

# 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](#)
- [Codebeispiele](#)

# 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)
```

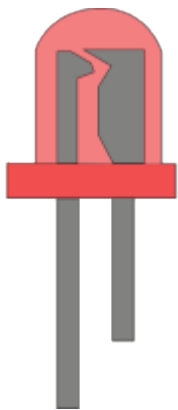
```
def blink(timer):  
    led.toggle()  
  
timer = Timer()  
Timer().init(freq=2, mode=Timer.PERIODIC, callback=blink)
```

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

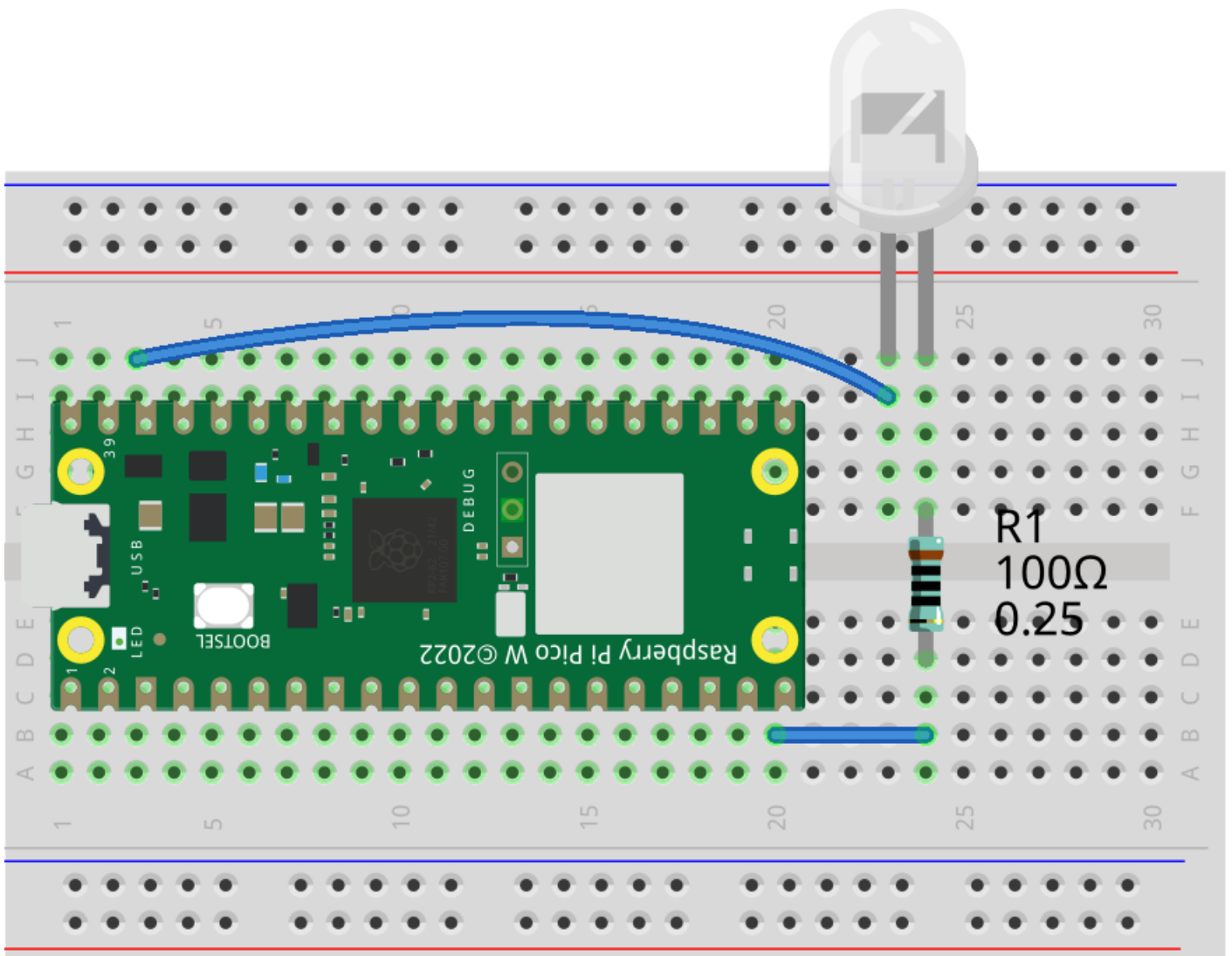
**Aufgabe:** Die LED zum Blinken bringen.

## 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.



