

# Grundlagen der Programmierung des Raspberry Pi Pico mit Micropython

Schritt-für-Schritt-Anleitung, um einige der vielen Möglichkeiten des Picos kennenzulernen.

- [LEDs schalten](#)
- [Ampelschaltung mit LEDs](#)
- [Der Motor](#)
- [Der Ultraschallsensor](#)
- [Der Infrarotsensor](#)

# LEDs schalten

## Die interne LED

Der Pico hat eine interne LED, die folgendermaßen angesteuert werden kann:

```
import time
from machine import Pin

#led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
led.value(1)
time.sleep(1)
led.value(0)
```

Soll die LED unabhängig vom Programmablauf blinken, dann kann man das mit einem Timer realisieren.

```
from machine import Pin,Timer
# led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
timer = Timer()
timer.init(freq=2, mode=Timer.PERIODIC, callback=lambda t: led.toggle())
```

Soll der Timer beendet werden, so kann man das mit dem Befehl `timer.deinit()` erreicht werden.

Möchte man mehr Kontrolle haben oder komplexere Funktionen einbauen, dann geht das so:

```
import time
from machine import Pin,Timer

# led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
led.value(1)
time.sleep(1)
led.value(0)
```

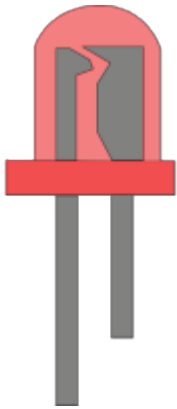
```
def blink(timer):  
    led.toggle()  
  
timer = Timer()  
Timer().init(freq=2, mode=Timer.PERIODIC, callback=blink)
```

Ein Timer läuft auch nach dem Ende des Programms weiter. Also nicht wundern, wenn es weiter blinkt.

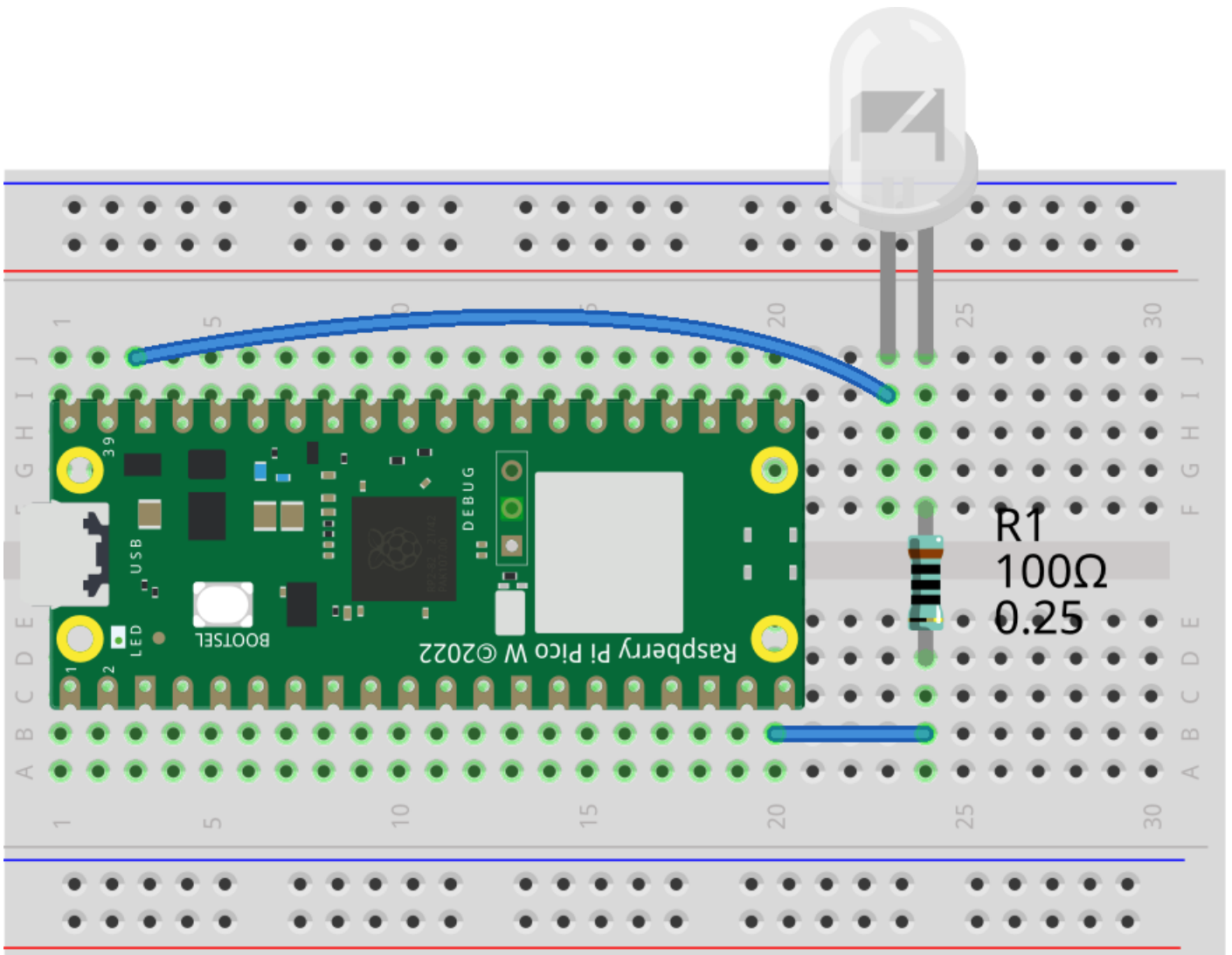
**Aufgabe:** Die LED zum Blinken bringen.

