver Lasten schützen.

Die Eingänge eines ULN2003A eignen sich für die Ansteuerung mit TTL- (Transistor-Transistor-Logik) und CMOS-Signalen (Komplementär-Metalloxid-Halbleiter) bis maximal 5 Volt. mit einem Transistor beschaltet werden soll.

ULN2003A: Beschaltung

Eingänge beschalten

Die Eingänge arbeiten mit Standard-TTL-Signalen. Also bei High-Pegel mit 5 Volt und bei Low-Pegel mit 0 Volt. Typischerweise wird ein TTL-Eingang eine anliegende Spannung bis 2 Volt runter als High-Pegel erkennen. Das bedeutet, dass ein High-Pegel eines GPIO-Ausgangs mit 3,3 Volt vom ULN2003A-Eingang als High-Pegel erkannt wird.

Ausgänge beschalten

Welche Spannung geschaltet werden soll hängt von den angeschlossenen Verbrauchern ab. Diese Spannung muss in der Regel aus einer eigenen Spannungsversorgung bereitgestellt werden.

Aufgrund der integrierten Darlington-Stufe haben die Ausgänge bei einem Low-Pegel nie ganz 0 Volt, sondern etwa 0,9 Volt. Unter Umständen ist das beim Dimensionieren nachfolgender Schaltungsteile zu berücksichtigen.

Werden alle 7 Transistorstufen dauerhaft durchgeschaltet (Tastverhältnis 100%), darf jede Stufe nur mit max. 160 mA belastet werden.

COM-Anschluss (Pin 9): Freilaufdiode aktivieren

Der Anschluss Pin 9 (COM) ist der gemeinsame Anschluss der integrierten Freilaufdioden. Dieser Anschluss muss nur dann beschaltet werden, wenn an den Ausgängen induktive Lasten angeschlossen sind. Zum Beispiel Relais oder Motoren. Sollte das der Fall sein, dann wird Pin 9 typischerweise mit der Versorgungsspannung der angeschlossenen Verbraucher verbunden.

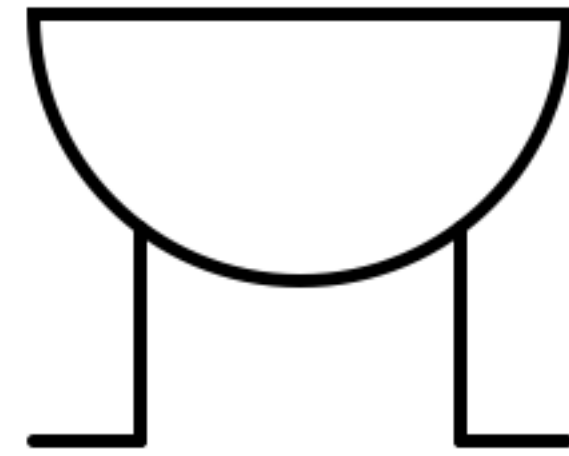
Masse bzw. Ground (GND)

Der Anschluss Pin 8 ist der gemeinsame Masse-Anschluss bzw. Ground (GND) für die sieben Eingänge und die sieben Schaltstufen. In der Regel muss auch der Ground einer separaten Versorgungsspannung damit verbunden werden.

Hinweis

Es empfiehlt sich einen Blick ins Datenblatt zu werfen.

Summer (Buzzer)



Bei einem Summer muss man in der Regel auf die **Polarität** achten. Dabei ist der positive Pol mit einem Plus (+) oben auf dem Gehäuse gekennzeichnet. In der Regel ist das Anschlussbein beim Pluspol länger.

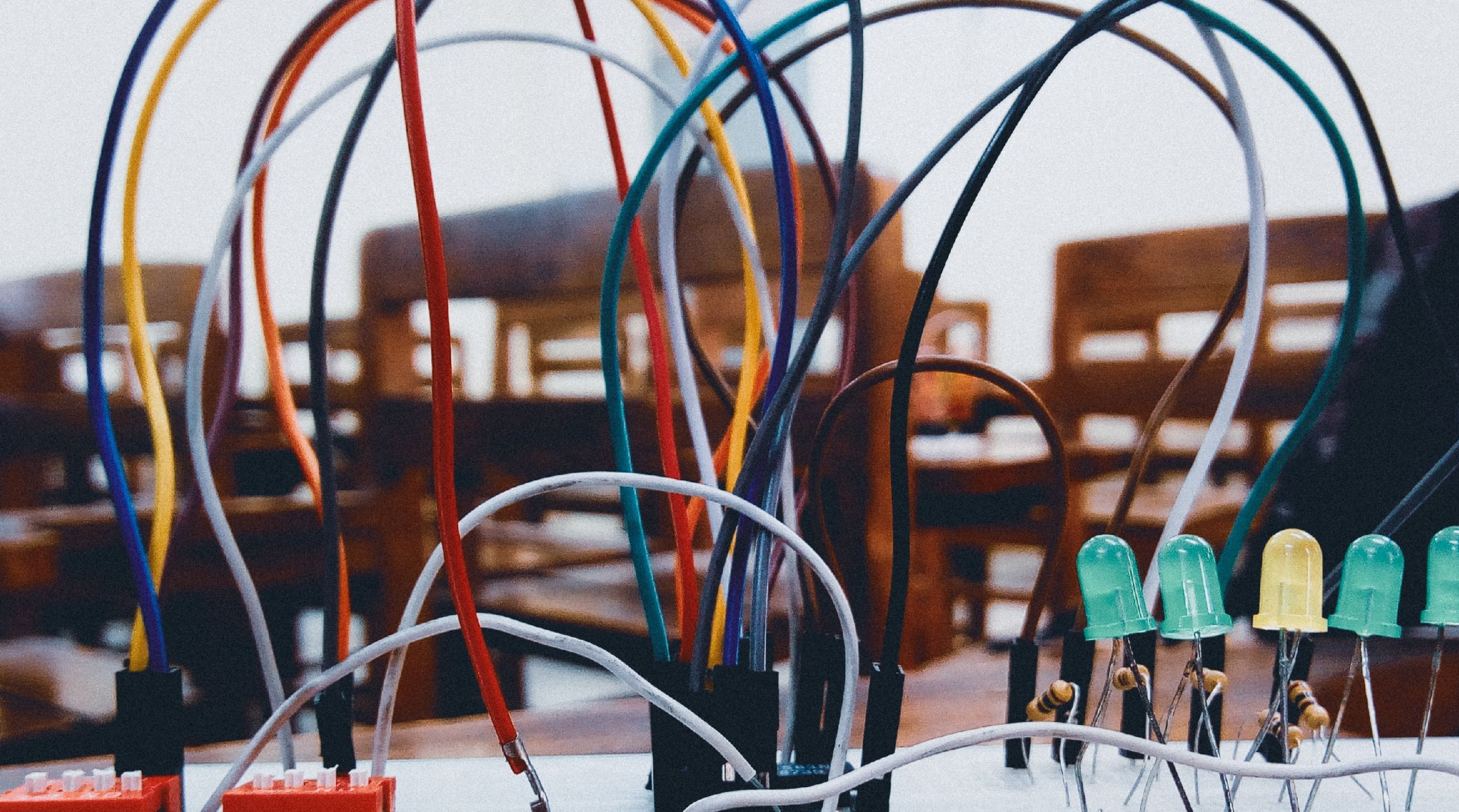
Die **Betriebsspannung** der marktüblichen Summer ist von der Farbe des bedruckten Etiketts abhängig. Die Farbe **Blau** deutet auf **5 Volt** hin. Die Betriebsspannung darf aber auch darüber oder darunter liegen.

- akustischer Signalgeber
- Tonausgabe mit festgelegter Frequenz (Tonhöhe)

Ein Summer, auch Buzzer genannt, ist ein akustischer Signalgeber. Er wandelt ein elektrisches Signal in ein akustisches Signal um. Das heißt, es erklingt ein Ton mit einer festgelegten Frequenz, die man nicht ändern kann. Über die Zeit gesehen, empfinden wir diesen Ton als nervig, weshalb er abschaltbar sein soll.

Ein Summer ist einfach zu benutzen. Er muss nur an eine Spannung angelegt werden und schon „summt“ er. Der Nachteil dabei, die Frequenz und damit die Tonhöhe können nicht beeinflusst werden. Die ist festgelegt. Desweiteren hat man nur wenig Einfluss auf die Lautstärke.

Das Etikett mit der Beschriftung „REMOVE SEAL AFTER WASHING“ dient zum Schutz des Summers. Nach der maschinellen Produktion werden Platinen „gewaschen“. Danach kann das Etikett entfernt werden.



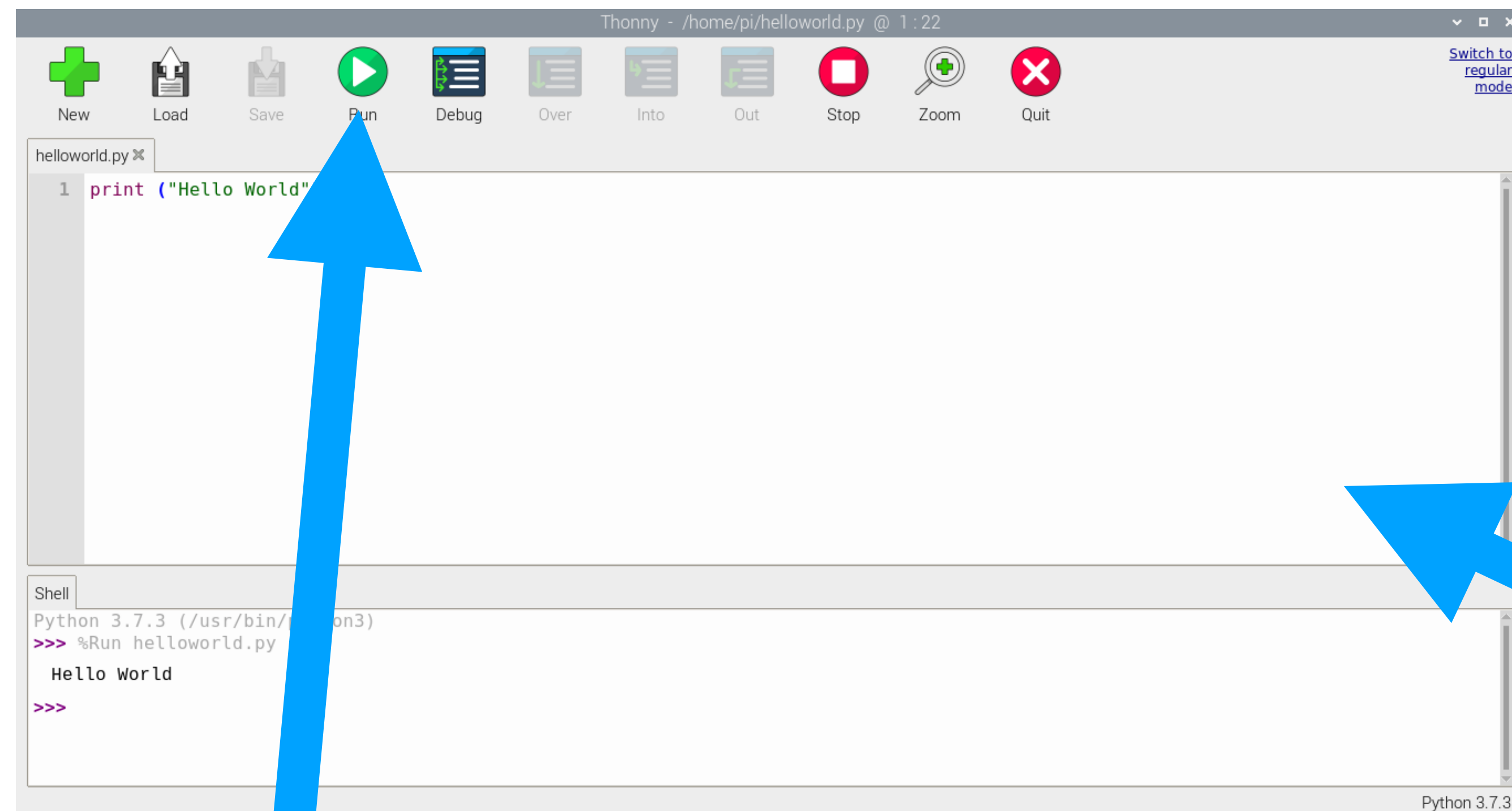
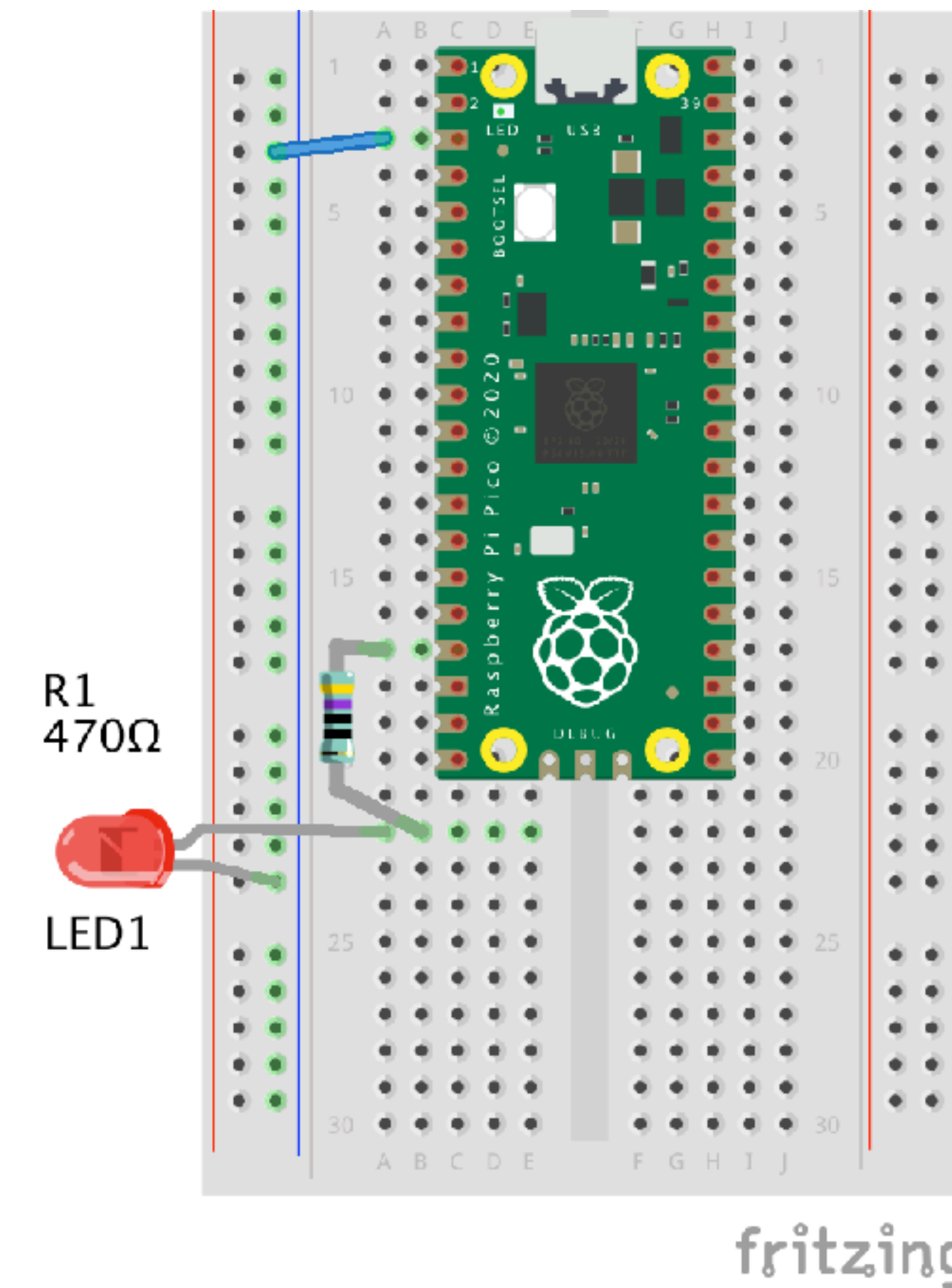
Experimente und Anwendungen

Experimente und Anwendungen mit MicroPython

- Vorgehensweise beim Programmieren (1)
- Vorgehensweise beim Programmieren (2)
- Onboard-LED einschalten und ausschalten
- Onboard-LED blinken lassen
- Externe LED einschalten und ausschalten
- Externe LED blinken lassen
- Summer einschalten und ausschalten
- LED-Helligkeit steuern
- Lichtsteuerung für Lichteffekte mit LEDs
- LED mit Taster einschalten und ausschalten
- Taster-Zustand auswerten und mit einer LED anzeigen
- Taster-Zustand invertieren (Inverter/NICHT-Funktion)
- Reaktionsspiel mit zwei Tastern und LEDs
- Alarmkontakt für Alarmanlage
- LED-Ampel-Steuerung
- LED-Lauflicht
- Binärer Zähler mit LEDs
- Zufallszahl generieren
- Elektronischer Würfel
- Temperatur mit dem integrierten Temperatursensor messen
- EIN/AUS-Temperaturregelung (Zweipunktregelung)
- Schwellwert-Temperaturregelung
- Stromausfall-Monitor bei Abwesenheit

Vorgehensweise beim Programmieren (1)

1. Zuerst die Schaltung aufbauen



2. Dann den Programmcode ins Textfeld kopieren und speichern

```
from machine import Pin  
from utime import sleep
```

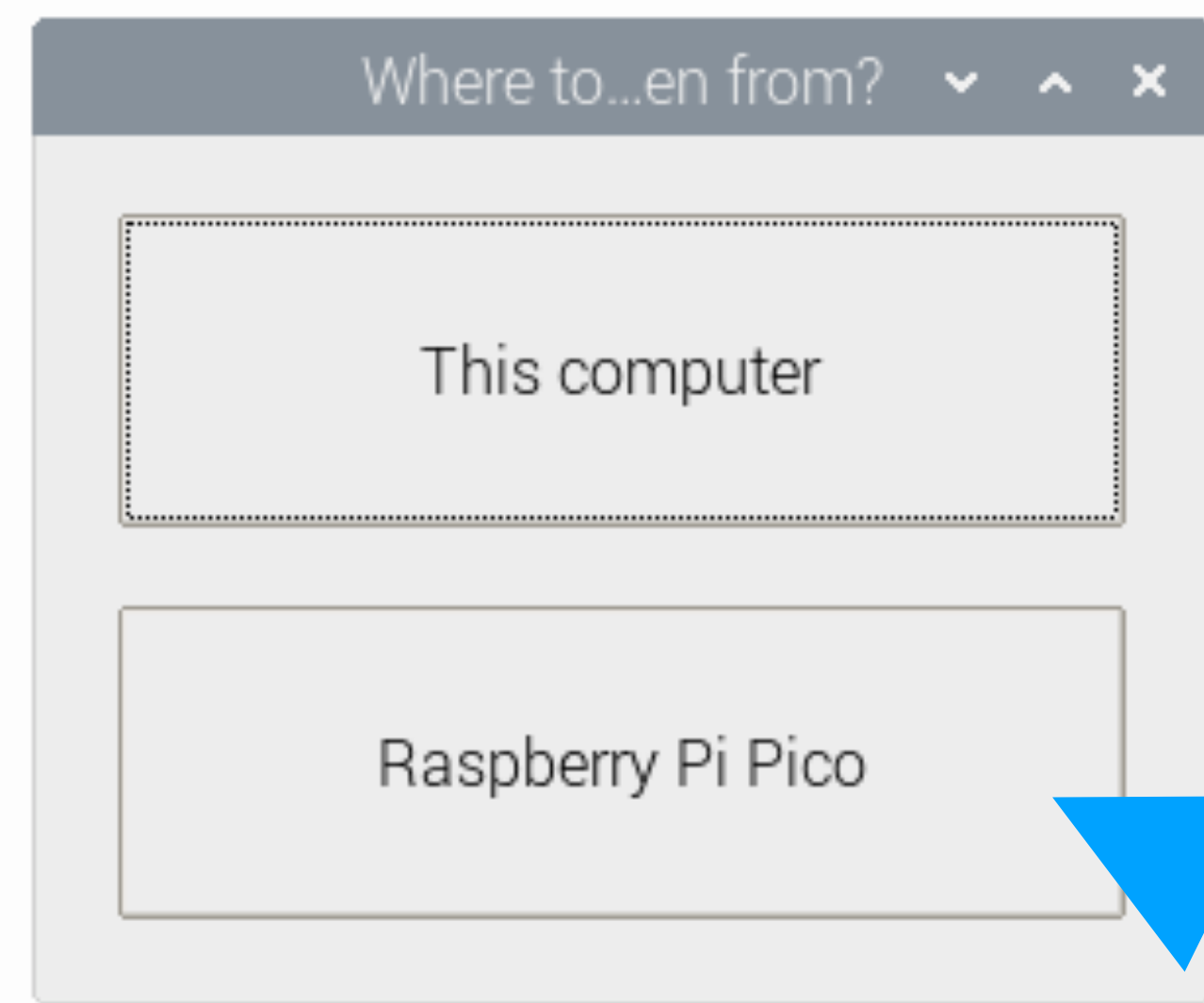
```
led = Pin(25, Pin.OUT)
```

```
led.on()  
sleep(5)  
led.off()
```

3. Danach das Programm ausführen

4. Funktion von Aufbau und Programm prüfen

Vorgehensweise beim Programmieren (2)



Programmcode mit der Dateiendung „.py“ auf dem Pico speichern.