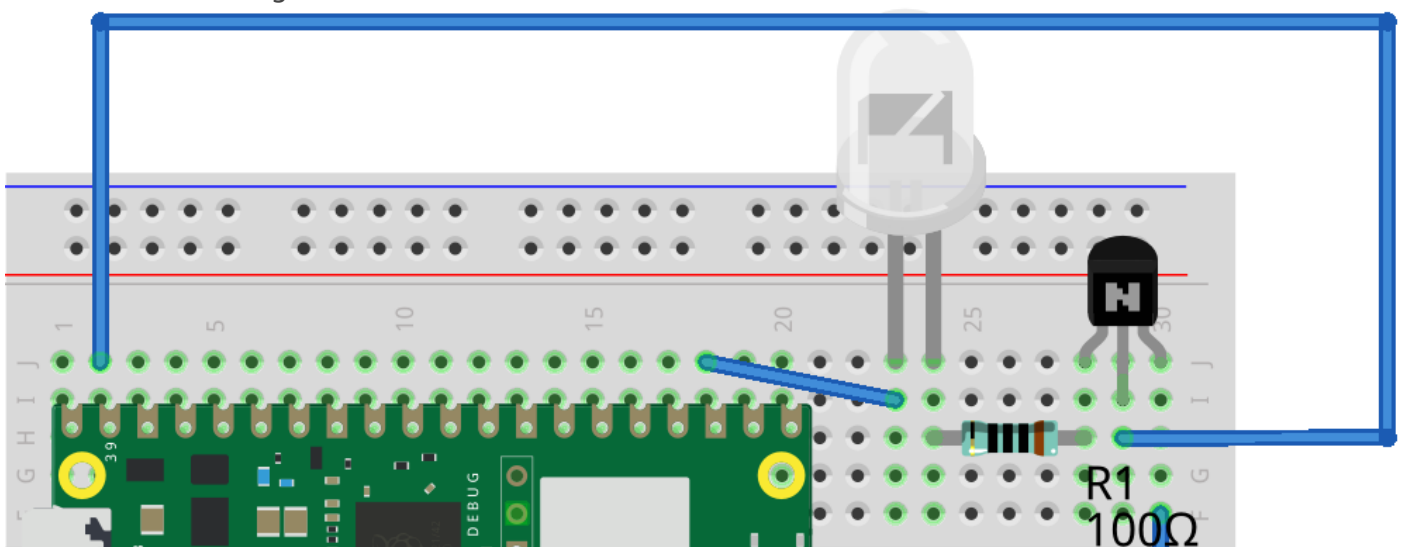
**Hier ist der dazugehörige Schaltplan:**

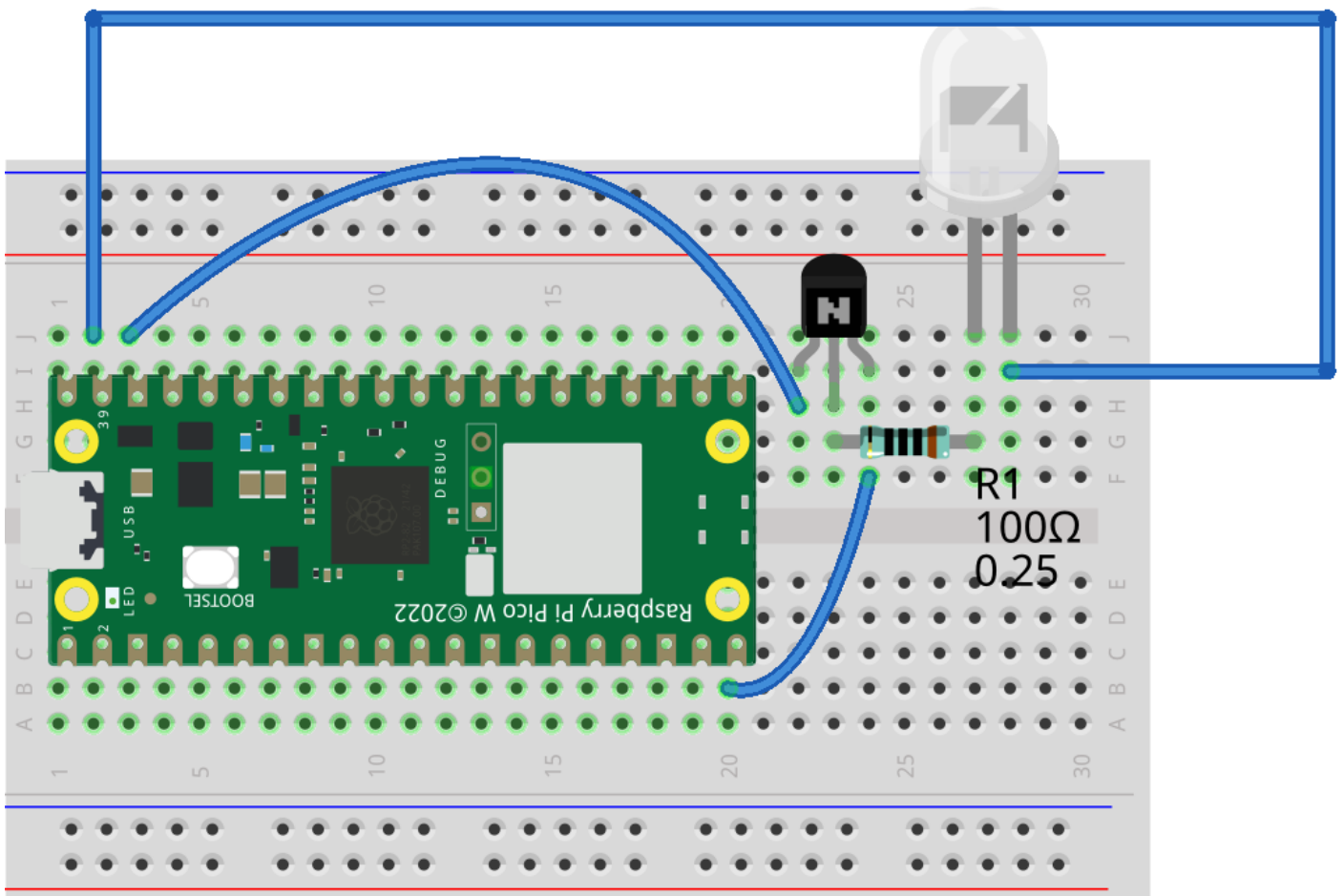


fritzing

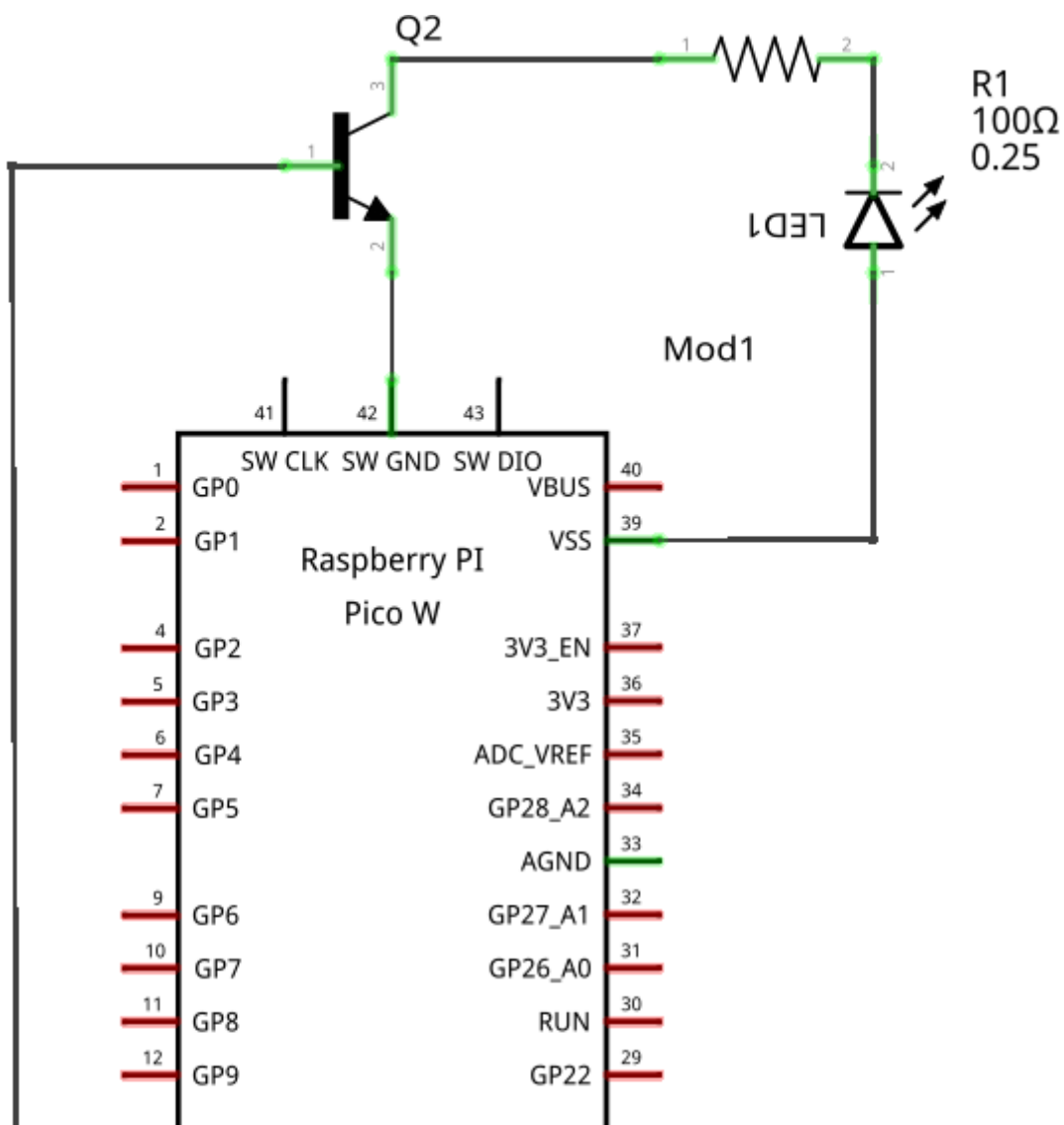