## Eine externe LED

Das war zwar schon spannend, aber nun wollen wir eine externe LED an den Pico anschließen. Dazu benötigen wir eine LED, einen 100  $\Omega$  Widerstand sowie zwei Kabel. Stecke die Schaltung genau so zusammen, wie abgebildet. Achte vor allem darauf, dass die LED richtig herum ist.



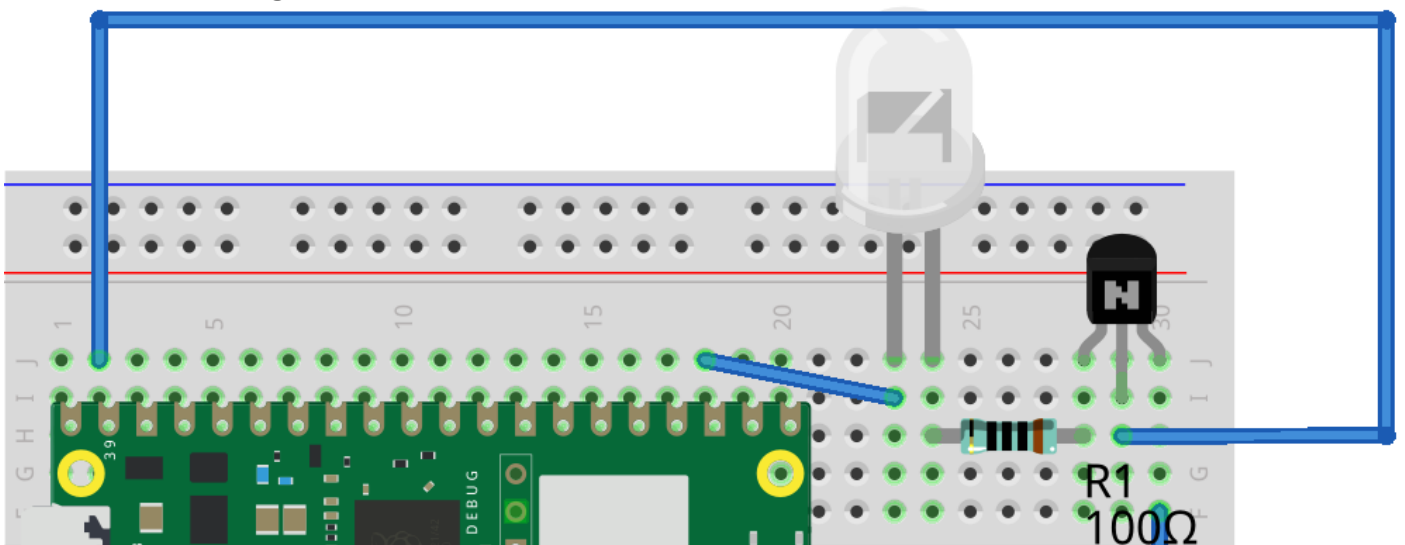
**Hier ist der dazugehörige Schaltplan:**