1. Zuerst die Schaltung aufbauen. Zum Beispiel auf einem Steckbrett.
2. Im Thonny-Editor eine neue Datei öffnen, den Programmcode ins Textfeld kopieren und die Datei mit der Dateiendung „.py“ speichern und als Speicherort den Raspberry Pi Pico auswählen.
3. Das Programm ausführen bzw. starten.
4. Die Funktion des Programms prüfen.

Auch wenn ein Programm nicht mehr läuft, leuchten die Leuchtdioden immer noch, die beim letzten Programmschritt geleuchtet haben. Es bleiben also „Zustände“ im Speicher erhalten. Wenn man dann ein anderes Programm startet, dann leuchten manche LEDs einfach weiter.

Es empfiehlt sich, den Raspberry Pi Pico bei einem neuen Programm vorher einmal auszustecken und neu einzustecken. Dabei werden alle Rückstände des alten Programms aus dem Speicher gelöscht.

Hinweis: Nach dem erneuten Einstecken muss die Verbindung zum Raspberry Pi Pico durch Klicken auf „Stop“ zurückgesetzt werden.



```
Couldn't find the device automatically.  
Check the connection (making sure the device is not in bootloader mode) or choose  
"Configure interpreter" in the interpreter menu (bottom-right corner of the window)  
to select specific port or another interpreter.
```

Online-Workshop: Picobello

Du musst das
nicht alleine
machen!



Im Online-Workshop PicoBello

- ... Lernst Du Programmieren mit dem Raspberry Pi Pico.
- ... Wir arbeiten mit den Teilnehmern gemeinsam.
- ... Bekommst Du Unterstützung bei individuellen Problemen.

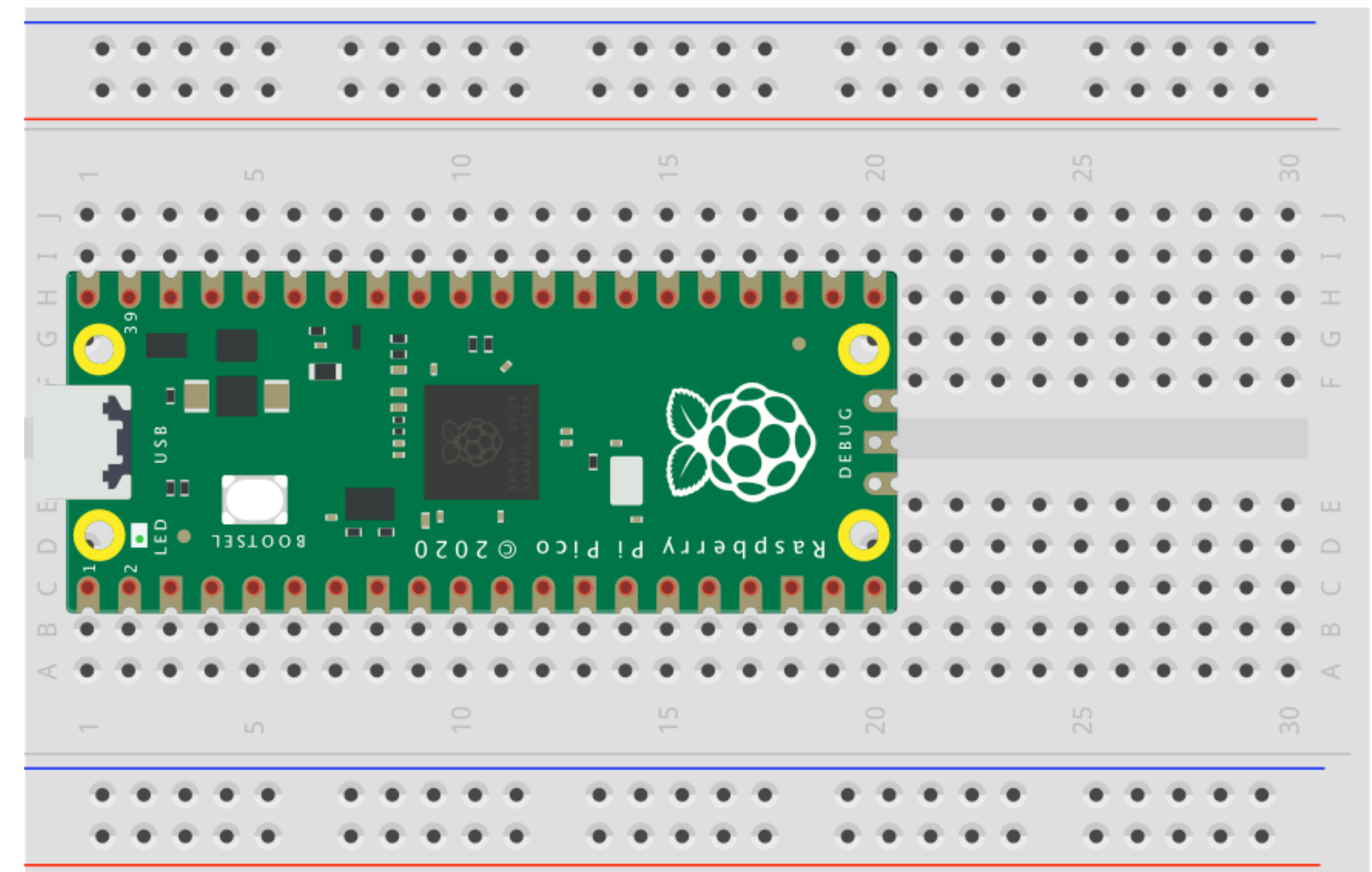
<https://www.elektronik-kompendium.de/service/events/>

Onboard-LED einschalten und ausschalten

Mit diesem Aufbau geht es erst einmal nur darum, den Raspberry Pi Pico mit einem allerersten Programm auszuprobieren. Dafür benutzen wir die Onboard-LED des Picos. Damit kann man den Pico testen, prüfen und erste Experimente durchführen. Ohne den Aufwand einer externen Beschaltung.

Es geht nur darum, die Onboard-LED auf dem Raspberry Pi Pico einzuschalten und wieder auszuschalten.

Der Vorteil dieses Aufbaus ist, dass außer dem Raspberry Pi Pico nichts weiter erforderlich ist.



```
# Bibliotheken laden
from machine import Pin
from time import sleep

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT)

# LED einschalten
led_onboard.on()

# 5 Sekunden warten
sleep(5)

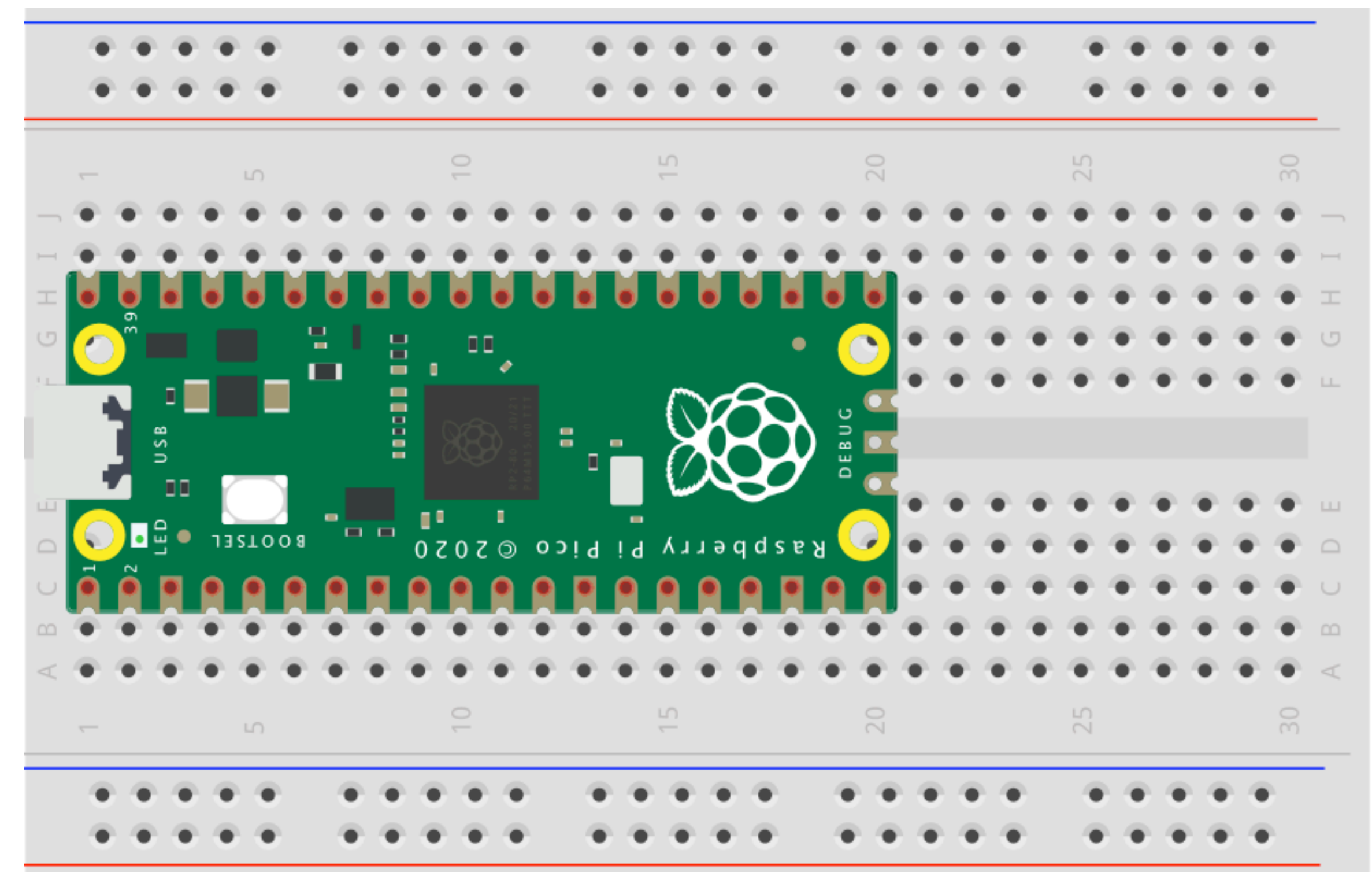
# LED ausschalten
led_onboard.off()
```


Onboard-LED blinken lassen (1)

Elektronik macht immer dann am meisten Spaß, wenn es blinkt und blitzt. Aber bitte nicht zu viel davon. Wenn es zu sehr blitzt, dann ist es meistens kaputt. Deshalb beschränken wir uns sicherheitshalber auf das Blinken einer Leuchtdiode.

Damit eine LED blinkt oder blitzt gibt es in der Elektronik ein paar Grundschaltungen. Die folgenden Programmcodes ersetzen die Funktion dieser elektronischen Schaltungen. Die Leuchtdiode wird per Software gesteuert, was uns viele Möglichkeiten der Steuerung eröffnet, während wir bei einer elektronischen Schaltung auf deren Funktion beschränkt bleiben. Mit MicroPython lässt sich das Blinken der Onboard-LED mit wenigen Zeilen erledigen.

Dabei gibt es nicht nur einen Weg, wie man die Onboard-LED zum Blinken bringen kann. Es gibt gleich mehrere davon. Und genau das sollst Du ausprobieren. Und es wäre denkbar, dass es noch weitere Lösungen gibt, die hier nicht berücksichtigt wurden.



Onboard-LED blinken lassen (2)

Nach dem Start der Programme beginnt die Onboard-LED zu blinken. Beim ersten Programm wird die LED nach einer bestimmten Wartezeit eingeschaltet und ausgeschaltet.

Wichtig: Beim Kopieren oder Abtippen auf die Einrückungen (4 Leerzeichen) achten.



Das zweite Programm nutzt die Toggle-Funktion, die zwischen zwei Zuständen hin- und herwechselt, unabhängig davon, welcher Zustand gerade herrscht.

```
from machine import Pin
from time import sleep

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT)

# Wiederholung (Endlos-Schleife)
while True:
    # LED einschalten
    led_onboard.on()
    # halbe Sekunde warten
    sleep(0.5)
    # LED ausschalten
    led_onboard.off()
    # 1 Sekunde warten
    sleep(1)
```

```
from machine import Pin
from time import sleep

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT)

# Wiederholung (Endlos-Schleife)
while True:
    # LED-Zustand wechseln (EIN/AUS)
    led_onboard.toggle()
    # 1 Sekunde warten
    sleep(1)
```

Onboard-LED blinken lassen (3)

Die beiden vorherigen Programme sind sicherlich zweckmäßig. Allerdings entspricht die Lösung mit einer Schleife eher dem Stil eines Anfängers. Wir wollen aber etwas lernen und auch unsere Programmierfähigkeiten verbessern. Deshalb nutzen wir hier die Lösung mit einem Timer. Das ist eine interne Funktion des Mikrocontrollers.

Zu beachten ist hier, dass der Wechsel von ein nach aus nicht über die Zeit, sondern die Frequenz in Hertz (Hz) eingestellt wird. Eine Frequenz gibt die Zahl von etwas an, das in einer Sekunde durchlaufen wird. Soll eine Leuchtdiode 2 mal in einer Sekunde blinken, dann gibt man 2 Hertz an.

Die Timer-Lösung hat den Vorteil, dass sie unabhängig von anderen Programmabläufen funktioniert. Bei einer Schleifen-Lösung blinkt die LED nur dann, wenn die Schleife nicht unterbrochen oder gestoppt wird.

```
# Bibliotheken laden
from machine import Pin, Timer

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT)

# Definition einer Funktion
def blink(timer):
    led_onboard.toggle()

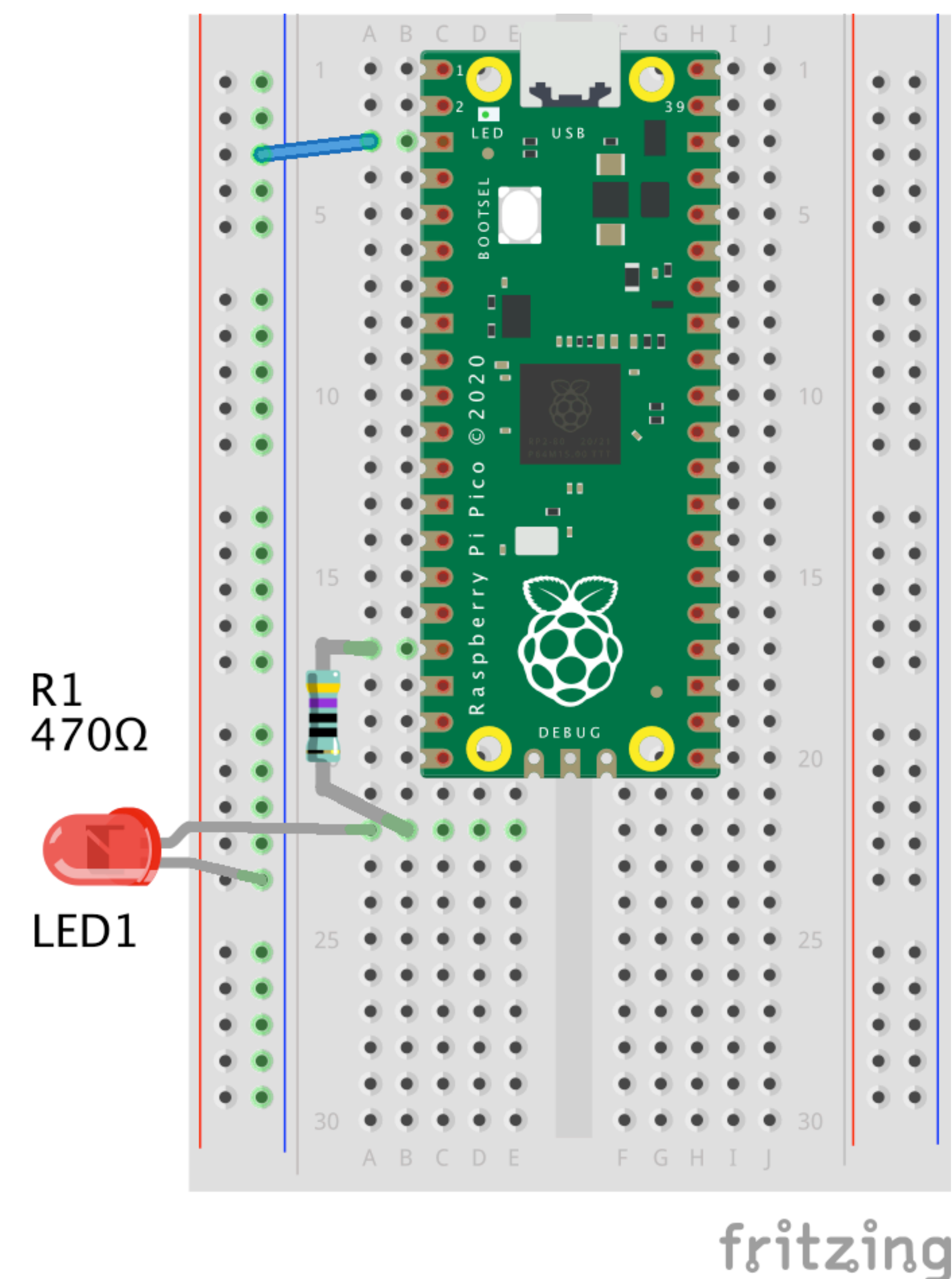
# Timer initialisieren
Timer().init(freq=2.5, callback=blink)
```

Externe LED einschalten und ausschalten (1)

In diesem Aufbau geht es darum, an einem Raspberry Pi Pico eine LED richtig anzuschließen und mit einem Programm die LED einzuschalten und wieder auszuschalten. Das ist nicht weiter schwer. Wir brauchen dazu nur eine LED (egal welche Farbe) und einen Widerstand. Auf einem Steckbrett werden die Bauteile zusammen mit den GPIO-Pins verbunden. Es bietet sich an, auch mal andersfarbige Leuchtdioden auszuprobieren.

Die folgende Schaltung und das Programm sind denkbar einfach. Es ermöglicht Dir, Dich mit dem Verbinden von nur zwei Bauteilen auf dem Steckbrett vertraut zu machen und ein erstes Programm in MicroPython für eine externe Beschaltung zu schreiben.

Wenn Du das alles geschafft hast, dann können Dich die nächsten Experimente nicht mehr schrecken.