## Externe LED mit Transistor

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



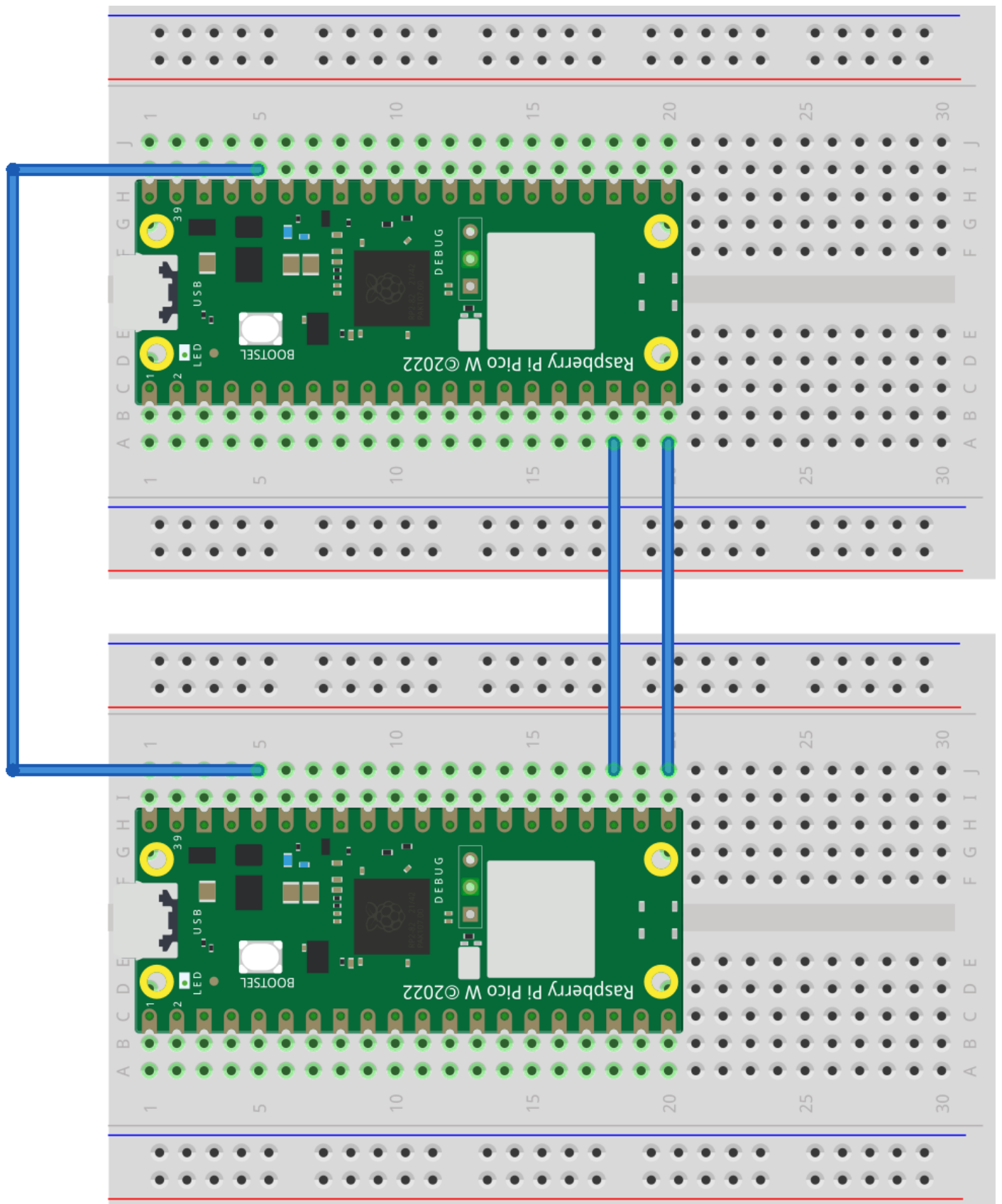
# Regelung der Helligkeit