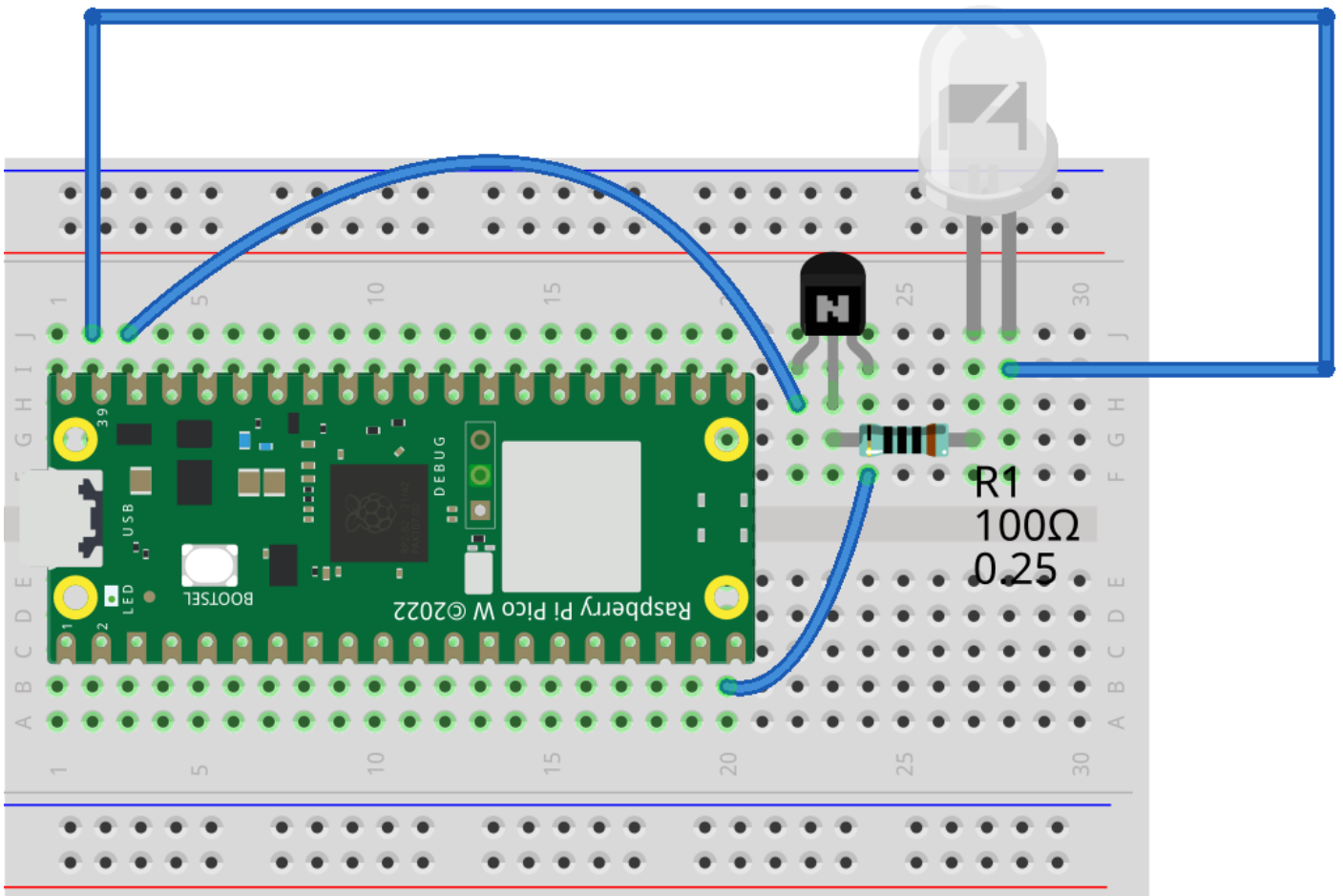


fritzing

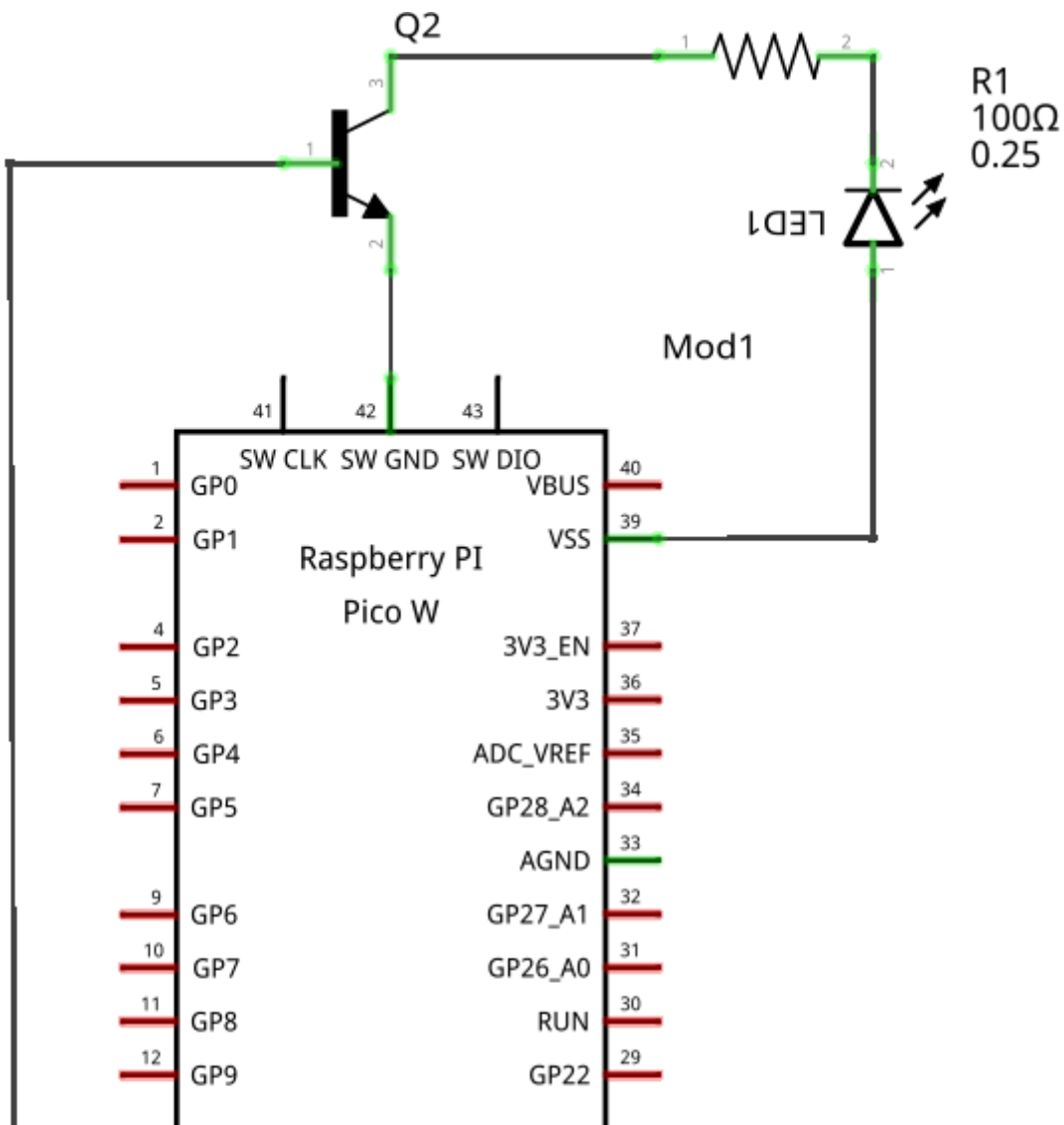
## Externe LED mit Transistor

Der Pico liefert an den Pins nur eine Spannung von 3,3 Volt und die Stromstärke ist auch nicht sehr hoch. Es ist leicht möglich, die Anschlüsse zu überlasten, zwar nicht mit nur einer LED, jedoch bleibt es dabei ja nicht. Daher müssen Transistoren verwendet werden. Es gibt zwei mögliche Schaltungen für die LED mit Transistor. Der Unterschied ist nur, ob die Schaltung auf Anoden- oder auf Kathodenseite geschieht.