LED1: Leuchtdiode, rot, gelb oder grün

R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)

Externe LED einschalten und ausschalten (2)

Nach dem Start des Programms leuchtet die Leuchtdiode für 5 Sekunden. Dann geht sie wieder aus.

```
# Bibliotheken laden
from machine import Pin
from time import sleep

# Initialisierung von GPIO13 als Ausgang
led = Pin(13, Pin.OUT)

# LED einschalten
led.on()

# 5 Sekunden warten
sleep(5)

# LED ausschalten
led.off()
```

Externe LED blinken lassen (1)

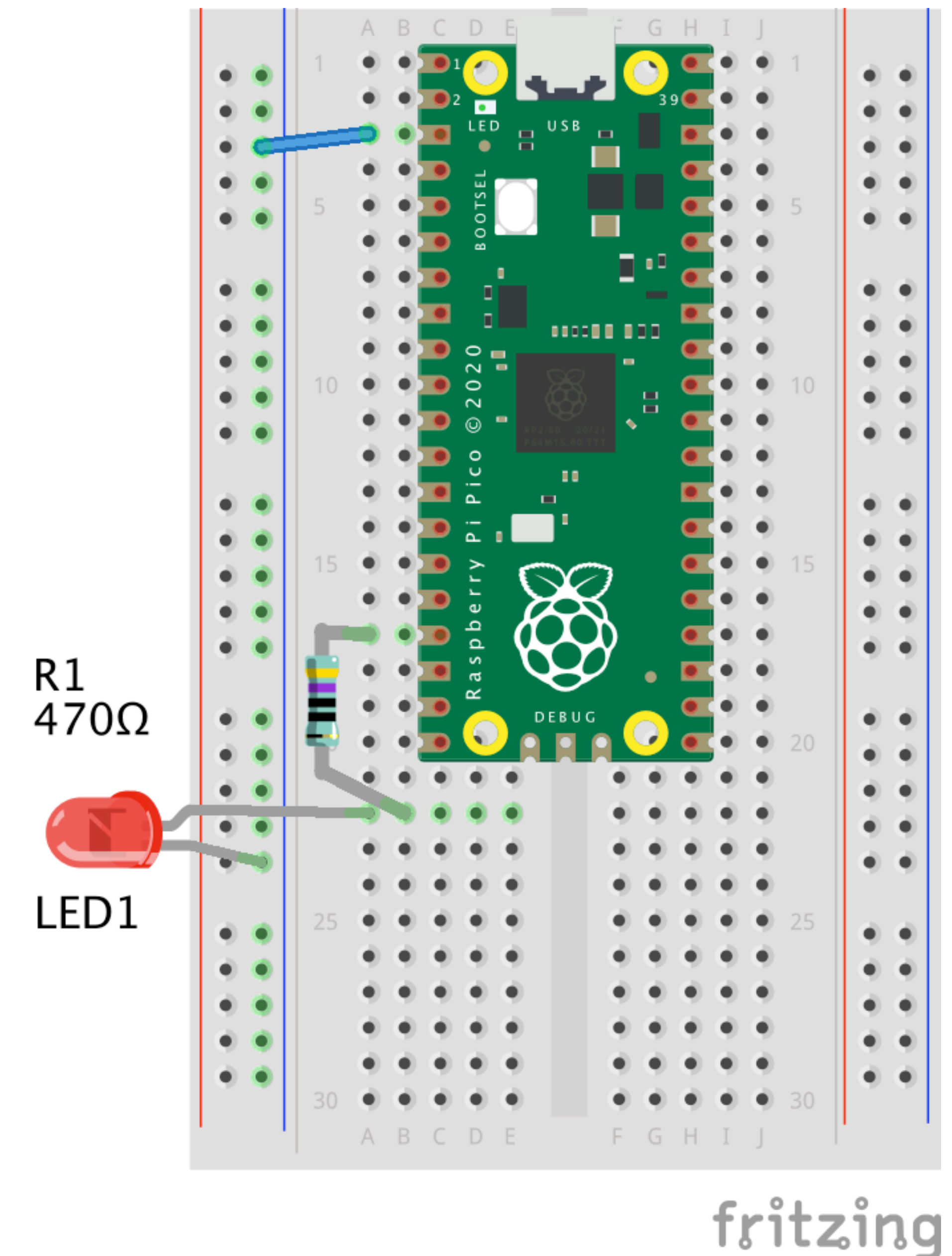
Am Besten ist Elektronik immer dann, wenn es blinkt und blitzt. Aber wenn es zu sehr blitzt, ist danach meistens etwas kaputt. Deshalb beschränken wir uns sicherheitshalber auf das Blinken einer Leuchtdiode.

Für den Raspberry Pi Pico lässt sich in MicroPython das Blinken einer Leuchtdiode in wenigen Zeilen schreiben.

Damit eine LED blinkt oder blitzt gibt es in der Elektronik nur ein paar Grundschaltungen, die man je nach Blinkfrequenz variieren kann oder sogar bevorzugt. Die folgenden Programmcodes ersetzen die Funktion dieser Grundschaltungen. Das einzige, was an Hardware bleibt, ist die Leuchtdiode mit dem Vorwiderstand. Und natürlich der Raspberry Pi Pico.

Die Leuchtdiode wird per Software gesteuert.

Es gibt nicht nur einen Weg, wie man eine Leuchtdiode zum Blinken bringen kann. Es gibt gleich mehrere Möglichkeiten. Und genau die solltest Du alle ausprobieren. Und es wäre denkbar, dass es noch weitere Lösungen gibt, die hier nicht berücksichtigt wurden.



LED1: Leuchtdiode, rot, gelb oder grün

R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)

Externe LED blinken lassen (2)

Nach dem Start der Programme beginnt die LED zu blinken. Beim ersten Programm wird die LED nach einer bestimmten Wartezeit eingeschaltet und ausgeschaltet.

```
from machine import Pin
from utime import sleep

# Initialisierung von GPIO13 als Ausgang
led = Pin(13, Pin.OUT)

# Wiederholung (Endlos-Schleife)
while True:
    # LED einschalten
    led.on()
    # halbe Sekunde warten
    sleep(0.5)
    # LED ausschalten
    led.off()
    # 1 Sekunde warten
    sleep(1)
```

Das zweite Programm nutzt die Toggle-Funktion, die zwischen zwei Zuständen hin- und herwechselt, unabhängig davon, welcher Zustand gerade herrscht.

```
from machine import Pin
from utime import sleep

# Initialisierung von GPIO13 als Ausgang
led = Pin(13, Pin.OUT)

# Wiederholung (Endlos-Schleife)
while True:
    # LED-Zustand wechseln (EIN/AUS)
    led.toggle()
    # 1 Sekunde warten
    sleep(1)
```

Externe LED blinken lassen (3)

Die beiden vorherigen Programme sind sicherlich zweckmäßig. Allerdings entspricht die Lösung mit einer Schleife eher dem Stil eines Anfängers. Wir wollen aber hier etwas lernen und auch unsere Programmierfähigkeiten verbessern. Deshalb nutzen wir hier die Lösung mit einem Timer. Das ist eine interne Funktion des Mikrocontrollers.

Zu beachten ist hier, dass der Wechsel von ein nach aus nicht über die Zeit, sondern die Frequenz in Hertz (Hz) eingestellt wird. Eine Frequenz gibt die Zahl von etwas an, das in einer Sekunde durchlaufen wird. Soll eine Leuchtdiode 2 mal in einer Sekunde blinken, dann gibt man 2 Hertz an.

Die Timer-Lösung hat den Vorteil, dass sie unabhängig von anderen Programmabläufen funktioniert. Bei einer Schleifen-Lösung blinkt die LED nur dann, wenn die Schleife nicht unterbrochen oder gestoppt wird.

```
# Bibliotheken laden
from machine import Pin, Timer

# Initialisierung von GPIO13 als Ausgang
led = Pin(13, Pin.OUT)

# Definition einer Funktion
def blink(timer):
    led.toggle()

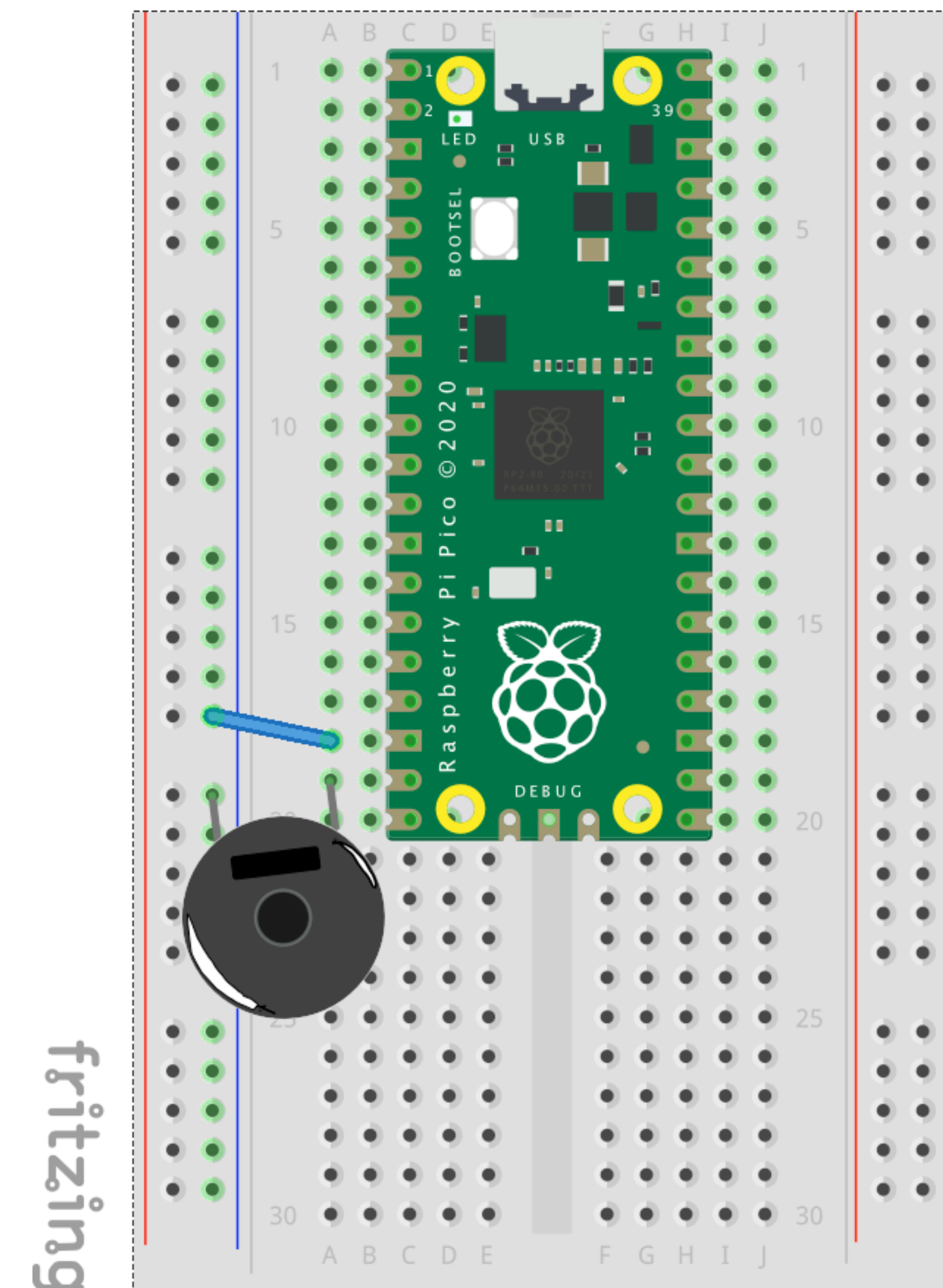
# Timer initialisieren
Timer().init(freq=2.5, callback=blink)
```


Summer einschalten und ausschalten (1)

In diesem Aufbau geht es darum, an einem Raspberry Pi Pico einen Summer richtig anzuschließen und mit einem Programm den Summer einzuschalten und wieder auszuschalten. Das ist nicht weiter schwer. Auf einem Steckbrett werden die Bauteile zusammen mit dem GPIO-Pin verbunden.

Die folgende Schaltung und das Programm sind denkbar einfach. Es ermöglicht Dir, Dich mit dem Verbinden von nur einem Bauteil auf dem Steckbrett vertraut zu machen und ein Programm in MicroPython für eine externe Beschaltung zu schreiben.

Ein aktiver Summer ist gepolt. Das lange Anschlussbein ist der positive Pol, der mit dem GPIO verbunden wird. Das kurze Anschlussbein ist der negative Pol, der mit Ground (GND) verbunden wird.



Summer einschalten und ausschalten (2)

Nach dem Start des Programms gibt der Summer für 3 Sekunden ein akustisches Signal wieder. Dann geht er wieder aus.

```
# Bibliotheken laden
from machine import Pin
from time import sleep

# Initialisierung von GPIO14 als Ausgang
device = Pin(14, Pin.OUT, value=0)

# LED einschalten
print('EIN')
device.on()

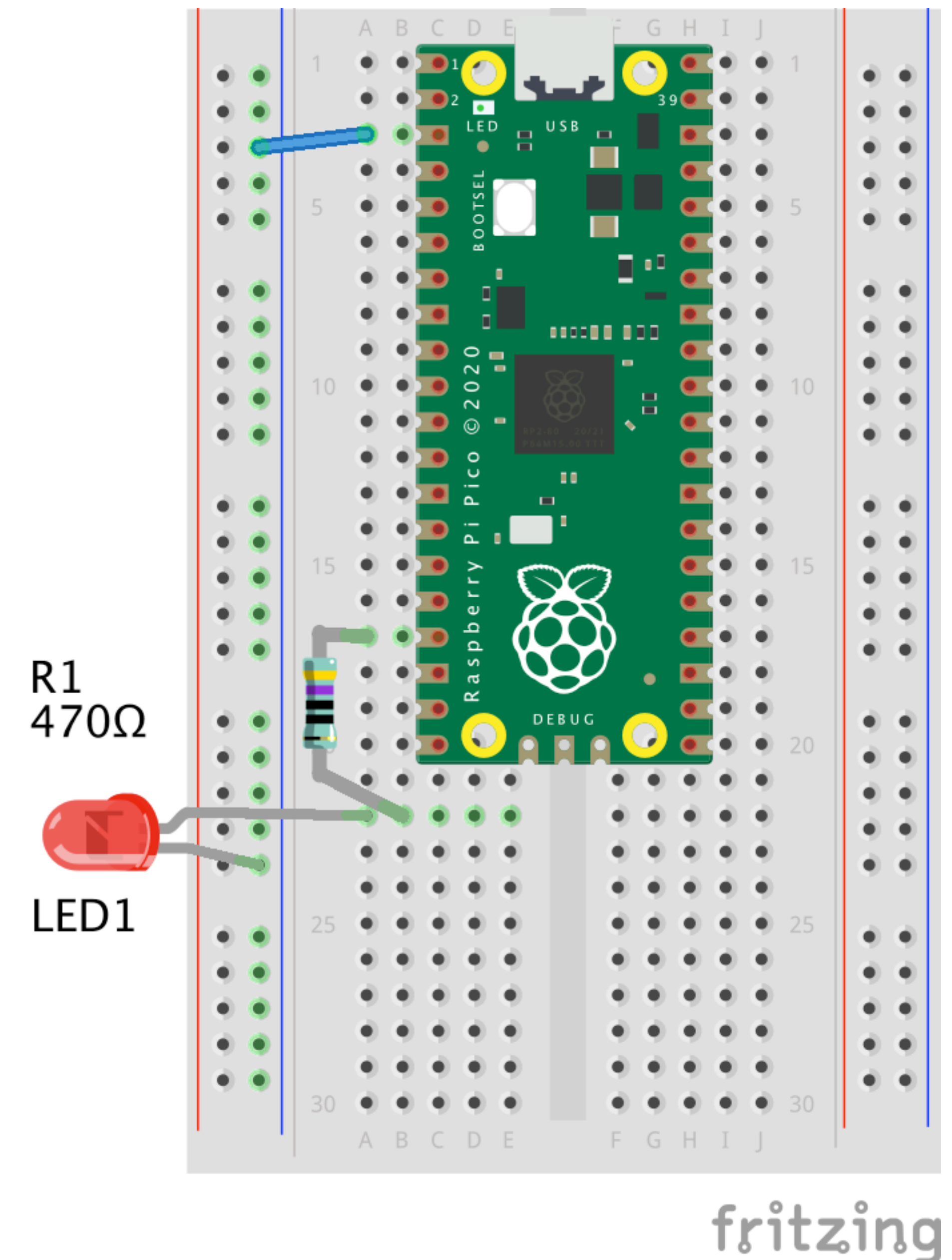
# 3 Sekunden warten
print('.')
sleep(1)
print('.')
sleep(1)
print('.')
sleep(1)

# LED ausschalten
print('AUS')
device.off()
```

LED-Helligkeit steuern (1)

Im Prinzip kennt ein GPIO nur zwei Zustände: „High“ und „Low“. Also „An“ und „Aus“ oder „1“ und „0“. Es handelt sich dabei um die binäre Logik. Werte dazwischen gibt es nicht. Logisch wäre es, dass man mit der Steuerung per GPIO eine LED auch nur ein- und ausschalten kann. Die Helligkeit kann man also nicht einstellen.

Trotzdem gibt es einen Trick mit dem man die Helligkeit einer LED doch steuern kann. Wenn die LED sehr schnell ein- und ausgeschaltet wird, also ein Taktsignal ausgegeben wird. Dieses Taktsignal wird als PWM-Signal realisiert, bei dem die Breite des Pulses über die Zeit eingestellt werden kann. Das heißt, es wird die Dauer des Leuchtens der LED eingestellt. Durch die Trägheit des Aufleuchtens und der Wahrnehmung des menschlichen Auges sieht es so aus, als ob die Helligkeit der Leuchtdiode gesteuert wird.



LED1: Leuchtdiode, rot, gelb oder grün
R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)

LED-Helligkeit steuern (2)

Im Programmcode wird der GPIO für die Leuchtdiode als PWM-Ausgang initialisiert und eine Frequenz von 1.000 Hertz (Hz) eingestellt. Anschließend wird innerhalb der Schleife das Duty-Cycle, das Verhältnis zwischen Impuls und Pause des Signals erhöht. Die Leuchtdiode leuchtet also immer heller, geht aus und wird wieder heller.

Verschiedene Parameter im Programm sorgen dafür, dass die LED pulsiert. Der Programmcode lädt zum Experimentieren ein.