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

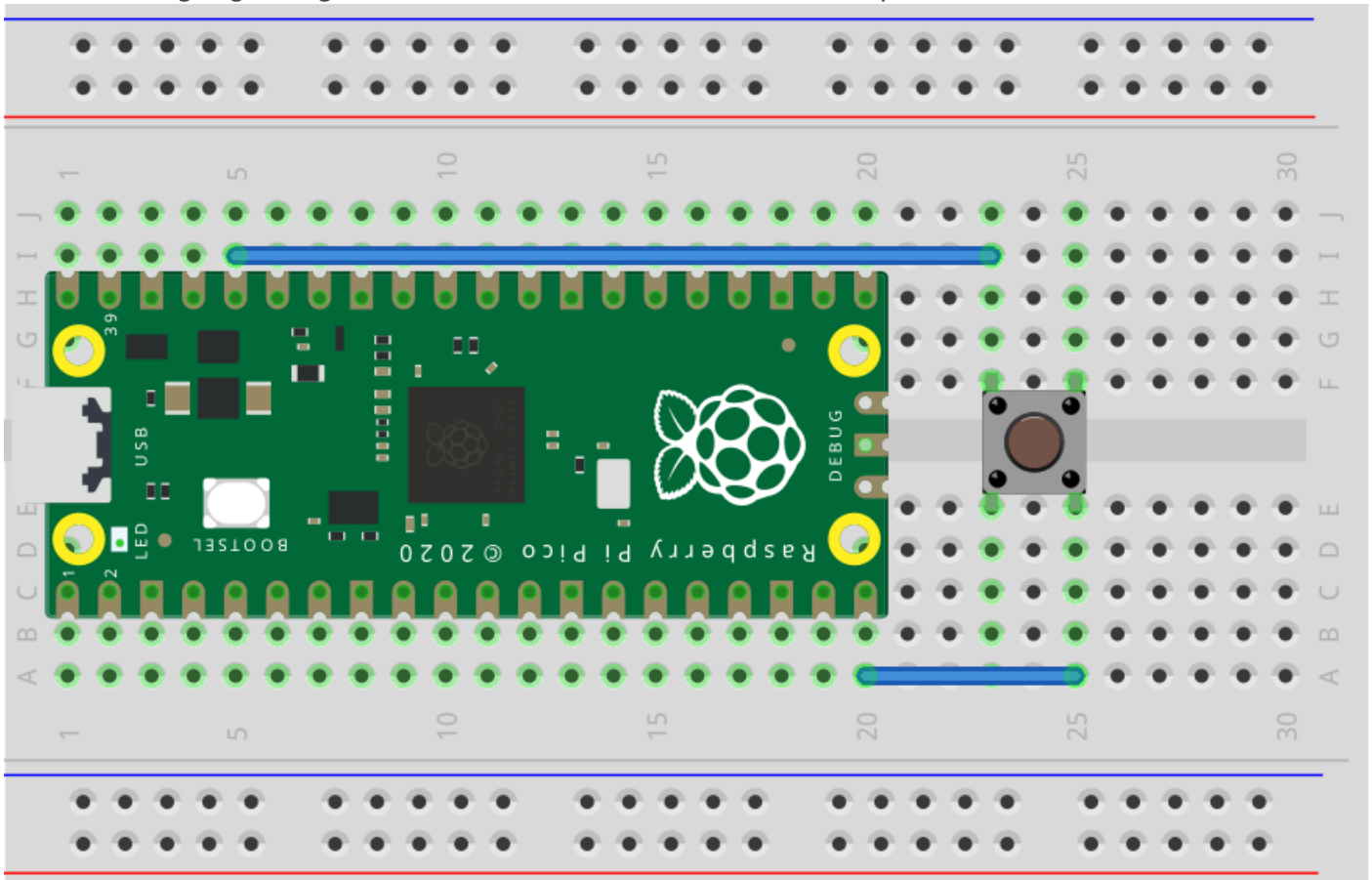
```
import time
from machine import Pin

#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.