fritzing



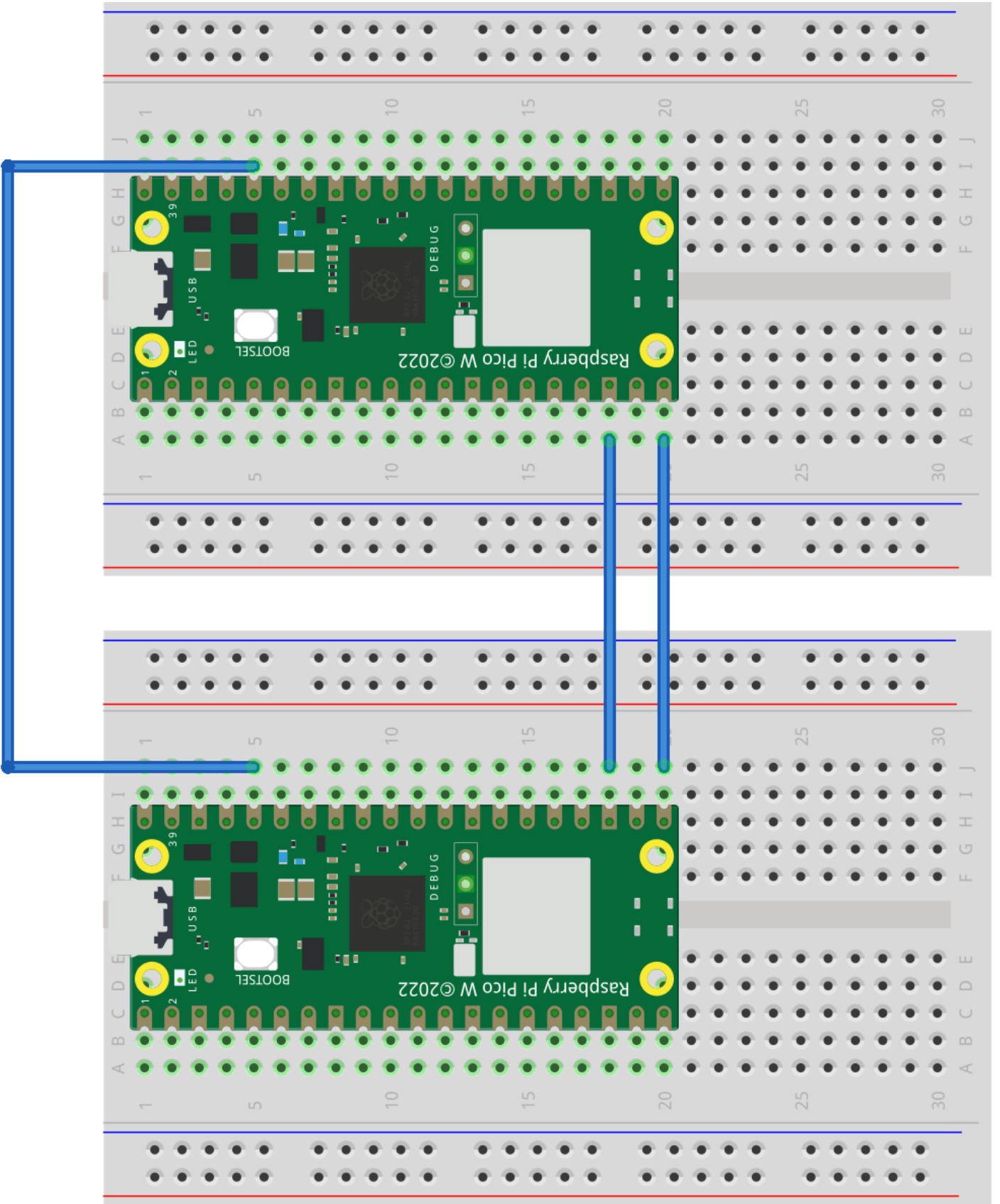
# Regelung der Helligkeit

Egal, ob du die LED mit oder ohne Transistor betreibst, so ist die Helligkeit bislang immer die gleiche. Glühlampen lassen sich sehr einfach dimmen, indem man die Spannung regelt. Bei der LED funktioniert das nicht, da die Spannung an der LED immer dieselbe ist. Die LED ist nämlich kein ohmsches Bauteil. Außerdem haben wir mit dem Pico die Schwierigkeit, dass er ein digitales Gerät ist und bekanntermaßen kennen digitale Geräte nur 1 und 0 oder an und aus. Wie lässt sich damit also die Helligkeit regulieren?

# Ampelschaltung mit LEDs

## Ampelschaltung

Verwende die LED-Ampel, um eine Ampelschaltung zu programmieren. Schaltet dann mehrere Ampeln zu einer Kreuzung zusammen, indem ihr die Picos miteinander kommunizieren lasst. Dazu müsst ihr einen Pin auf dem Pico, der Befehle erhält, als Eingangspin definieren.



fritzing

# Empfange Daten auf Pin 16 und blinke mit der internen LED

```
from machine import Pin

sensor=Pin(16, Pin.IN, Pin.PULL_UP)
led=Pin(25, Pin.OUT)
while True:
    while sensor.value():
        led.value(1)
    led.value(0)
```

# Sende Daten mit Pin 15 und blinke mit der internen LED

```
import time
from machine import Pin

#led=Pin("LED", Pin.OUT) # Für den Pico mit eingebautem WLAN
led=Pin(25, Pin.OUT) # Für den Pico ohne WLAN
sender=Pin(15,Pin.OUT)
while True:
    led.value(1)
    sender.value(1)
    time.sleep(1)
    led.value(0)
    sender.value(0)
    time.sleep(1)
```

Nur einer der beiden Picos muss über USB an Strom angeschlossen werden. Beide Programme werden unter dem Namen "main.py" auf dem Pico abgespeichert, dann laufen die Programme automatisch, sobald die Picos Strom bekommen.

# Zwei Ampeln koordinieren

# Code für den Sender

```
from machine import Pin
from time import sleep_ms

red = Pin(10,Pin.OUT)
yellow = Pin(11,Pin.OUT)
green = Pin(12,Pin.OUT)
trigger = Pin(15, Pin.OUT)

wait = 800
phase = 3000
alarm_wait = 500

def go_red():
    red.off()
    yellow.on()
    green.off()
    sleep_ms(wait)
    red.on()
    yellow.off()
    trigger.off()
    sleep_ms(wait*2)

def go_green():
    trigger.on()
    sleep_ms(wait*3)
    red.on()
    yellow.on()
    green.off()
    sleep_ms(wait)
    red.off()
    yellow.off()
    green.on()

while True:
    go_red()
    sleep_ms(phase)
    go_green()
```

```
sleep_ms(phase)
```