- Die Zeile „sleep(0.1)“ gibt an wie lange in Sekunden zwischen den Veränderungsschritten gewartet werden soll. Vergrößere und verkleinere den Wert von „0.1“. Zum Beispiel auf „0.5“ oder „0.05“.
- Softwareseitig kann das PWM-Signal einen Wert zwischen 0 und 65.535 annehmen. Bei 0 ist die LED aus. Bei 65.535 leuchtet die LED am hellsten. Allerdings leuchtet die LED schon wesentlich vor 65.535 sehr hell. Verkleinere den Wert von „65535“ so weit, wie es wirklich nötig ist.
- Von 0 (LED aus) nach 65.535 (LED leuchtet voll) wird in 3000er Schritten die Helligkeit gesteigert. Vergrößere den Wert von „3000“ und verkleinere ihn, um den Effekt auf die Helligkeitsänderung herauszufinden.

```
# Bibliotheken laden
from machine import Pin, PWM
from time import sleep

# Initialisierung von GPIO13 als PWM-Ausgang
led = PWM(Pin(13))

# PWM-Einstellung: Frequenz in Hertz (Hz)
led.freq(1000)

i = 0

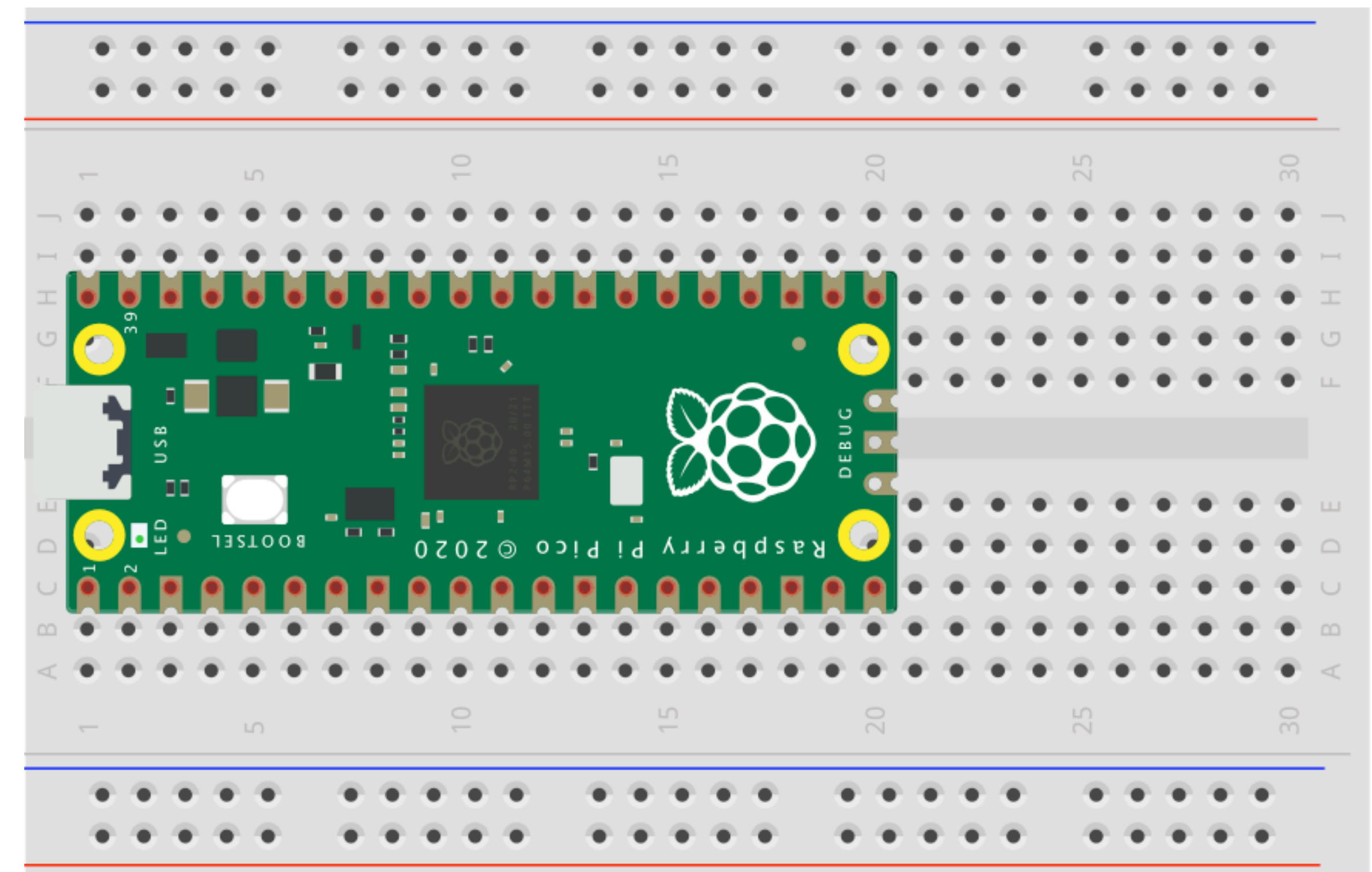
# Wiederholung (Endlos-Schleife)
while True:
    led.duty_u16(i)
    sleep(0.1)
    i = i + 3000
    if i > 65535:
        i = 0
```

Lichtsteuerung für Lichteffekte mit LEDs (1)

Lichteffekte mit Leuchtdioden leben von unterschiedlichen Abstufungen der Helligkeit über einen zeitlichen Verlauf. Eine Lichtsteuerung erzeugt Übergänge, was man als Fading bezeichnet. Über die Steuerung mehrerer LEDs kann zusätzlich eine Bewegung simuliert werden.

Wir wollen hier eine Mikrocontroller-gesteuerte Lichtsteuerung für eine Leuchtdiode realisieren. Genauer gesagt wollen wir unterschiedliche Helligkeiten einer LED mit einer Lichtsequenz steuern und damit einen bestimmten Lichteffekt erzeugen.

Wenn Du hinter das Geheimnis flackernder Lampen oder Kerzen kommen willst, dann bist Du hier genau richtig.



Lichtsteuerung für Lichteffekte mit LEDs (2)

```
# Bibliotheken laden
from machine import Pin, PWM
from time import sleep_ms

# Sequenz
seq = 'aazz'
wait = 100

# Initialisierung der Onboard-LED
led = PWM(Pin(25, Pin.OUT))
led.freq(1000)

# Funktion
def light(seq, wait):
    abc = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
    step = int(65535 / len(abc))
    # Wiederholung einleiten (Schleife)
    while True:
        i = 0
        for i in range(0, len(seq)):
            for j in range(0, len(abc)):
                if seq[i] == abc[j]:
                    break
            led.duty_u16(j * step)
            sleep_ms(wait)

light(seq, wait)
```

Im Programmcode wird die Helligkeit der LED über ein PWM-Signal gesteuert bzw. eingestellt. Wir müssen uns nur noch überlegen, wie wir die Helligkeitswerte in einer Sequenz im Programm darstellen.

Es gibt sicherlich auch andere Lösungen. Wir haben uns hier für Kleinbuchstaben entschieden. Die Werte reichen von „a“ bis „z“. „a“ bedeutet soviel wie „aus“ und „z“ bedeutet soviel wie „an“. Bei den Kleinbuchstaben dazwischen variiert die Helligkeit der LED zwischen „aus“ und „an“

Die Steuerung der Helligkeit über den zeitlichen Verlauf erfolgt durch die Kombination der Buchstaben.

- Eine Zeichenfolge von „a“ wäre ein Daueraus, „z“ ein Dauerlicht bei voller Helligkeit und „m“ ein Dauerlicht bei etwa halber Helligkeit.
- Eine Zeichenfolge von „mama“ oder nur „ma“ wäre ein Blinken.
- Eine Zeichenfolge von „mmamma“ wäre ein Blinken mit einer kurzen Pause.

Lichtsteuerung für Lichteffekte mit LEDs (3)

Es geht jetzt darum, dass Du einige Sequenzen ausprobierst. Hierzu ein paar Beispiele, die aber nicht perfekt sind. Das heißt, Du darfst sie gerne nach Deinem Empfinden verbessern.

- Dauerlicht: m
- Pulsierendes Licht 1: mmnmmommommnonmmonqnmno
- Pulsierendes Licht 2: ahmvzvmha
- Flackerndes Licht 1: nmonqnmomnmomomno
- Flackerndes Licht 2: mmmaammmaammammammamma
- Flackerndes Licht 3: mmamammammamamaamamma
- Kerze 1: mmmmmaaaaammmmaaaaaabcdefghijklmnop
- Kerze 2: mmmaaaabcdefghijklmnopmmmmaaaaammmaamm
- Kerze 3: mmmaaaammmaaaammmaabcdefghijklmnopmmmmaaaa
- Kerze 4: mnyznmjknmfghrsyz
- Schnelles Blinken: mamamamama
- Langsames Blinken: aaaaaaaazzzzzzzz
- Langsames Pulsieren 1: jklmnopqrstuvwxyzzyxwvutsrqponmlkj
- Langsames Pulsieren 2: abcdefghijklmnopqrrqponmlkjihgfedcba
- Langsames Pulsieren 3: abcdefghijklmnopqrstuvwxyzzyxwvutsrqponmlkjihgfedcba

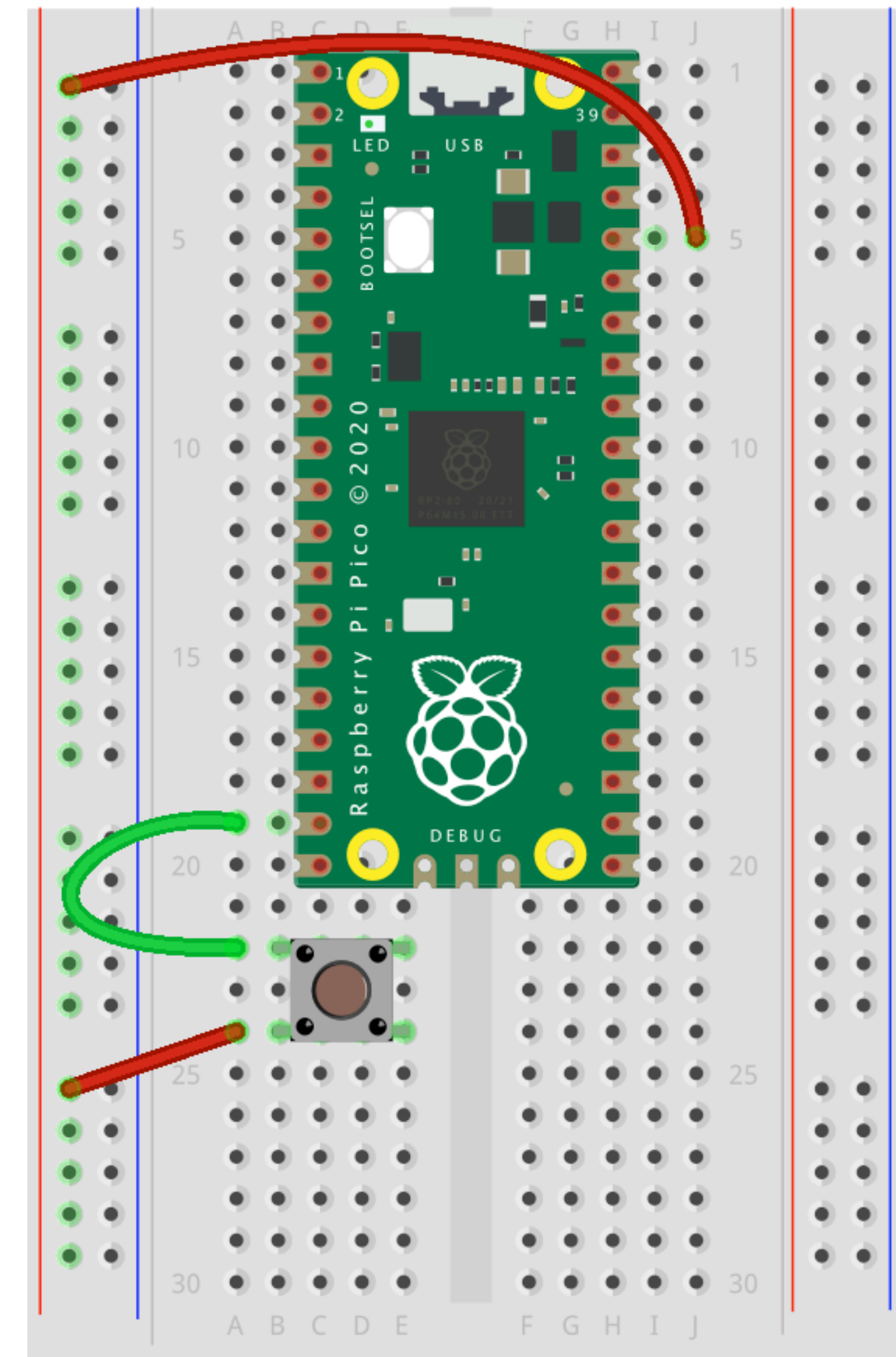
...

Taster-Zustand auswerten und mit einer LED anzeigen (1)

Ein Taster kann zwei Zustände haben: „gedrückt“ und „nicht gedrückt“. Entsprechend für „Ein“ und „Aus“. Das entspricht der binären Logik und ist eigentlich ganz einfach. Doch zu allem Überfluss gibt es gleich 4 Möglichkeiten, wie man einen Taster mit dem Raspberry Pi Pico verbindet und wie der GPIO-Eingang initialisiert werden muss.

Zum Anzeigen des Taster-Zustands verwenden wir die Onboard-LED des Picos.

Das folgende Programmbeispiele berücksichtigt völlig wertfrei nur eine Variante. Denkbar wäre es, dass in der Praxis eine andere Variante besser wäre.



fritzing

Der Taster wird auf dem Steckbrett auf der einen Seite mit +3,3V (VCC) und auf der anderen Seite mit dem GPIO verbunden.

Taster-Zustand auswerten und mit einer LED anzeigen (2)

Die Funktion des Programms ist denkbar einfach. Ein GPIO wird als Ausgang definiert. Ein GPIO wird als Eingang definiert, an dem der Taster angeschlossen ist. Wird der Taster gedrückt, dann leuchtet die Leuchtdiode auf dem Pico. Wird der Taster losgelassen, dann wird sie wieder ausgeschaltet.

```
# Bibliotheken laden
from machine import Pin

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT)

# Initialisierung von GPIO14 als Eingang
btn = Pin(14, Pin.IN, Pin.PULL_DOWN)

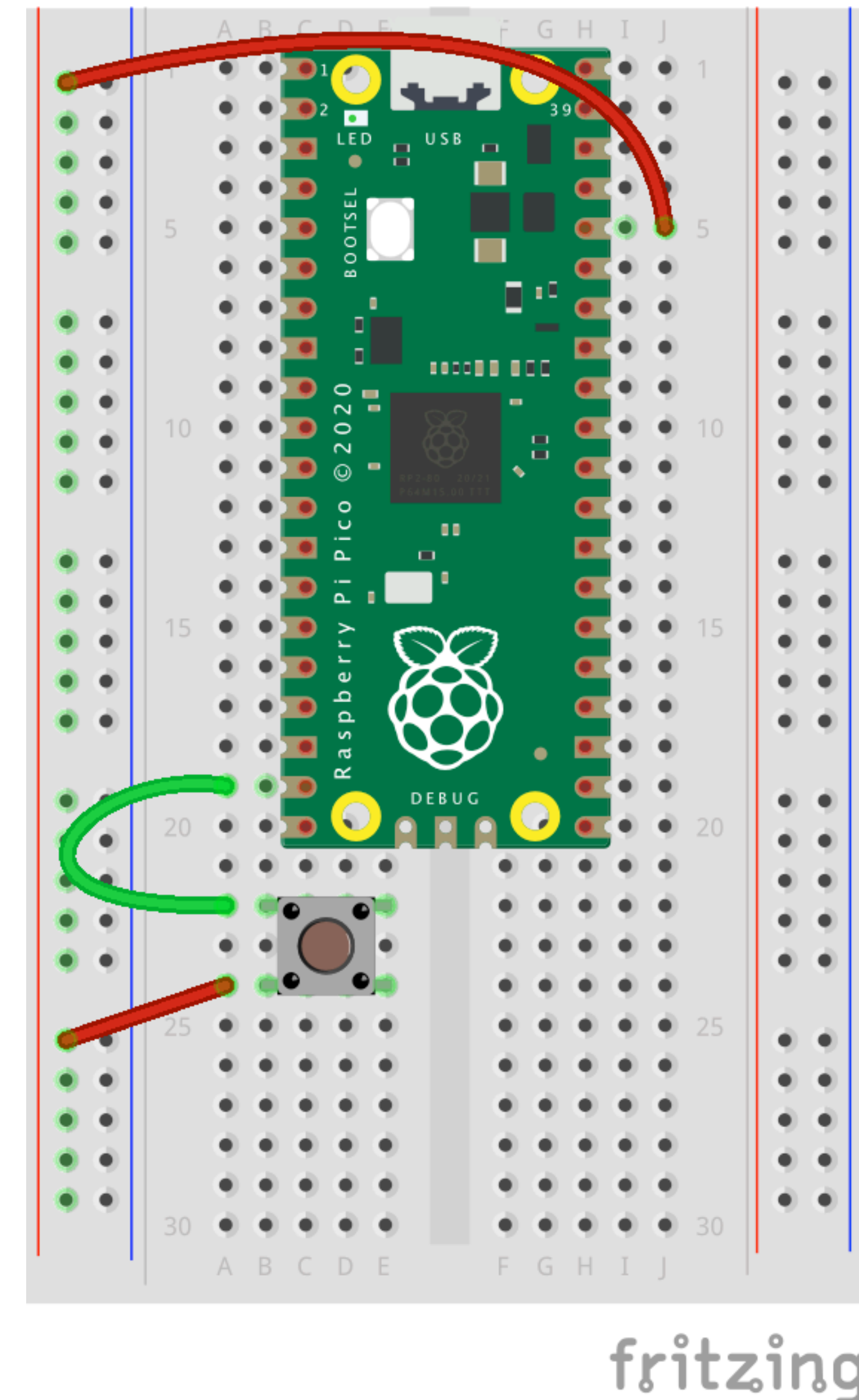
# Funktion zur Taster-Auswertung
while True:
    if btn.value() == 1:
        led_onboard.on()
    else:
        led_onboard.off()
```

Taster-Zustand invertieren (Inverter/NICHT-Funktion) (1)

Die NICHT-Funktion ist eine logische Funktion bzw. Operator. Man kennt diese Funktion auch unter anderen Begriffen. Zum Beispiel als Negation oder in der Elektronik als Inverter. Im Prinzip wird hier ein Zustand oder Wert umgekehrt.

In der Digitaltechnik und Computertechnik wird durch die NICHT-Funktion eine logische 1 zu einer 0 und eine logische 0 zu einer 1.

In diesem Aufbau werden das Drücken des Tasters und das Leuchten der Leuchtdiode durch die „0“ repräsentiert. Der ungedrückte Taster (Grundzustand) und die nicht leuchtende Leuchtdiode werden durch die „1“ repräsentiert. Demnach leuchtet die Leuchtdiode, wenn der Taster nicht gedrückt ist. Umgekehrt geht die Leuchtdiode aus, wenn der Taster gedrückt wird.



Der Taster wird auf dem Steckbrett auf der einen Seite mit +3,3V (VCC) und auf der anderen Seite mit dem GPIO verbunden.

Taster-Zustand invertieren (Inverter/NICHT-Funktion) (2)

Der Programmablauf ist vergleichsweise einfach. Im Programm werden die LED und der Taster initialisiert. Anschließend wird in einer Endlosschleife überprüft, ob der Taster gedrückt wurde. Wenn ja, dann wird die LED ausgeschaltet, wenn nicht, dann wird sie eingeschaltet.

Auf dem ersten Blick scheint das hier der gleiche Programmcode zu sein, wie im Programmcode davor von „Taster-Zustand auswerten und mit einer LED anzeigen“. Der Unterschied ist die „0“ in der „if“-Verzweigung innerhalb der „while“-Schleife.

```
# Bibliotheken laden
from machine import Pin

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT)

# Initialisierung von GPIO14 als Eingang
btn = Pin(14, Pin.IN, Pin.PULL_DOWN)

# Funktion zur Taster-Auswertung
while True:
    if btn.value() == 0:
        led_onboard.on()
    else:
        led_onboard.off()
```

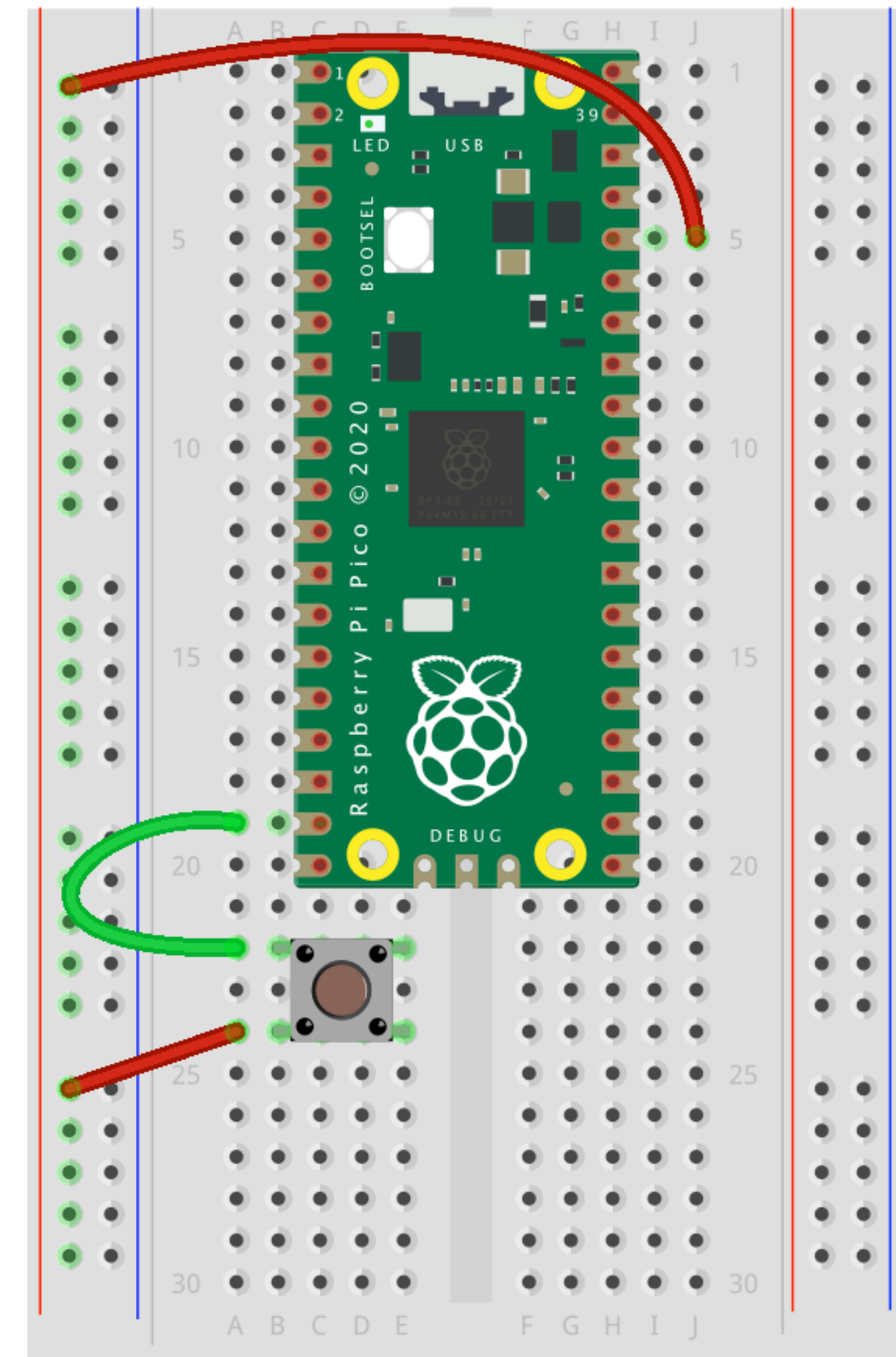
LED mit Taster einschalten und ausschalten (1)

Damit eine Leuchtdiode (LED) beim Drücken eines Tasters an- und ausgeht braucht man eigentlich keine Softwaresteuerung. Diese Funktion kann man natürlich auch ohne Software realisieren. Einfach nur dadurch, dass der Taster mit der Leuchtdiode und einem Vorwiderstand in Reihe geschaltet wird.

Allerdings kann der Taster dann auch nur diese eine festgelegte Funktion. Wenn die Funktion fest verdrahtet oder gelötet ist, dann ist eine Änderung nicht so einfach möglich. Soll der Taster eine andere Funktion haben, dann muss man die Hardware ändern.

Es gibt also gute Gründe, warum heute alles mit Software realisiert wird und prozessorgesteuert ist. Eine Funktion lässt sich in der Software einfach leichter ändern, als in der Hardware.

Das folgende Programm hat eine Haltefunktion. Das heißt, drücken wir den Taster, dann wird die LED eingeschaltet. Drücken wir ein weiteres Mal wird die LED ausgeschaltet. Man spricht hier auch von einer manuellen Toggle-Funktion.



fritzing

Der Taster wird auf dem Steckbrett auf der einen Seite mit +3,3V (VCC) und auf der anderen Seite mit dem GPIO verbunden.

LED mit Taster einschalten und ausschalten (2)

Das folgende Programm schaltet die LED immer dann ein und wieder aus, wenn der Taster gedrückt wird. Das Programm wird nicht automatisch beendet.

Das besondere an diesem Programm ist, dass der Taster-Druck einen Interrupt auslöst und dadurch eine bestimmte Funktion im Programm ausführt.

Na, funktioniert es? Naja, irgendwo schon. Aber irgendwie auch nicht immer. Manchmal geht die LED an und gleich wieder aus. Die Funktion ist also nicht so ganz zuverlässig.

Doch woran liegt das? Das Problem sind die Taster. Die neigen zum Prellen. Gemeint ist, dass ein Taster bei Betätigung nicht zwangsläufig einen dauerhaften Kontakt herstellt, sondern durch den mechanischen Aufschlag der Kontakte zurückfedern kann. Das heißt, das Drücken eines Tasters kann zu einer mehrmaligen Kontakt- und damit Funktionsauslösung führen. Das sieht dann so aus, dass die LED angeht, aber auch sofort wieder ausgeht. Der eine Druck auf den Taster hat den internen Kontakt zwei oder mehrmals betätigt und den Interrupt zwei oder mehr ausgelöst.

```
# Bibliotheken laden
from machine import Pin

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT, value=0)

# Initialisierung von GPIO14 als Eingang
btn = Pin(14, Pin.IN, Pin.PULL_DOWN)

# Taster-Funktion
def button(pin):
    led_onboard.toggle()

# Taster-Auslösung
btn.irq(trigger=Pin.IRQ_RISING, handler=button)
```

LED mit Taster einschalten und ausschalten (3)

Das folgende Programm schaltet die LED immer dann ein und wieder aus, wenn der Taster gedrückt wird. Das Programm wird nicht automatisch beendet.

Das besondere an diesem Programm ist, dass der Taster-Druck einen Interrupt auslöst und dadurch eine bestimmte Funktion im Programm ausführt. Diese Funktion setzt erst einen Timer mit einer Verzögerung von 200 ms und führt dann die Funktion einmalig aus, die die LED ansteuert.

Diese Konstruktion im Programmcode führt dazu, dass wenn der Taster innerhalb der 200 ms prellt, also zwischen High und Low hin- und herschwingt, die Funktion nicht erneut ausgeführt wird.

Hinweis: Die hier dargestellt Lösung ist vergleichsweise einfach und nicht in jedem Fall die beste Lösung.

```
# Bibliotheken laden
from machine import Pin, Timer

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT, value=0)

# Initialisierung von GPIO14 als Eingang
btn = Pin(14, Pin.IN, Pin.PULL_DOWN)

# Timer erstellen
timer = Timer()

# Taster-Funktion
def on_pressed(timer):
    led_onboard.toggle()
    print('pressed')

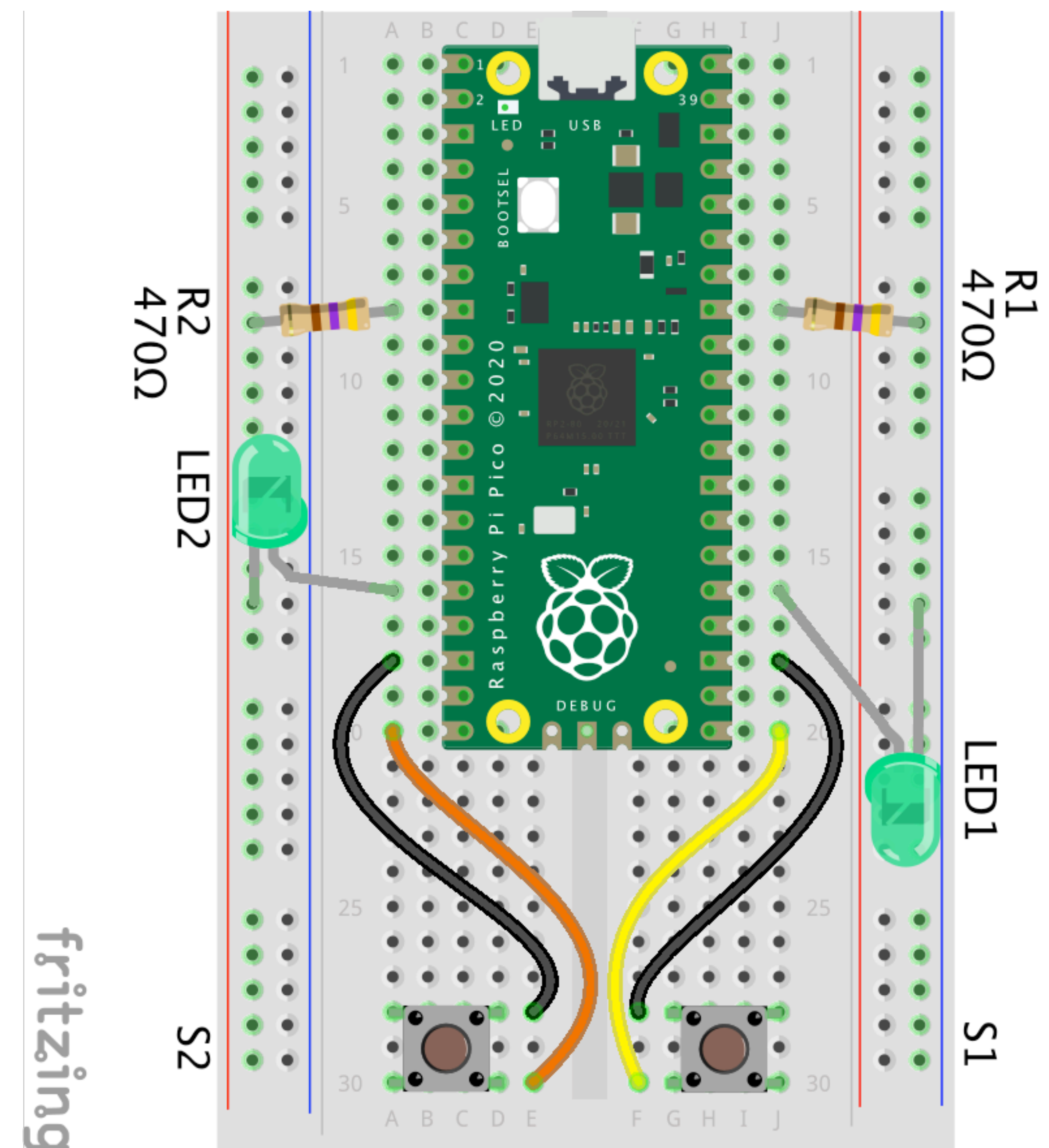
# Entprell-Funktion
def debounce(pin):
    # Timer setzen (200 ms)
    timer.init(mode=Timer.ONE_SHOT, period=200,
               callback=on_pressed)

# Taster-Auslösung
btn.irq(handler=debounce, trigger=Pin.IRQ_RISING)
```

Reaktionsspiel mit zwei Tastern und LEDs (1)

Experimentieren mit Tastern und LEDs ist ja ganz nett. Aber soll es das schon gewesen sein? Taster drücken und LED geht an. Voll langweilig. Es wird Zeit das wir etwas wirklich sinnvolles mit Tastern und LEDs machen. Ein Spiel gefällig? Du musst natürlich zu Zweit sein, sonst macht das keinen richtigen Spaß.

Na dann los. Aufgebaut und hergeschaut. Wer hat am schnellsten den Taster gedrückt, wenn die Onboard-LED angeht?



S1, Taster

S2, Taster

R1, Widerstand, 470 Ohm

R2, Widerstand, 470 Ohm

LED1, Leuchtdiode, rot, gelb oder grün

LED2, Leuchtdiode, gelb, gelb oder grün

Reaktionsspiel mit zwei Tastern und LEDs (2)

Im folgenden Programm werden zwei Taster und drei Leuchtdioden definiert. Die Onboard-LED soll nach einer zufälligen Zeit zwischen 5 und 10 Sekunden angehen. Wer zuerst seinen Taster drückt, dessen Leuchtdiode leuchtet und der Spieler hat gewonnen. Wessen LED leuchtet, kann das Spiel mit Drücken seines Tasters erneut starten.

```
# Bibliotheken laden
from machine import Pin
from time import sleep
from random import randint

# Initialisierung
led_onboard = Pin('LED', Pin.OUT, value=0)
led1 = Pin(19, Pin.OUT, value=0)
led2 = Pin(12, Pin.OUT, value=0)
btn1 = Pin(16, Pin.IN, Pin.PULL_UP)
btn2 = Pin(15, Pin.IN, Pin.PULL_UP)

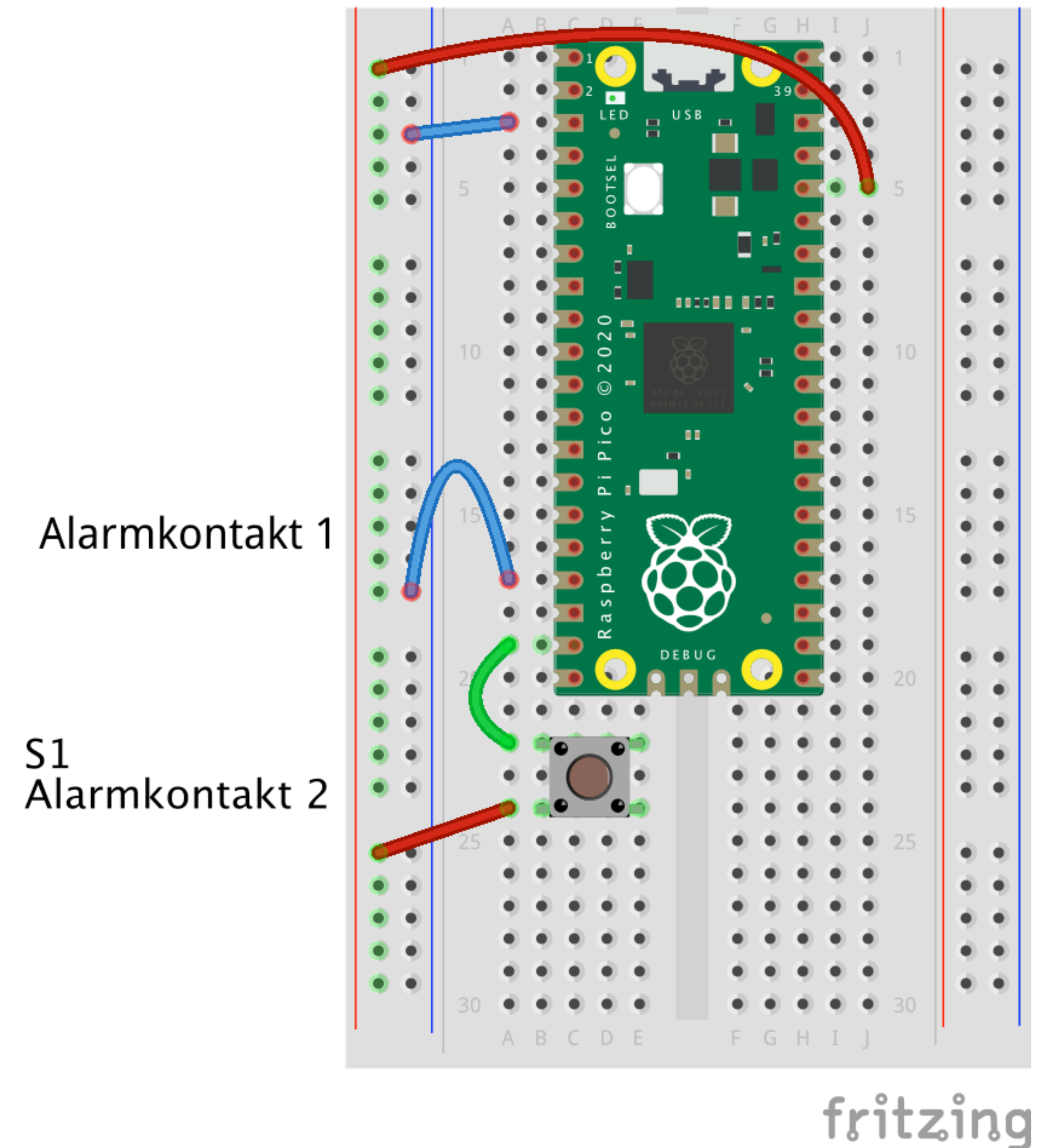
# Wiederholung
while True:
    # LEDs ausschalten
    led1.off()
    led2.off()
    # Blinken: gleich geht das Spiel los
    for i in range(1, 10):
        led_onboard.toggle()
        sleep(0.2)
    led_onboard.off()
    # Zufällige Zeit ermitteln und warten
    rand = randint(5, 10)
    sleep(rand)
    # Onboard-LED einschalten
    led_onboard.on()
    # Wiederholung: Wer drückt seinen Taster zuerst?
    while True:
        # Taster 1 wird zuerst gedrückt
        if btn1.value() == 0:
            led1.on()
            sleep(2)
            while True:
                if btn1.value() == 0: break
                sleep(0.1)
        # Taster 2 wird zuerst gedrückt
        if btn2.value() == 0:
            led2.on()
            sleep(2)
            while True:
                if btn2.value() == 0: break
                sleep(0.1)
    # Schleife beenden, wenn eine LED an ist
    if led1.value() == 1 or led2.value() == 1: break
    sleep(0.1)
```


Alarmkontakt für Alarmanlage (1)

Eine Alarmanlage ist eine technische Einrichtung, die einen Alarm meldet, wenn ein Sensor oder Kontakt ausgelöst wird. Bei Kontakten gibt es das Prinzip des Öffnens und des Schließens. Beim Öffnen wird der Alarm beim Öffnen des Kontakts ausgelöst. Beim Schließen wird der Alarm beim Schließen des Kontakts ausgelöst.

Wir wollen hier beide Prinzipien ausprobieren und miteinander vergleichen.

- Der erste Alarmkontakt folgt dem Prinzip des Öffnens und wird durch ein Verbindungskabel repräsentiert, das den GPIO mit Ground (GND) verbindet. Durch Herausziehen des Verbindungskabels wird der Alarm ausgelöst.
- Der zweite Alarmkontakt folgt dem Prinzip des Schließens und wird durch einen Taster repräsentiert, der den GPIO mit +3,3V (VCC) verbindet. Wird der Taster betätigt, wird der Alarm ausgelöst.



Alarmkontakt für Alarmanlage (2)

Der Programmablauf ist vergleichsweise einfach. Im Programm werden die Alarmkontakte initialisiert.

Anschließend wird in einer Endlosschleife überprüft, ob die Alarmkontakte ausgelöst wurden. Wenn ja, dann wird die LED eingeschaltet.