## Code für den Empfänger

```
from machine import Pin
from time import sleep_ms

red = Pin(10,Pin.OUT)
yellow = Pin(11,Pin.OUT)
green = Pin(12,Pin.OUT)
trigger = Pin(16, Pin.IN, Pin.PULL_UP)

wait = 800
phase = 3000
alarm_wait = 500

def go_red():
    red.off()
    yellow.on()
    green.off()
    sleep_ms(wait)
    red.on()
    yellow.off()

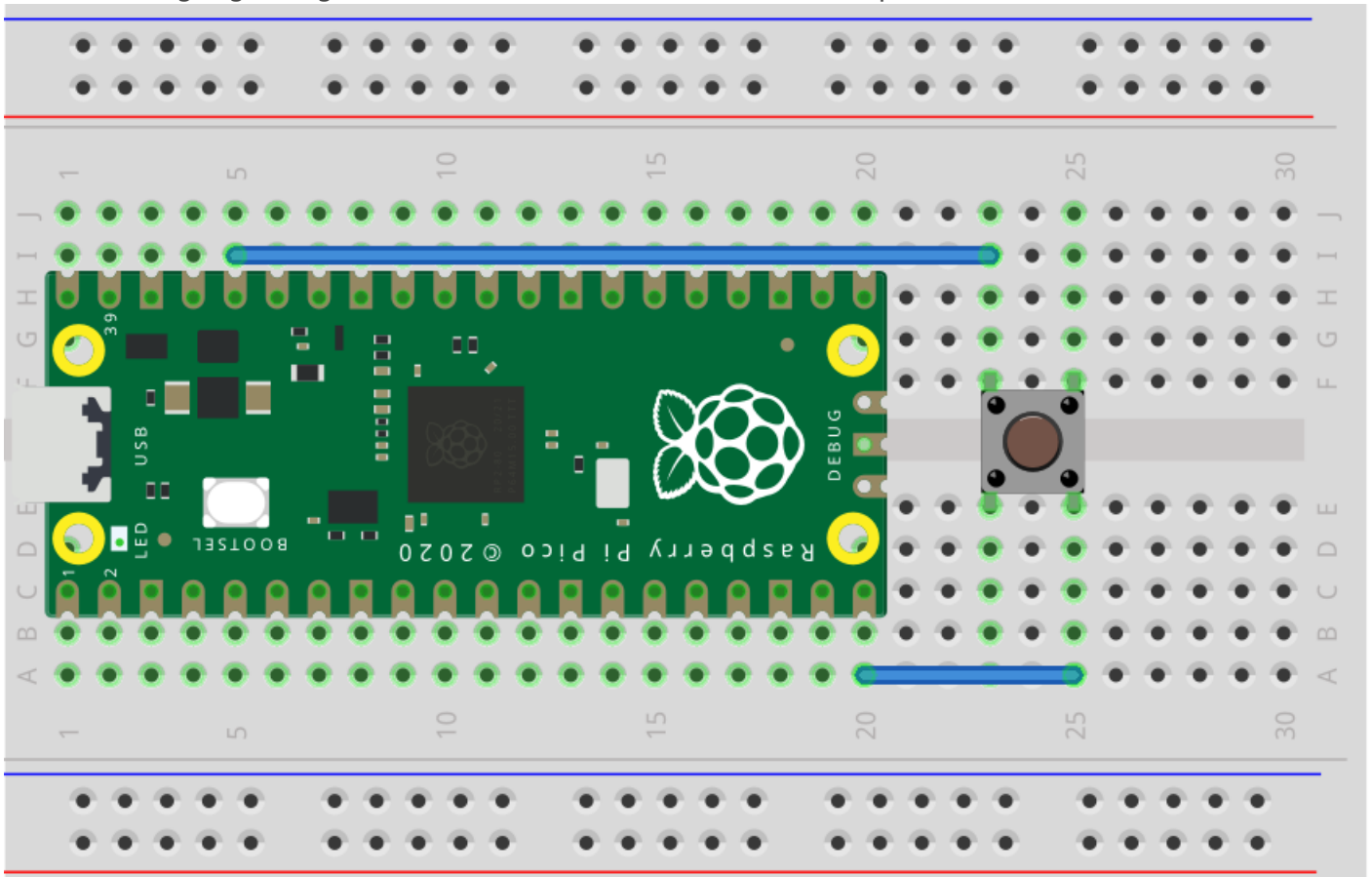
def go_green():
    red.on()
    yellow.on()
    green.off()
    sleep_ms(wait)
    red.off()
    yellow.off()
    green.on()

while True:
```

```
if trigger.value():# Wenn trigger == True
    go_red()
    while trigger.value():
        pass
if not trigger.value():
    go_green()
    while not trigger.value():
        pass
```

# Knopfsteuerung der Ampel

Für die Programmierung eines Ampelknopfes, muss man den Knopf *entprellen*, damit keine Geisterbewegungen registriert werden. Ein minimales Codebeispiel ist dieses:



fritzing

```
from machine import Pin
import time
```

```

button = Pin(15, Pin.IN, Pin.PULL_DOWN)
pressed = False
num_pressed = 0
last_pressed = 0
DEBOUNCE_WAIT = 100
def button_handler(pin):
    global pressed, num_pressed, last_pressed #mit dem Befehl global teilt man Python mit,
    dass man die Variabel verwenden möchte, die außerhalb der Funktion initialisiert wurde.
    while utime.ticks_diff(utime.ticks_ms(), last_pressed) < DEBOUNCE_WAIT: # Hier wird
    verhindert,dass mehrere Auslösungen hintereinander registriert werden.
        pass
    last_pressed = utime.ticks_ms()
    if not pressed:
        while time.ticks_diff(time.ticks_ms(), last_pressed) < DEBOUNCE_WAIT:
            pass
        if pin.value() == 1:
            pressed=True #Damit kann im Programmablauf der Knopfdruck registriert werden.
            num_pressed +=1
            print(pin.value(), "number presses: ", num_pressed)
            last_pressed = time.ticks_ms()
            pressed=False # Dies hier eher im weiteren Programmablauf verwenden.

button.irq(trigger=Pin.IRQ_RISING, handler=button_handler)

# Hier weiterer Programmablauf
while True:
    pass

```

[Programmiergrundkurs in Python](#)

# Der Motor

Ein Motor wird genauso gesteuert wie eine LED. Man kann ihn entweder einfach an- und ausschalten, oder mithilfe der Pulsweitenmodulation die Geschwindigkeit regeln. Probiert es einfach einmal aus. Da ein Motor deutlich mehr Leistung hat als eine LED, kann man den Raspberry Pi Pico schnell überlasten. Daher betreiben wir den Motor nur über einen Transistor. Es sorgt dafür, dass das Steuersignal des Picos verstärkt von der 9V Batterie an den Motor geleitet wird. Aber Achtung: Der Motor ist für dauerhaft 6V ausgerichtet und sollte daher nicht zu lange mit 9V betrieben werden.

Für diese Schaltung könnt ihr die Schaltung der LED mit Transistor ohne den Widerstand verwenden.

Der Code für diese Steuerung ist genauso wie für eine LED. In dem Schaltbild wird der Pin 15 benutzt.