Wenn ein Alarm ausgelöst wurde, dann leuchtet die Leuchtdiode. Um die Alarmanlage zurückzusetzen, müssen die Alarmkontakte in Ruhe gebracht werden (Grundzustand). Das heißt, das Verbindungskabel vom Alarmkontakt 1 muss gesteckt sein und der Taster vom Alarmkontakt 2 darf nicht betätigt sein. Wenn beides der Fall ist, dann kann das Programm erneut gestartet werden. Die Leuchtdiode geht aus, wenn alle Kontakte in Ruhe sind.

```
# Bibliotheken laden
from machine import Pin

# Initialisierung der Onboard-LED
led_onboard = Pin('LED', Pin.OUT, value=0)

# Initialisierung von GPIO13 als Eingang
# (Alarmkontakt 1)
kontakt1 = Pin(13, Pin.IN, Pin.PULL_UP)

# Initialisierung von GPIO14 als Eingang
# (Alarmkontakt 2)
kontakt2 = Pin(14, Pin.IN, Pin.PULL_DOWN)

# Funktion zur Alarmkontakt-Auswertung
while True:
    if kontakt1.value() == 1:
        led_onboard.on()
    elif kontakt2.value() == 1:
        led_onboard.on()
```

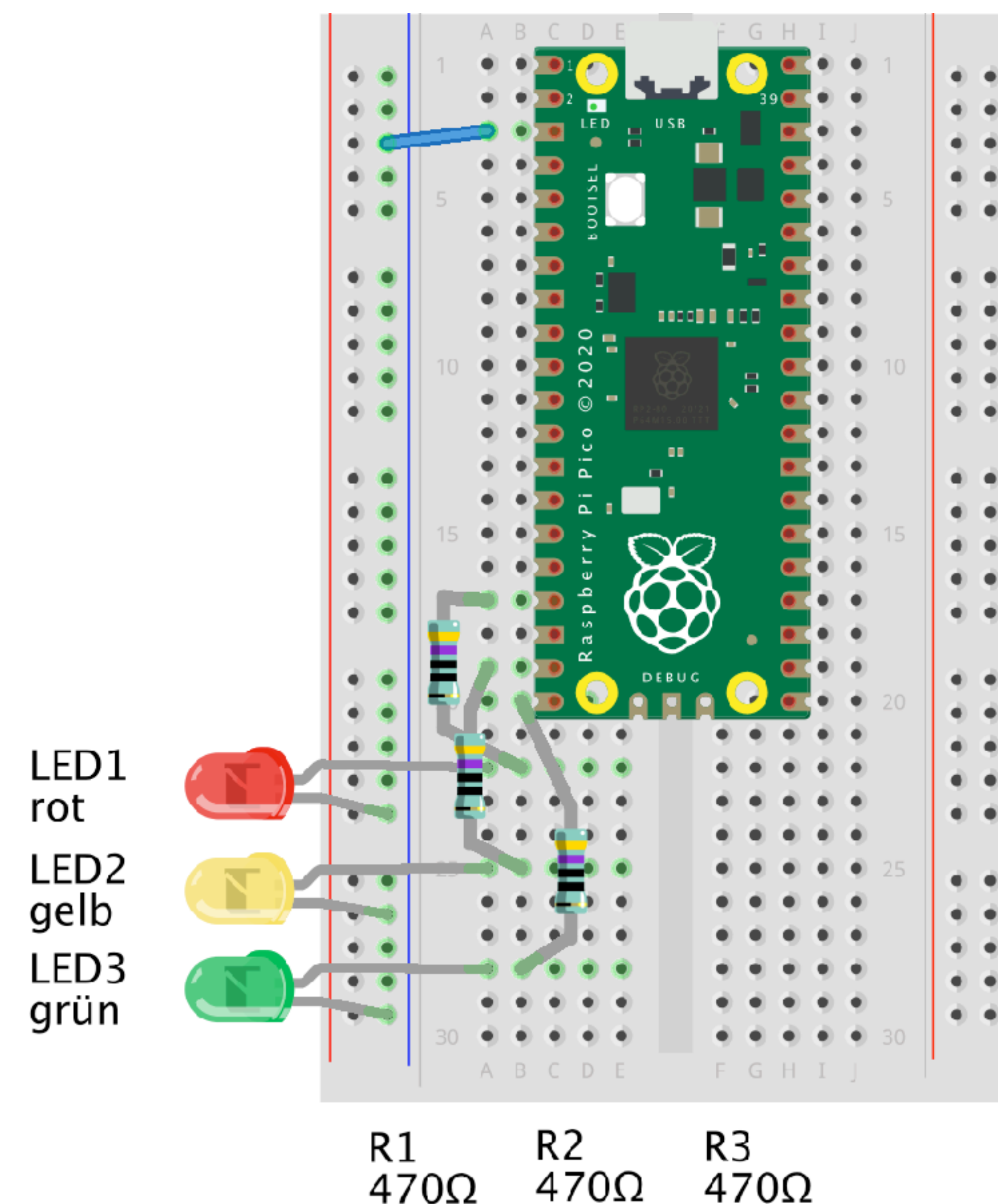
LED-Ampel-Steuerung (1)

Bei einer Ampel-Steuerung geht es darum, dass die Lampen bzw. LEDs in einer bestimmten zeitlichen Abfolge leuchten und auch wieder ausgehen müssen. In bestimmten Ampelphasen leuchten bestimmte Leuchtdioden. Manche Phasen sind nur übergangsweise aktiv, während andere Phasen länger dauern.

Für eine Ampel-Steuerung müssen wir 4 Ampelphasen programmieren:

- ROT
- GELB
- GRÜN
- ROT + GELB

Der folgende Aufbau sieht eine Verkehrsampel mit drei Leuchtdioden vor und einem Programmcode mit der Ampel-Steuerung.



fritzing

- R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R2: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R3: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- LED1: Leuchtdiode, rot
- LED2: Leuchtdiode, gelb
- LED3: Leuchtdiode, grün

LED-Ampel-Steuerung (2)

Im Programmcode werden die Leuchtdioden initialisiert.
In der anschließenden Schleife werden insgesamt 4
Ampelphase nacheinander durchlaufen und wiederholt.

```
# Bibliotheken laden
from machine import Pin
from time import sleep

# Initialisierung von GPIO 13, 14 und 15
red = Pin(13, Pin.OUT, value=0)
yellow = Pin(14, Pin.OUT, value=0)
green = Pin(15, Pin.OUT, value=0)

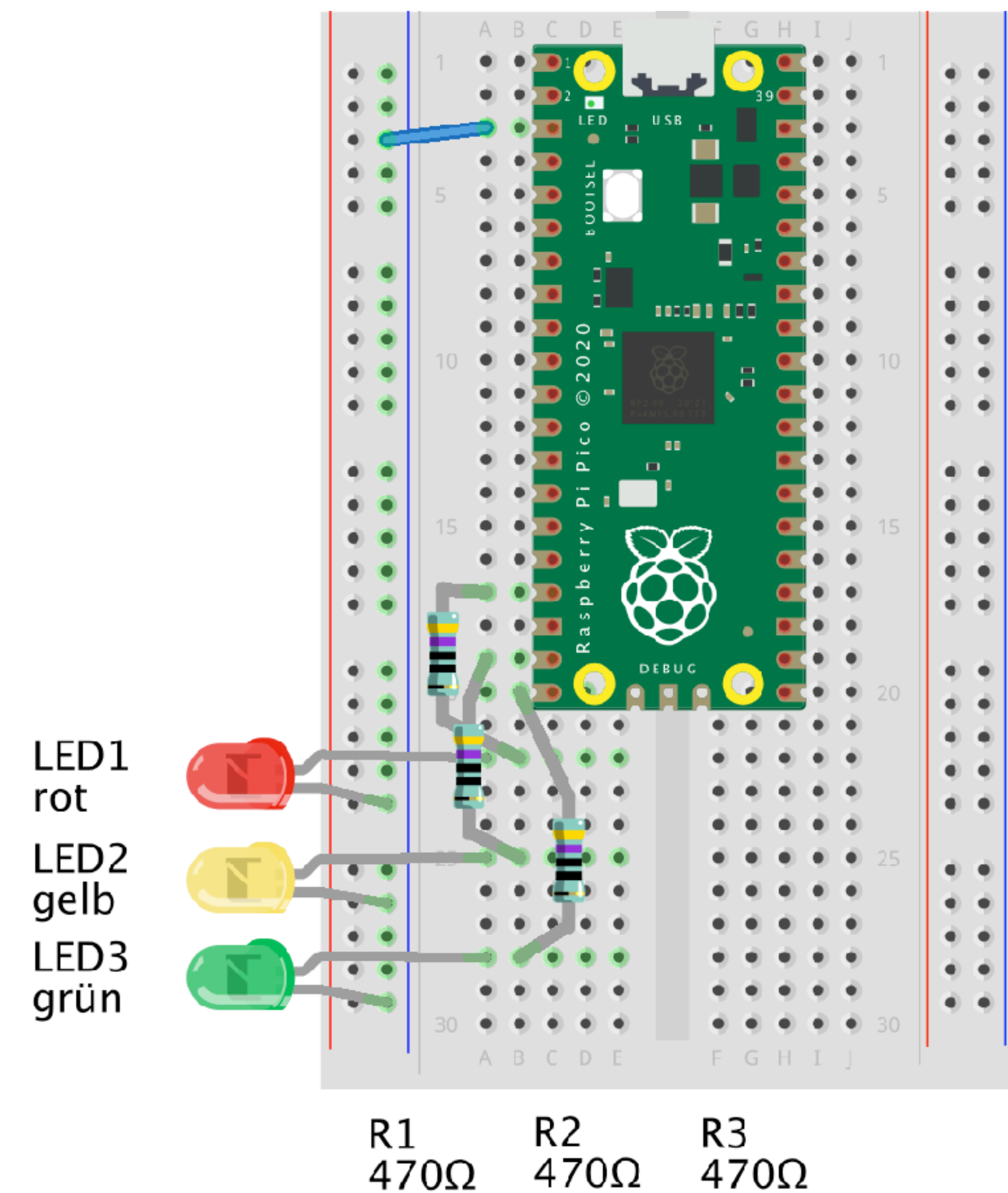
# Wiederholung (Endlos-Schleife)
while True:
    # Ampelphase: ROT
    red.value(1)
    yellow.value(0)
    sleep(5)
    # Ampelphase: ROT + GELB
    yellow.value(1)
    sleep(2)
    # Ampelphase: GRÜN
    red.value(0)
    yellow.value(0)
    green.value(1)
    sleep(5)
    # Ampelphase: GELB
    green.value(0)
    yellow.value(1)
    sleep(1)
```

Binärer Zähler mit LEDs (1)

Binär bedeutet, dass etwas zwei Zustände annehmen kann. Zwei Zustände lassen sich gut mit Leuchtdioden darstellen. Der eine Zustand ist „leuchtend“, der andere Zustand „nicht leuchtend“. Ein binärer Zähler würde nach dem dualen Zahlensystem zählen und lässt sich deshalb mit einer Reihe von Leuchtdioden sehr leicht realisieren.

Das interessante an einem binären Zähler ist, dass seine Stellenwerte nur 0 oder 1 sein können und sich diese Werte mit Leuchtdioden darstellen lassen. Wenn ein Stellenwert 0 ist, dann leuchtet die Leuchtdiode nicht und wenn ein Stellenwert 1 ist, dann leuchtet die Leuchtdiode.

Wir bauen uns hier einen 3-Bit-Zähler mit drei Leuchtdioden. Die Zählweise und Darstellung der dualen Zahlen sieht wie folgt aus.



fritzing

- R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R2: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R3: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- LED1: Leuchtdiode, rot
- LED2: Leuchtdiode, gelb
- LED3: Leuchtdiode, grün

Binärer Zähler mit LEDs (2)

Der binäre Zähler beginnt bei 0 und zählt mit 3 Leuchtdioden bis 7. Danach beginnt er wieder von vorne zu zählen.

Zahl	LED1	LED2	LED3
0	AUS	AUS	AUS
1	AUS	AUS	EIN
2	AUS	EIN	AUS
3	AUS	EIN	EIN
4	EIN	AUS	AUS
5	EIN	AUS	EIN
6	EIN	EIN	AUS
7	EIN	EIN	EIN

```
from machine import Pin
from time import sleep

# GPIOs für den Zähler auswählen
GPIO = [13, 14, 15]
leds = len(GPIO)
led = [0] * leds

# Initialisierung der GPIOs als Ausgang
for i in range(leds):
    led[i] = Pin(GPIO[i], Pin.OUT, value=0)

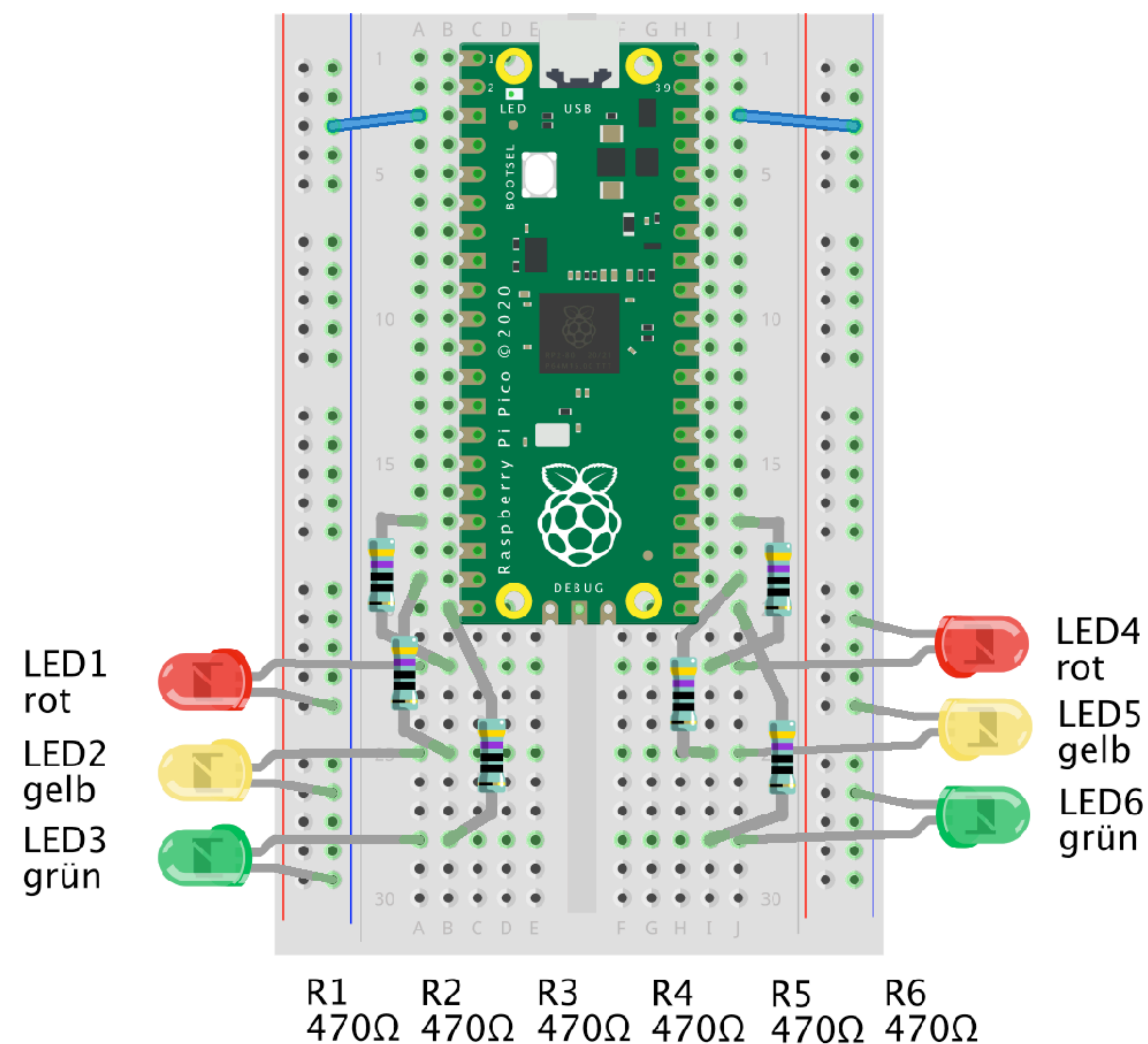
# Zähler-Ausgabe
def output_counter(x):
    global leds
    b = bin(x)
    b = b.replace("0b", "")
    diff = leds - len(b)
    for i in range(0, diff):
        b = "0" + b
    for i in range(0, leds):
        if b[i] == "1":
            led[i].value(1)
        else:
            led[i].value(0)
    return

# Zählen
count = 0
max_count = 2 ** leds
while True:
    output_counter(count)
    sleep(0.75)
    count = count + 1
    if count == max_count:
        count = 0
```

LED-Lauflicht (1)

In einem LED-Lauflicht sind die Leuchtdioden wie in einer Lichterkette nebeneinander angeordnet. Allerdings werden in einem Lauflicht alle Leuchtdioden einzeln angesteuert.

So ist es möglich, dass alle Lampen bzw. Leuchtdioden in einer beliebigen Reihenfolge leuchten. In einem Lauflicht werden die Leuchtdioden nacheinander ein- und ausgeschaltet. Damit die Leuchtdioden an- und ausgehen müssen wir das Lauflicht programmieren.



fritzing

R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
R2: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
R3: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
R4: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
R5: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
R6: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)

LED1: Leuchtdiode, rot, gelb oder grün
LED2: Leuchtdiode, rot, gelb oder grün
LED3: Leuchtdiode, rot, gelb oder grün
LED4: Leuchtdiode, rot, gelb oder grün
LED5: Leuchtdiode, rot, gelb oder grün
LED6: Leuchtdiode, rot, gelb oder grün

LED-Lauflicht (2)

Ein Programmieranfänger wird normalerweise jede LED einzeln initialisieren separat ein- und ausschalten.

Ein guter Programmierer wird jedoch versuchen nicht jedes gleiche Element, in dem Fall die LED, einzeln zu programmieren, sondern alle Elemente nacheinander in einer Schleife anzusprechen.

Dadurch wird der Programmcode auf den ersten Blick komplizierter, aber auch wesentlich flexibler.

Der Programmcode lädt zum Experimentieren ein:

- Versuche einfach mal das Lauflicht zu verkürzen, also LEDs softwareseitig zu entfernen und nach einem erfolgreichen Test hinzuzufügen.
- Lege eine eigene Reihenfolge fest. Lass das Lauflicht zum Beispiel anders herum laufen.

```
# Bibliotheken laden
from machine import Pin
from time import sleep

# GPIOs für das Lauflicht auswählen
GPIO = [18, 17, 16, 13, 14, 15]
leds = len(GPIO)
led = [0] * leds

# Initialisierung von GPIOs als Lauflicht
for i in range(leds):
    led[i] = Pin(GPIO[i], Pin.OUT, value=0)

# Wiederholung (Endlos-Schleife)
while True:
    for i in range(leds):
        led[i].on()
        sleep(0.5)
        led[i].off()
```

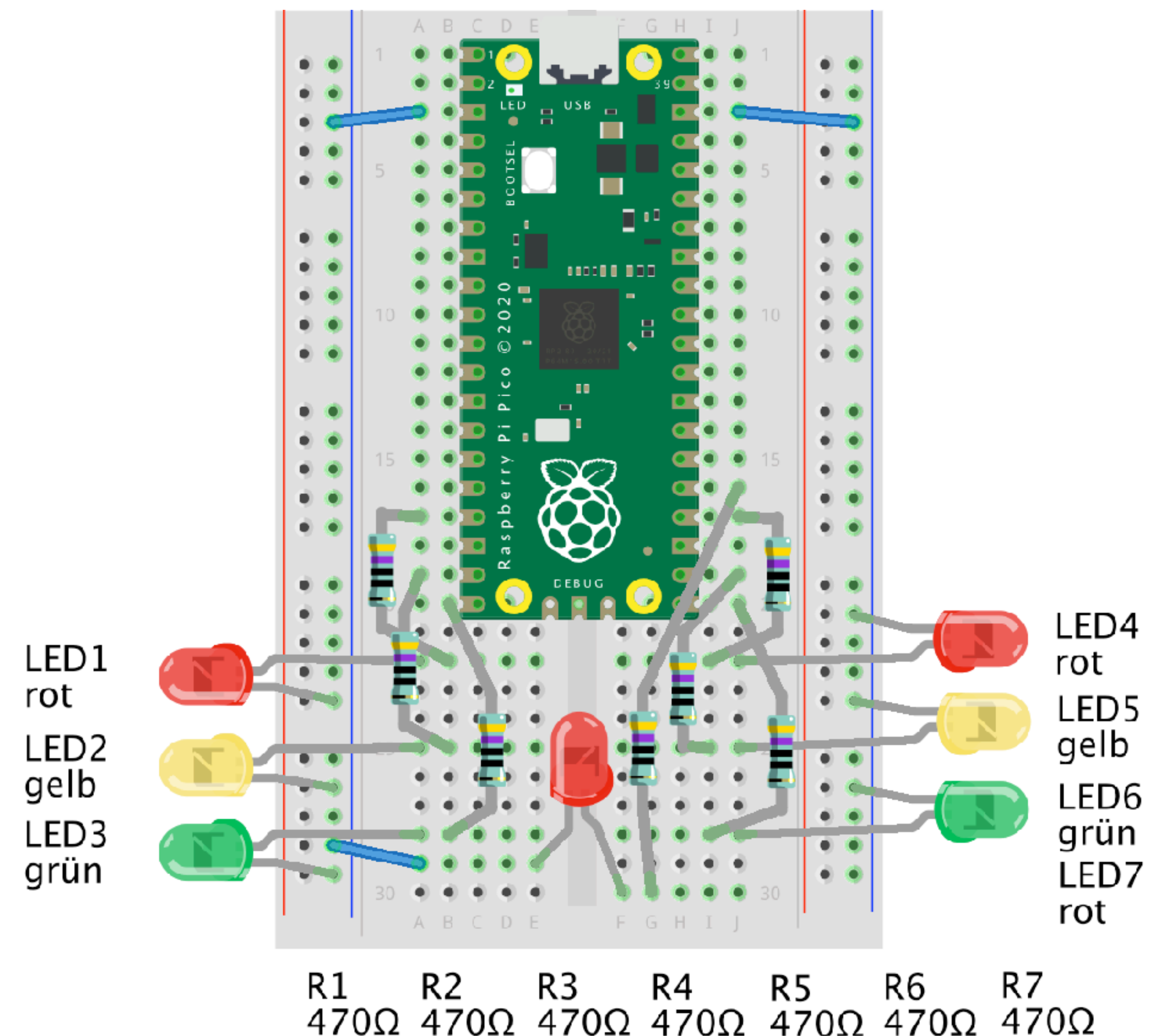

Zufallszahl generieren (1)

Normalerweise erwarten wir von einem Computer, dass die selbe Eingabe auch immer die selbe Ausgabe produziert. Das etwas zufällig in einem Computer passiert ist eigentlich nicht gewünscht. Deshalb ist er so konstruiert, dass sich alle Zustände in einem Computer vorhersagen oder berechnen lassen. So ein Computer ist also eine richtige Spaßbremse.

Was machen wir, wenn doch mal etwas zufälliges erzeugt werden soll? Dafür gibt es spezielle Funktionen, die so etwas wie Zufall erzeugen können.

Im folgenden Aufbau wollen wir eine von insgesamt 7 Leuchtdioden zufällig leuchten lassen.

- R1: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R2: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R3: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R4: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R5: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R6: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)
- R7: Widerstand, 470 Ohm (Gelb-Violett-Schwarz-Schwarz)



fritzing

- LED1: Leuchtdiode, rot, gelb oder grün
- LED2: Leuchtdiode, rot, gelb oder grün
- LED3: Leuchtdiode, rot, gelb oder grün
- LED4: Leuchtdiode, rot, gelb oder grün
- LED5: Leuchtdiode, rot, gelb oder grün
- LED6: Leuchtdiode, rot, gelb oder grün
- LED7: Leuchtdiode, rot, gelb oder grün

Zufallszahl generieren (2)

Im Programmcode werden 7 GPIOs initialisiert, die mit Leuchtdioden beschaltet sind. Anschließend wird eine zufällige Zahl zwischen 1 und 7 erzeugt und die entsprechende Leuchtdiode zum Leuchten gebracht.

Wenn das Programm gestartet wurde und eine Leuchtdiode leuchtet, wird das Programm beendet.

Wenn Du eine neue Zufallszahl haben möchtest, dann musst Du das Programm erneut starten.

Der Programmcode lädt zum Experimentieren ein:

- Eine sinnvolle Erweiterung wäre, wenn das Drücken eines Tasters das Programm erneut ausführt und eine andere Leuchtdiode zufällig leuchten würde.
- Manipuliere das Array, in dem die GPIOs definiert sind so, dass eine der LEDs nicht mehr leuchtet.

```
# Bibliotheken laden
from machine import Pin
from random import randint

# GPIOs für die Anzeige des Zufalls auswählen
GPIO = [19, 18, 17, 16, 13, 14, 15]
leds = len(GPIO)
led = [0] * leds

# Initialisierung der GPIOs als Ausgang
for i in range(leds):
    led[i] = Pin(GPIO[i], Pin.OUT, value=0)

# Zufall erzeugen
rand = randint(0, leds-1)

# LED aktivieren
led[rand].on()
```


Elektronischer Würfel (2)

Im Programmcode werden 7 GPIOs initialisiert, die mit Leuchtdioden beschaltet sind. Anschließend wird eine zufällige Zahl zwischen 1 und 6 erzeugt und die Leuchtdioden zum Leuchten gebracht, die der Würfelseite entsprechen.

Wenn das Programm gestartet wurde und die Leuchtdioden leuchten, wird das Programm beendet. Wenn Du eine erneut würfeln möchtest, dann musst Du das Programm erneut starten.

Der Programmcode lädt zum Experimentieren ein:

- Eine sinnvolle Erweiterung wäre, wenn das Drücken eines Tasters das Programm erneut ausführt und ein anderes Würfel-Auge zufällig leuchten würde.

```
# Bibliotheken laden
from machine import Pin
from random import randint

# GPIOs für den Zähler auswählen
GPIO = [19, 18, 17, 16, 13, 14, 15]
leds = len(GPIO)
led = [0] * leds

# Würfel-Augen definieren
cube = ("0000000",
        "1000000",
        "0100001",
        "1100001",
        "0101101",
        "1101101",
        "0111111")

# Initialisierung von GPIOs
for i in range(leds):
    led[i] = Pin(GPIO[i], Pin.OUT)

# Zufall erzeugen
rand = randint(1, leds-1)
eye = cube[rand]

# LEDs aktivieren
for i in range (leds):
    if eye[i] == "1":
        led[i].on()
    else:
        led[i].off()
```


Temperatur mit dem integrierten Temperatursensor messen (2)

Nach dem Start des Programms wird die Temperatur ermittelt. Die dazugehörigen Werte, „Dezimalzahl“, „Spannung“ und „Temperatur“ werden angezeigt.

Wichtig zu verstehen ist, dass alle drei Werte das gleiche bedeuten. Sie unterscheiden sich nur in der Darstellung.

Wie lässt sich die Temperatur erhöhen und senken?

- Eine höhere Temperatur erreichst Du dadurch, dass Du den Pico anhauchst. Die Luft aus unserem Atem hat in der Regel eine höhere Temperatur als die Umgebung.
- Um die Temperatur zu senken, kannst Du einfach warten und zuschauen, wie Temperatur fällt. Oder Du pustest die warme Luft am Pico weg.

```
# Bibliotheken laden
from machine import ADC
from time import sleep

# Initialisierung des ADC4
sensor_temp = ADC(4)
conversion_factor = 3.3 / (65535)

# Wiederholung einleiten (Schleife)
while True:
    # Temperatur-Sensor als Dezimalzahl lesen
    read = sensor_temp.read_u16()
    # Dezimalzahl in eine reelle Zahl umrechnen
    spannung = read * conversion_factor
    # Spannung in Temperatur umrechnen
    temperatur = 27 - (spannung - 0.706) / 0.001721
    # Ausgabe in der Kommandozeile/Shell
    print("Dezimalzahl: ", read)
    print("Spannung (V): ", spannung)
    print("Temperatur (°C): ", temperatur)
    print()
    # 2 Sekunden warten
    sleep(3)
```

EIN/AUS-Temperaturregelung (Zweipunktregelung) (1)

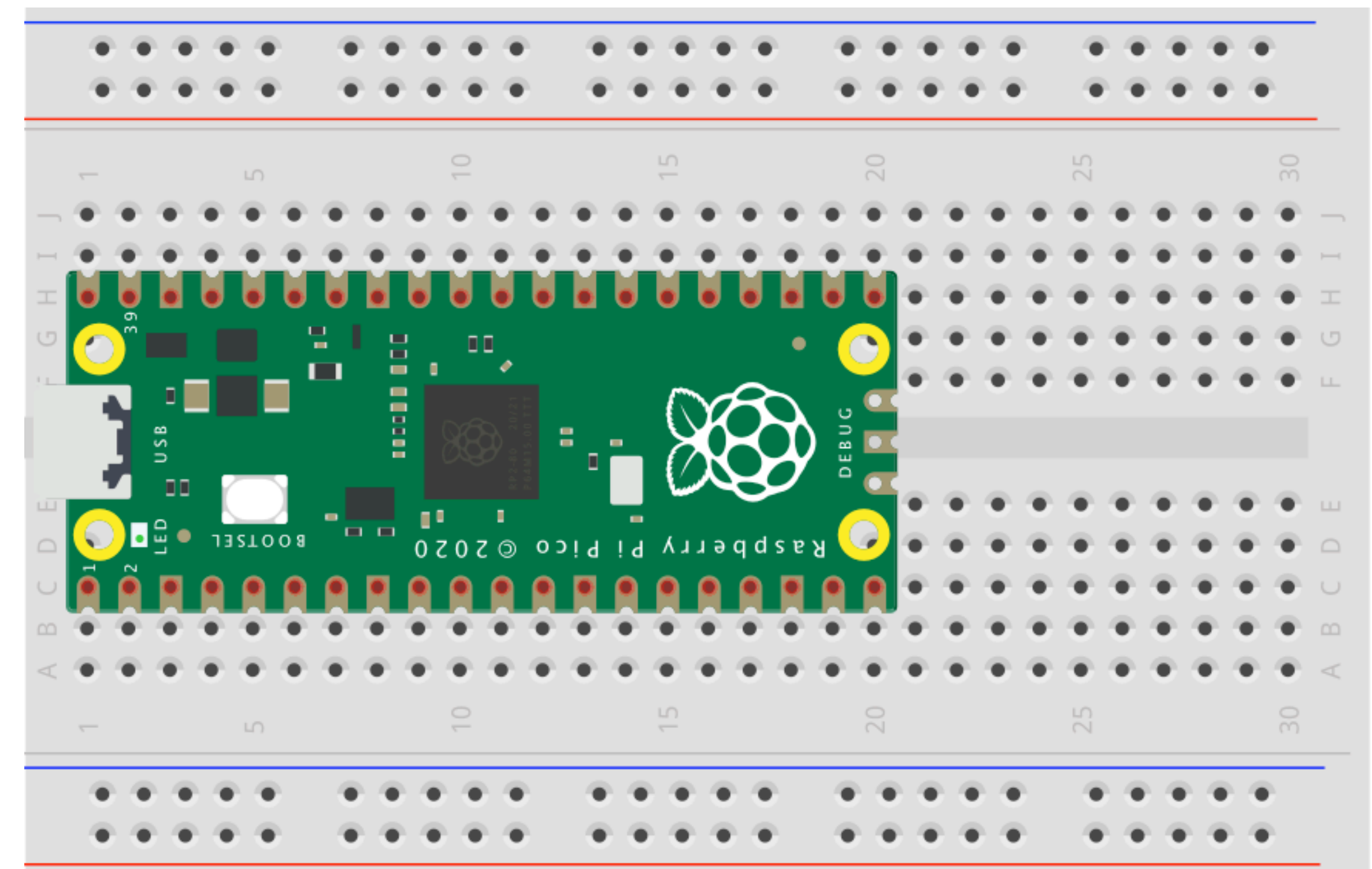
Das einfachste Regelsystem ist ein Zweipunktregler, der einen Sollwert kennt und über einen Größer- und Kleiner-Vergleich etwas ein- und ausschaltet. In Abhängigkeit von der Solltemperatur und der gemessenen Temperatur wird dann eine Heizung oder ein Lüfter ein- oder ausgeschaltet. Die Idee ist, die Heizung oder den Lüfter so zu steuern, dass die Temperatur bei dem gewünschten Sollwert liegt.

Regelung beim Heizen

- Wenn die gemessene Temperatur unter dem Sollwert liegt, dann wird die Heizung eingeschaltet.
- Wenn die gemessene Temperatur über dem Sollwert liegt, dann wird die Heizung ausgeschaltet.

Regelung beim Kühlen

- Wenn die gemessene Temperatur über dem Sollwert liegt, dann wird der Lüfter eingeschaltet.
- Wenn die gemessene Temperatur unter dem Sollwert liegt, dann wird der Lüfter ausgeschaltet.



EIN/AUS-Temperaturregelung (Zweipunktregelung) (2)

Innerhalb einer Schleife wird der Temperatursensor des Raspberry Pi Pico abgefragt. Anschließend wird der binäre Wert in zwei Stufen in die Temperatur in Grad Celsius umgerechnet. Danach wird die Temperatur mit dem Sollwert verglichen. Liegt die Temperatur darüber, wird die LED auf dem Pico eingeschaltet. Liegt sie darunter, wird die LED ausgeschaltet.

Um das Programm sinnvoll testen zu können, muss man zuerst die aktuelle Temperatur wissen. Welcher Sollwert zum Testen sinnvoll ist, hängt von der Umgebungstemperatur ab. Es kann gut sein, dass man das Programm mehrmals starten, stoppen, ändern und speichern muss, bis man ungefähr im Temperaturbereich liegt, mit dem man das Programm sinnvoll testen kann. Und dann muss man noch irgendwie die Temperatur erhöhen und senken können.

- Temperatur erhöhen: Lege einen Finger auf den Mikrocontroller (großer Chip). Der Temperatursensor befindet sich in der Nähe.
- Temperatur senken: Puste den Pico mehrmals an.

```
# Bibliotheken laden
from machine import Pin, ADC
from time import sleep

# Sollwert der Temperatur
soll_temp = 23.5

# Wartezeit bis zur nächsten Messung (Sekunden)
zeit = 3

# Initialisierung des ADC4
sensor_temp = ADC(4)
conversion_factor = 3.3 / (65535)

# Initialisieren der Onboard-LED
led = Pin('LED', Pin.OUT, value=0)

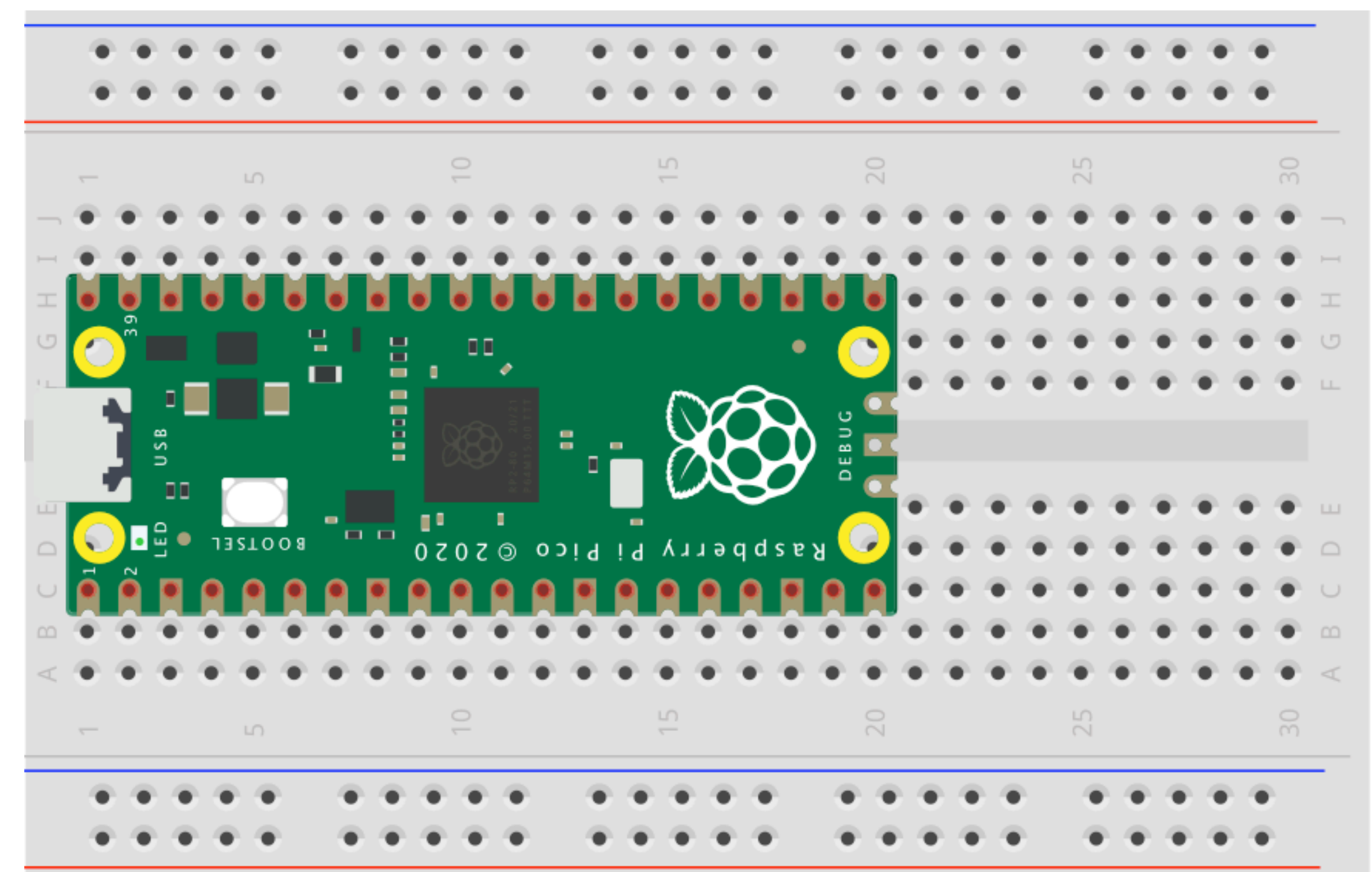
# Wiederholung (Endlos-Schleife)
while True:
    # Temperatur-Sensor als Dezimalzahl lesen
    read = sensor_temp.read_u16()
    # Dezimalzahl in eine reelle Zahl umrechnen
    spannung = read * conversion_factor
    # Spannung in Temperatur umrechnen
    temperatur = 27 - (spannung - 0.706) / 0.001721
    # Ausgabe in der Kommandozeile/Shell
    print("Temperatur (°C): ", temperatur)
    # EIN und AUS
    if temperatur > soll_temp:
        led.on()
    else:
        led.off()
    # Warten
    sleep(zeit)
```


Schwellwert-Temperaturregelung (1)

Ein Nachteil der sehr einfachen EIN/AUS-Temperaturregelung ist, dass immer beim Überschreiten bzw. Unterschreiten des Sollwerts Ein- und Ausgeschaltet wird. Das kann zur Folge haben, dass ein Lüfter oder eine Heizung ständig ein- und ausgeschaltet wird, wenn die Temperatur um den Sollwert herum schwankt.

Manche Kühl- oder Heizsysteme sind allerdings sehr träge, weshalb mal zum Einschalten und Ausschalten mit separaten Schwellwerten arbeitet.

Deshalb macht eine Schwellwert-Temperaturregelung Sinn, bei der bei unterschiedlichen Temperaturen ein- und ausgeschaltet wird.



EIN/AUS-Temperaturregelung (Zweipunktregelung) (2)

Der Programmcode der Schwellwert-Temperaturregelung sieht zwei unterschiedliche Temperaturen zum Einschalten und Ausschalten vor.

Die Wahl der richtigen Temperaturen der Schwellwerte sind entscheidend für die korrekte Funktion des Programms. Wenn man die Werte für EIN und AUS zu eng wählt, dann kommt das einer einfachen EIN/AUS-Temperaturregelung gleich. Dann hätte man nichts gewonnen.

Außerdem ergeben sich Schwierigkeiten den Programmcode zu testen. Welche Schwellwerte zum Testen sinnvoll sind, hängt von der Umgebungstemperatur ab. Es kann gut sein, dass man das Programm mehrmals starten, stoppen, ändern und speichern muss, bis man ungefähr im Temperaturbereich liegt, mit dem man das Programm sinnvoll testen kann. Und dann muss man noch irgendwie die Temperatur erhöhen und senken können.