## Richtungssteuerung

Wie könnte man nun die Drehrichtung des Motors ändern, ohne die Kabel umzustecken?

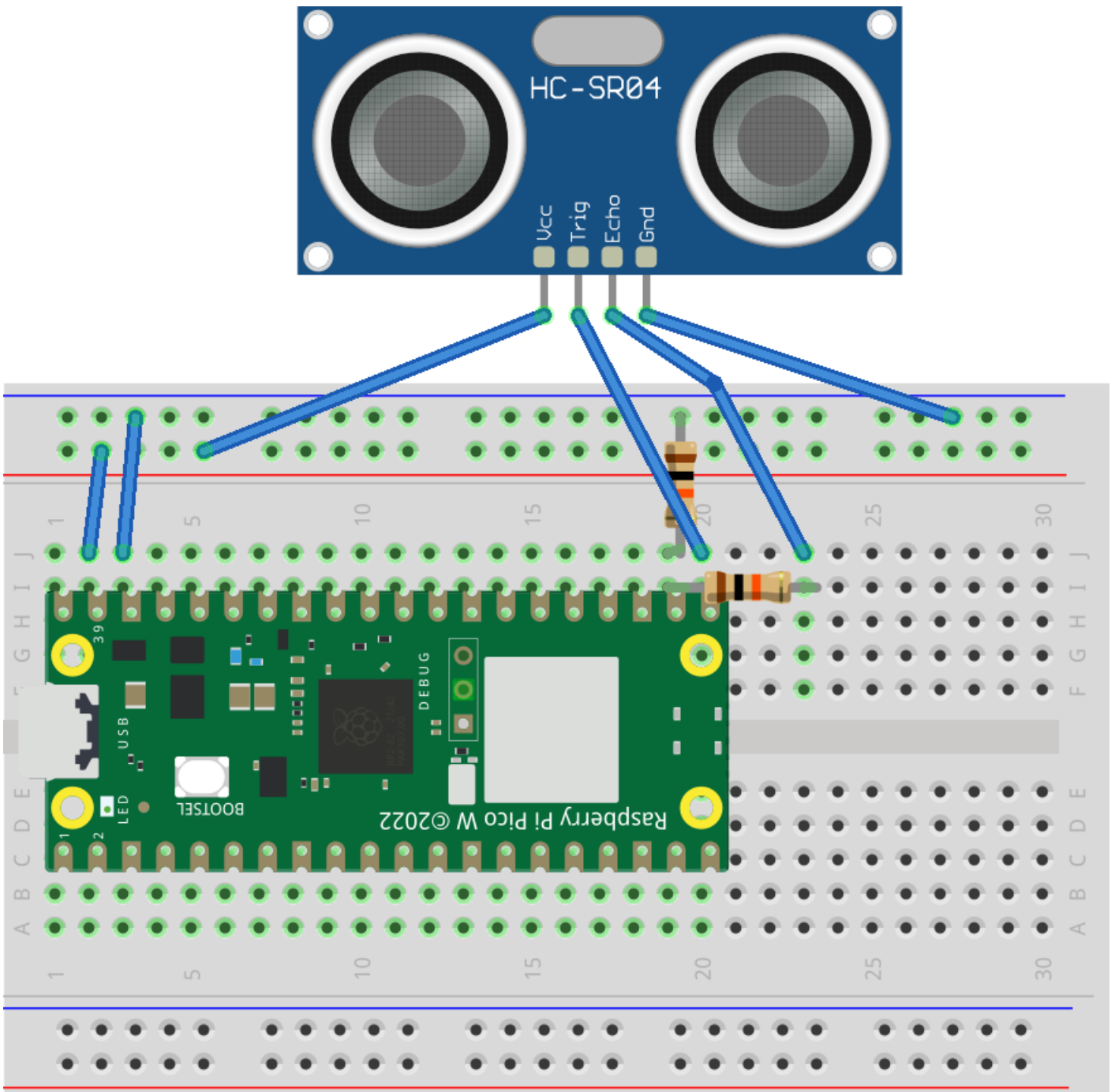
Überlegt erst einmal und dann schaut ihr euch das nächste Bauteil an:

[AB Die H-Brücke](#)

# Der Ultraschallsensor



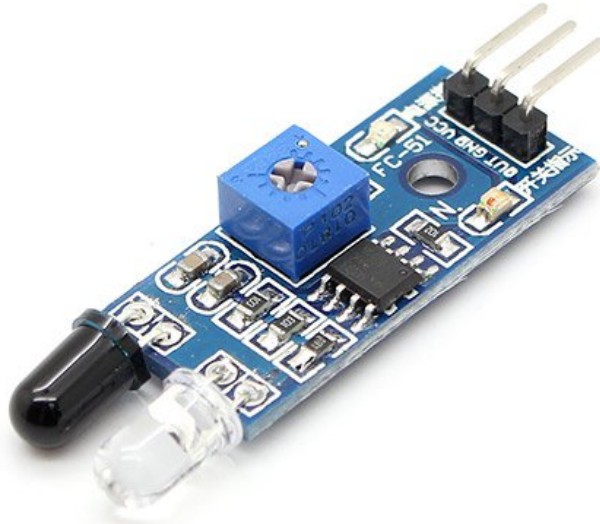
Schließt den Ultraschallsensor an den Pico an:



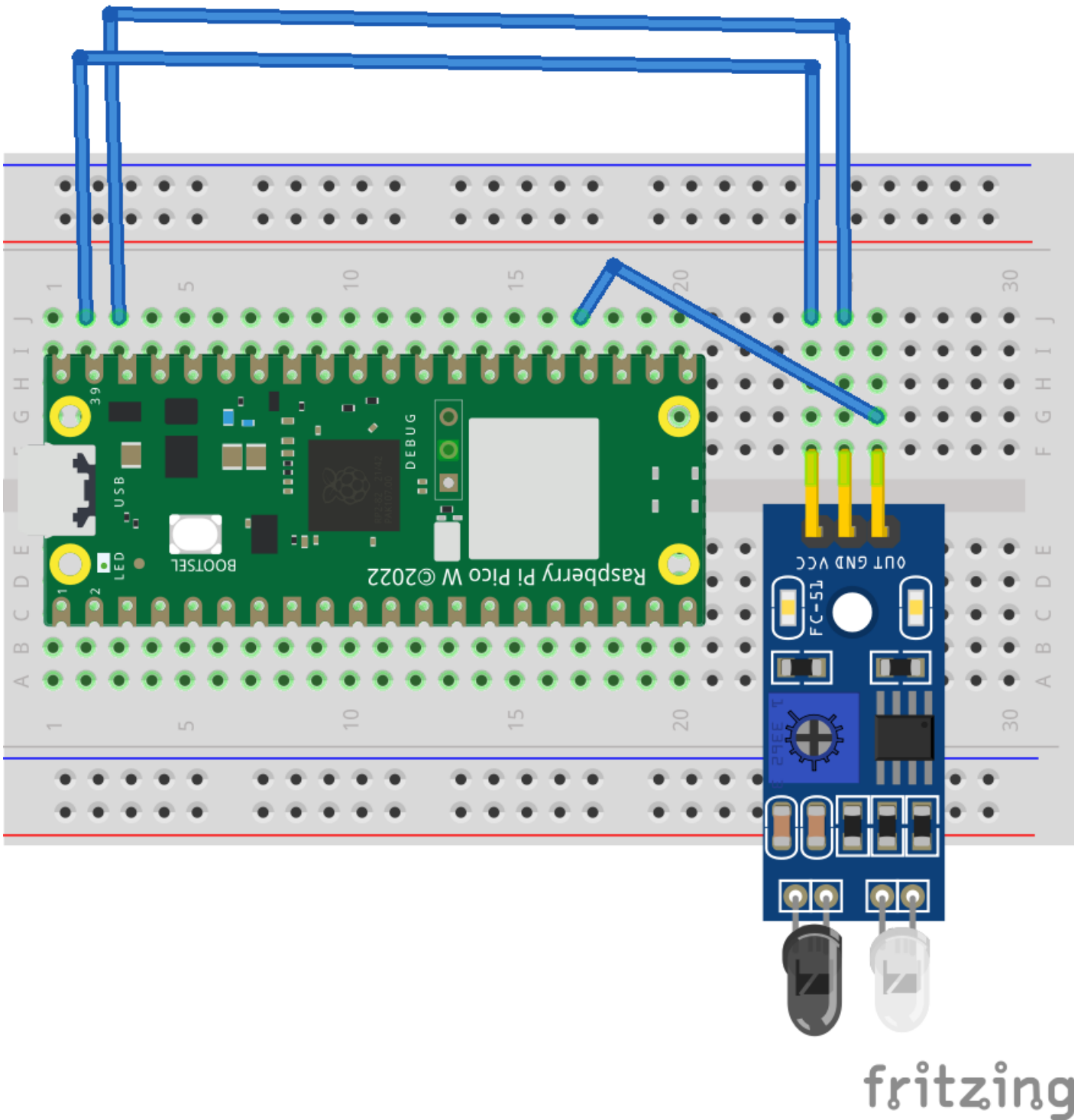
fritzing

Zur Ansteuerung des Ultraschallsensors wird die [Roboterbibliothek](#) benutzt.

# Der Infrarotsensor



Der IR-Sensor wird folgendermaßen an den Pico angeschlossen. Der OUT-Pin kann natürlich auch geändert werden.



So kann der Infrarotsensor ausgelesen werden:

```
from machine import *  
import time  
  
# Der Pin für den Infrarotsensor wird initialisiert.  
ir=Pin(16,Pin.IN,Pin.PULL_UP)  
  
while True:
```

```
print(ir.value()) #Es wird einmal der Wert ausgegeben, der am Pin anliegt.
while ir.value() == 0: # Solange der Wert 0 bleibt, ändert sich die Anzeige nicht.
    time.sleep_ms(50)
print(ir.value()) # Ist/Wird der Wert 1, wird erneut auf der Konsole ausgegeben.
while ir.value() == 1:
    time.sleep_ms(50) # Solange der Wert 1 bleibt, ändert sich die Anzeige nicht.
```

Für die Benutzung in den Robotern verwenden wir die [Roboterbibliothek](#).