- Temperatur erhöhen: Lege einen Finger auf den Mikrocontroller (großer Chip). Der Temperatursensor befindet sich in der Nähe.
- Temperatur senken: Puste den Pico mehrmals an.

```
# Bibliotheken laden
from machine import Pin, ADC
from time import sleep

# Schwellwerte (Temperatur)
ein_temp = 26.0
aus_temp = 25.5

# Wartezeit bis zur nächsten Messung (Sekunden)
zeit = 3

# Initialisierung des ADC4
sensor_temp = ADC(4)
conversion_factor = 3.3 / (65535)

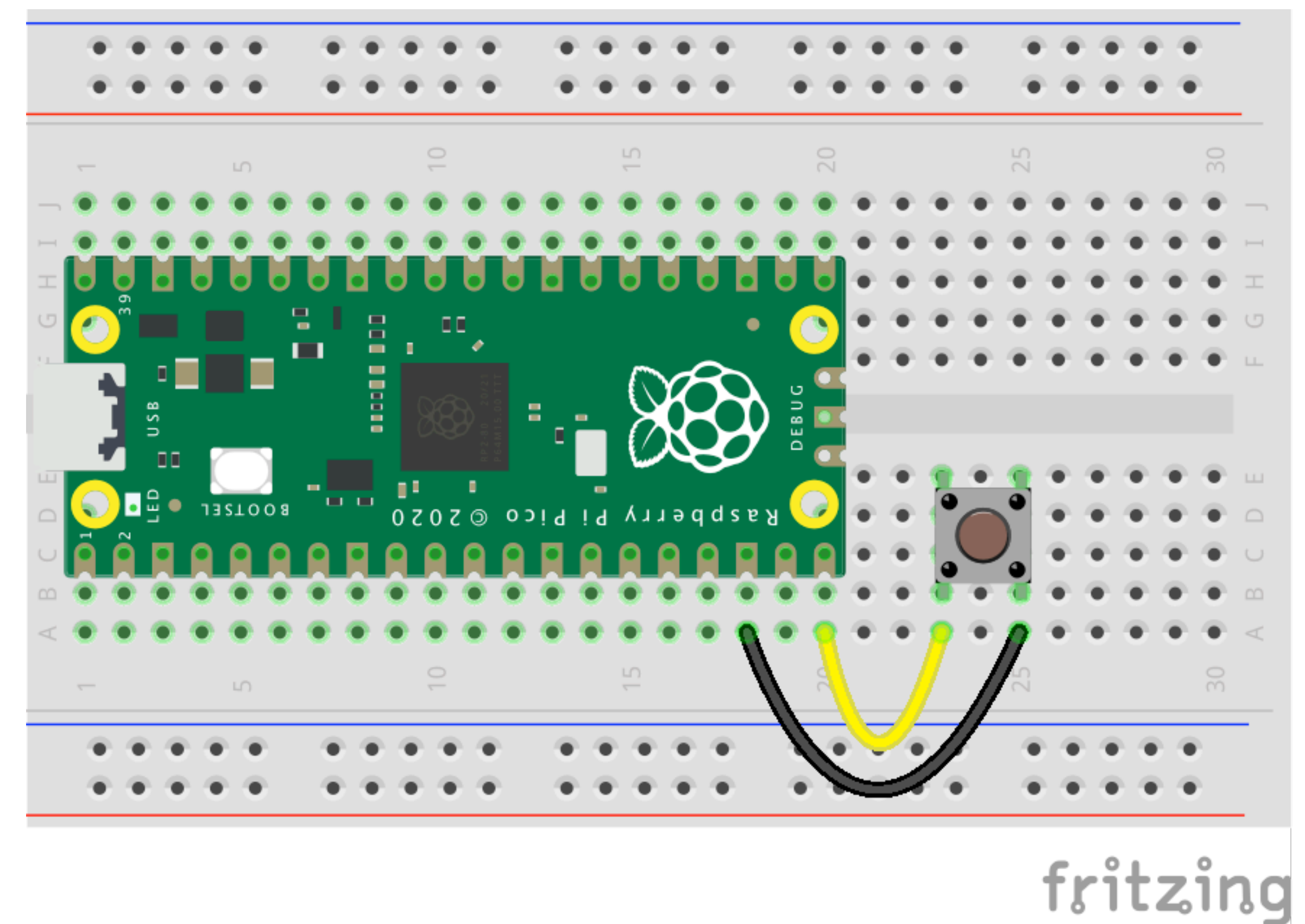
# Initialisieren der Onboard-LED
led = Pin('LED', Pin.OUT, value=0)

# Wiederholung einleiten (Schleife)
while True:
    # Temperatur-Sensor als Dezimalzahl lesen
    read = sensor_temp.read_u16()
    # Dezimalzahl in eine reelle Zahl umrechnen
    spannung = read * conversion_factor
    # Spannung in Temperatur umrechnen
    temperatur = 27 - (spannung - 0.706) / 0.001721
    # Ausgabe in der Kommandozeile/Shell
    print("Temperatur (°C): ", temperatur)
    print()
    # EIN und AUS
    if led.value() == 0 and temperatur > ein_temp:
        led.on()
    if led.value() == 1 and temperatur < aus_temp:
        led.off()
    # Warten
    sleep(zeit)
```

Stromausfall-Monitor bei Abwesenheit (1)

Unser Stromnetz ist vergleichsweise stabil. Normalerweise hat man keinen Bedarf einen Netzausfall oder Stromausfall festzustellen. Wenn man einen Stromausfall jetzt doch mal bei Abwesenheit feststellen will, dann gibt es dafür nicht sehr viele Möglichkeiten. Am einfachsten wäre da noch ein Radiowecker dessen Uhrzeit sich auf „0:00“ zurückstellt, wenn er keinen Strom mehr bekommt. Wenn man zurückkommt und die Uhrzeit nicht mehr stimmt, dann weiß man, dass der Strom weg war. Doch wer hat noch so einen Radiowecker? Man bedenke auch den Stromverbrauch von so einem Gerät.

Mit einem Raspberry Pi Pico lässt sich so ein Stromausfall vergleichsweise einfach feststellen.



Stromausfall-Monitor bei Abwesenheit (2)

Der Programmcode initialisiert am Anfang die Onboard-LED und einen Taster. Anschließend wird geprüft, ob der Taster gedrückt ist. Wenn ja, dann blinkt die LED ein paar mal und bleibt dann aus. Wenn nein, dann geht der Programmcode von einem Neustart nach einem Stromausfall aus und die LED leuchtet.

Zur Bedienung:

- Immer wenn das Programm neu gestartet wird (zum Beispiel durch Trennen und Verbinden vom Strom), wird die LED immer leuchten. Bei einem Neustart geht der Pico davon aus, dass ein Stromausfall vorgefallen ist.
- Zum RESET oder Zurücksetzen des Alarmzustands muss der Taster bei einem Neustart für ein paar Sekunden gedrückt sein.
- Wenn die LED mehrmals blinkt, hat der Programmcode den RESET erkannt. Der Taster kann dann losgelassen werden.

```
# Bibliotheken laden
import machine
import time

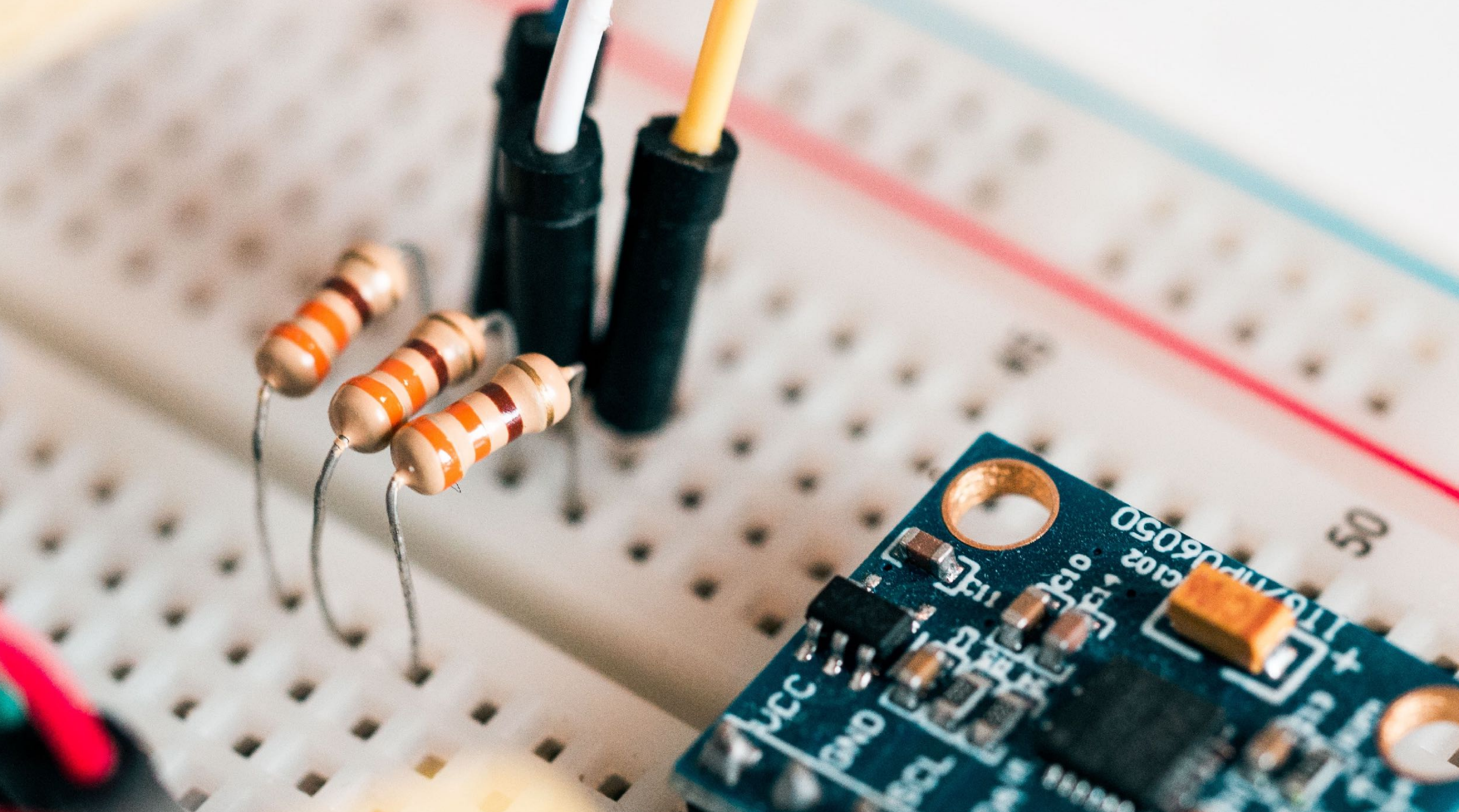
# Initialisierung
led = machine.Pin('LED', machine.Pin.OUT, value=0)
btn = machine.Pin(15, machine.Pin.IN,
machine.Pin.PULL_UP)

# 2 Sekunden warten
time.sleep(2)

# Prüfen, ob Button gedrückt
if btn.value() == 0:
    for i in range (5):
        led.toggle()
        time.sleep(0.5)
    # LED aus
    led.off()
else:
    # LED ein
    led.on()

# Tiefschlaf im Stand-alone-Betrieb
#machine.deepsleep()
```

Der Programmcode ist so geschrieben, dass das Programm zuerst mit Thonny gestartet und geprüft werden kann. Erst wenn der Raspberry Pi Pico im Stand-alone-Betrieb arbeiten soll, sollte in der letzten Zeile das #-Zeichen entfernt werden. Damit wird die Tiefschlaf-Funktion zum Strom sparen am Programmende in Betrieb gesetzt.



Lust auf mehr?

Elektronik-Set Sensor Edition



Das Elektronik-Set Sensor Edition ist eine Sammlung beliebter Sensoren und Bauteile für die Hardware-nahe Programmierung mit Mikrocontrollern und Mini-Computern.

Es werden verschiedene Bereiche abgedeckt, wie Temperatur, Bewegungserkennung, Akustikerkennung, Lichterkennung, Bewegen und Lichtsteuerung.

Spannende Experimente und Anwendungen aus dem Bereich Smart Home, Robotik und Automation warten auf Dich.

<https://www.elektronik-kompendium.de/shop/elektronik-set/sensor-edition>

Elektronik-Set Eingabe Ausgabe Edition

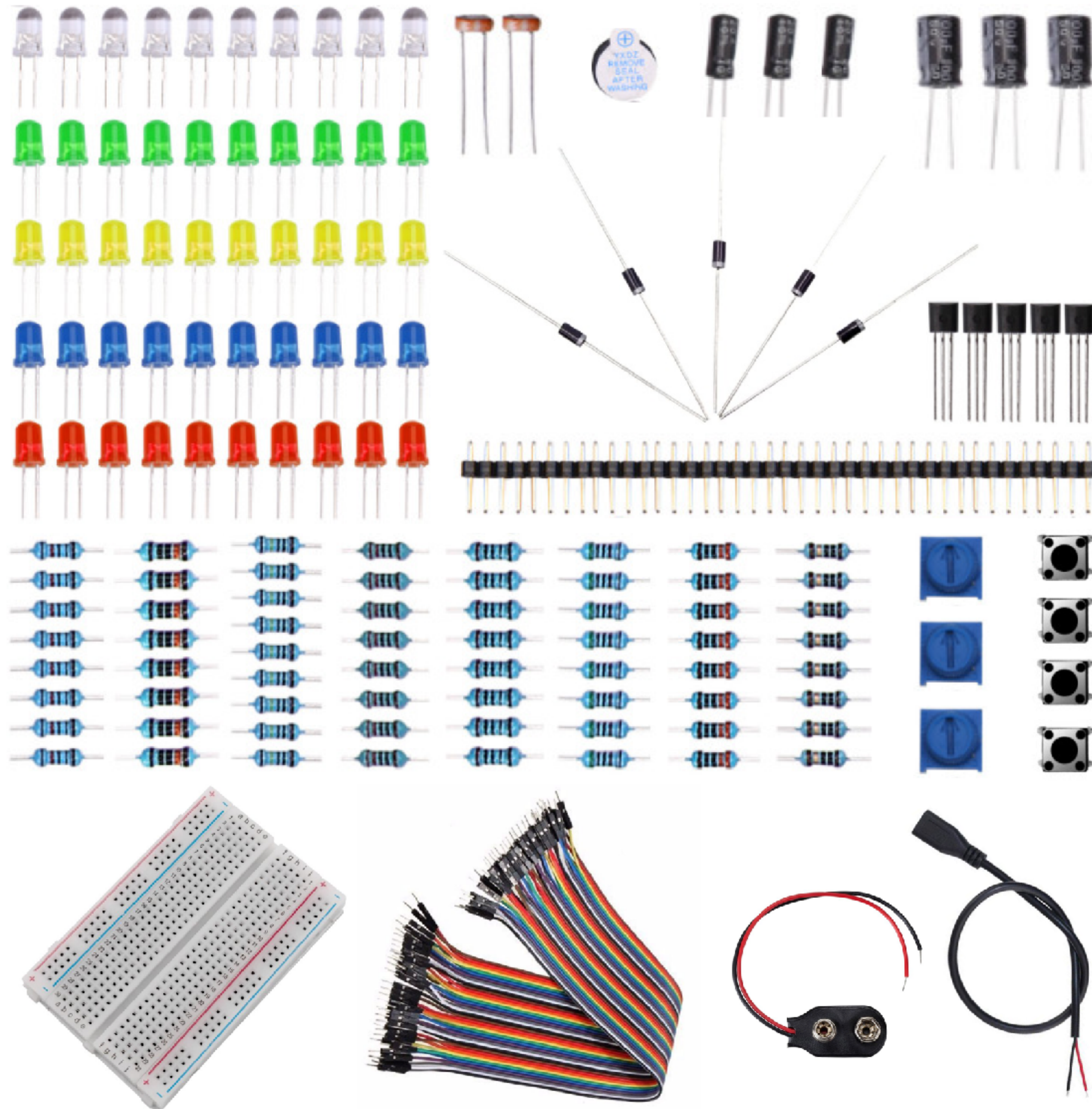


Das Elektronik-Set Eingabe Ausgabe Edition ist eine Sammlung beliebter Bauteile zur Dateneingabe, Steuerung und Datenausgabe für die Hardware-nahe Programmierung mit Mikrocontrollern und Mini-Computern.

Wenn Sie bereits das Elektronik-Set Pico Edition oder Pico WLAN Edition haben, dann können Sie es mit diesem Elektronik-Set sinnvoll erweitern und viele weitere spannende Experimente durchführen.

<https://www.elektronik-kompendium.de/shop/elektronik-set/ingabe-ausgabe-edition>

Elektronik-Set Starter Edition



Mit Elektronik ohne Löten experimentieren

Das Elektronik-Set Starter Edition ist die optimale Ergänzung zum Elektronik-Guide. Das Elektronik-Set enthält alle und noch viel mehr Bauteile, um alle Schaltungen und Experimente nachzubauen.

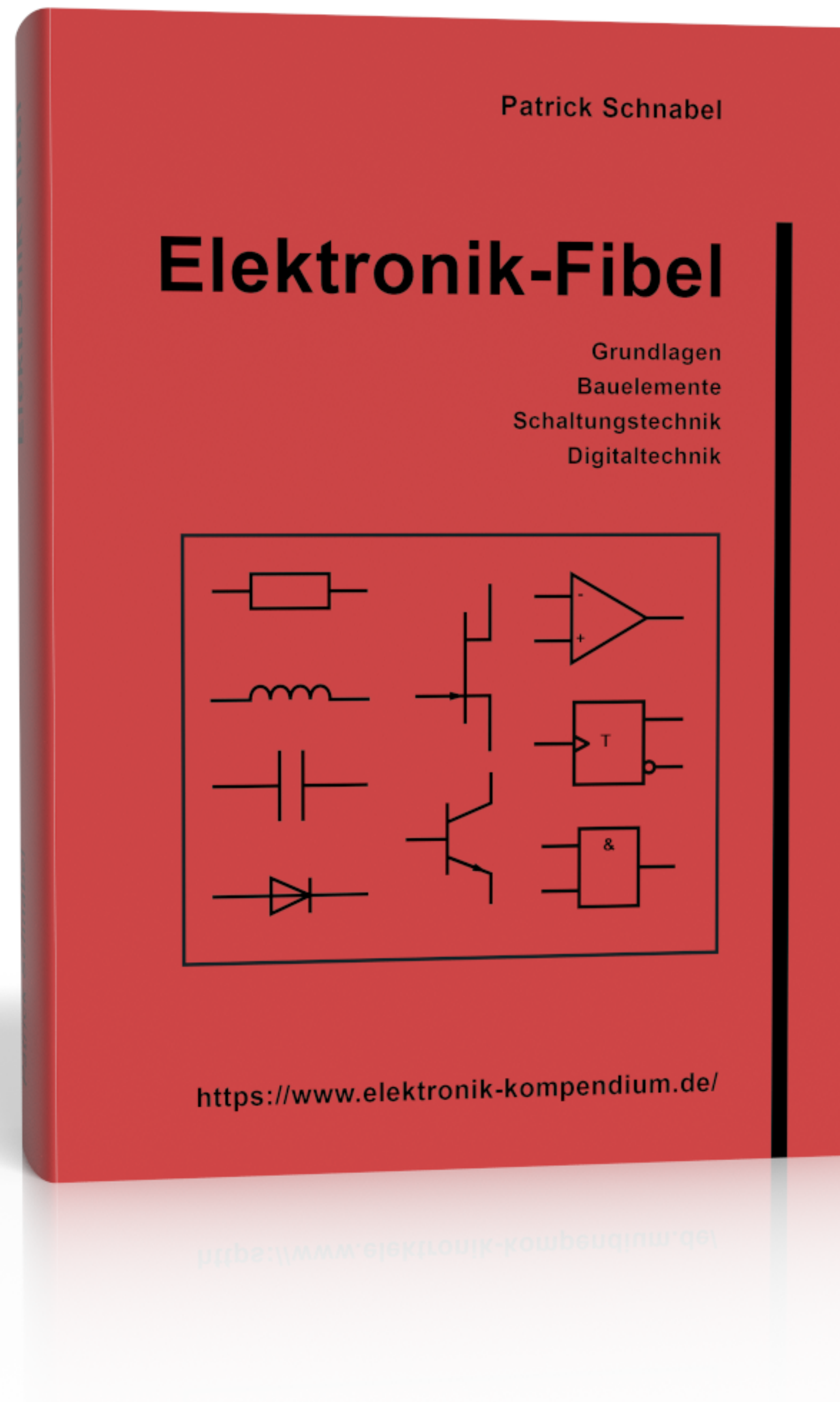
Zusätzlich enthält das Elektronik-Set:

- 1 Steckbrett mit 400 Pins
- 40 Verbindungskabel
- 1 Batterie-Clip für einen 9-Volt-Block
- 1 Micro-USB-Adapter für ein USB-Ladegerät

Nicht im Lieferumfang enthalten und zusätzlich empfohlen:

- 9-Volt-Block-Batterie, USB-Netzteil oder USB-Ladegerät

<https://www.elektronik-kompendium.de/shop/elektronik-set/starter-edition>



Elektronik - einfach und leicht verständlich

Elektronik muss nicht schwer sein. Die Elektronik-Fibel beschreibt die Grundlagen der **Elektronik einfach und leicht verständlich**, so dass der Einstieg in die Elektronik so einfach wie möglich gelingt.

Die Elektronik-Fibel eignet sich besonders **zum Lernen auf Klassenarbeiten, Klausuren und Prüfungen** oder als Nachschlagewerk für die Schule und Ausbildung.

Mit den vielen grafischen Abbildungen, Formeln, Schaltungen und Tabellen dient diese Buch dem Einsteiger und auch dem Profi immer und überall als **unterstützende und nützliche Lektüre**.

<https://www.elektronik-kompodium.de/shop/buecher/elektronik-fibel>