## 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?

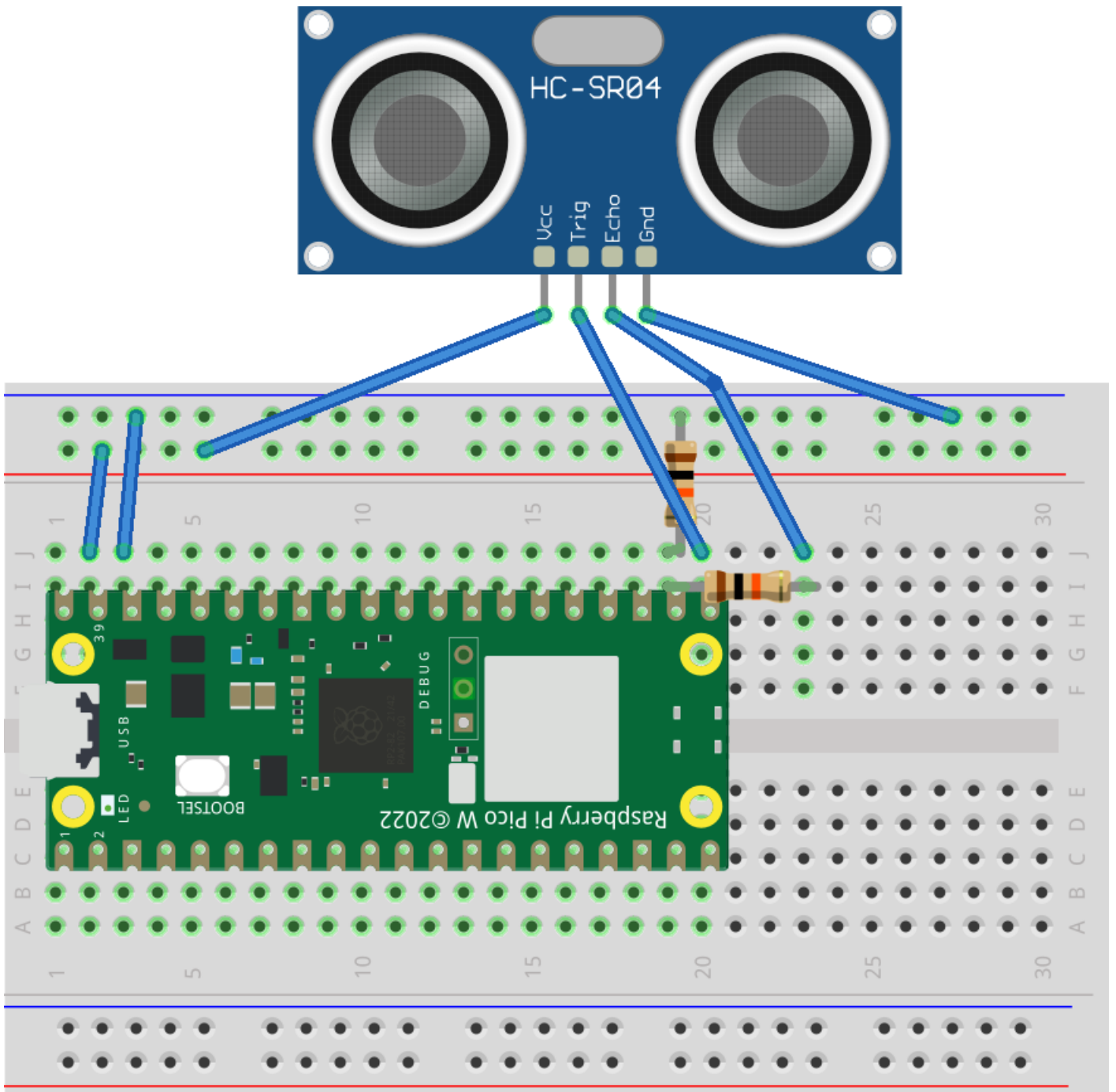
Überlegt erst einmal und dann schaut ihr euch das nächste Bauteil an:

[Die H-Brücke](#)

# Der Ultraschallsensor



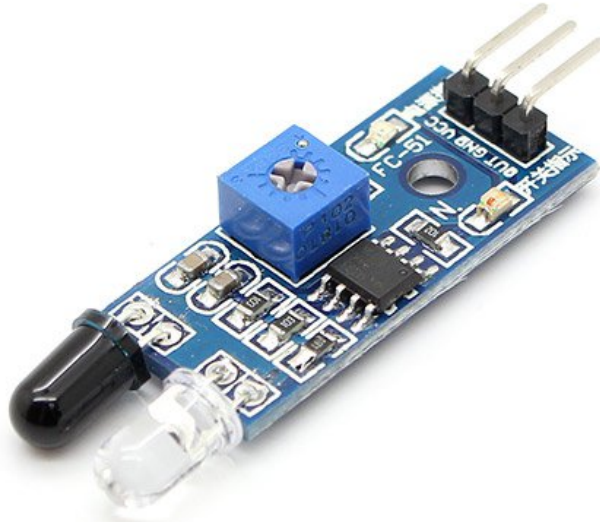
Schließt den Ultraschallsensor an den Pico an:



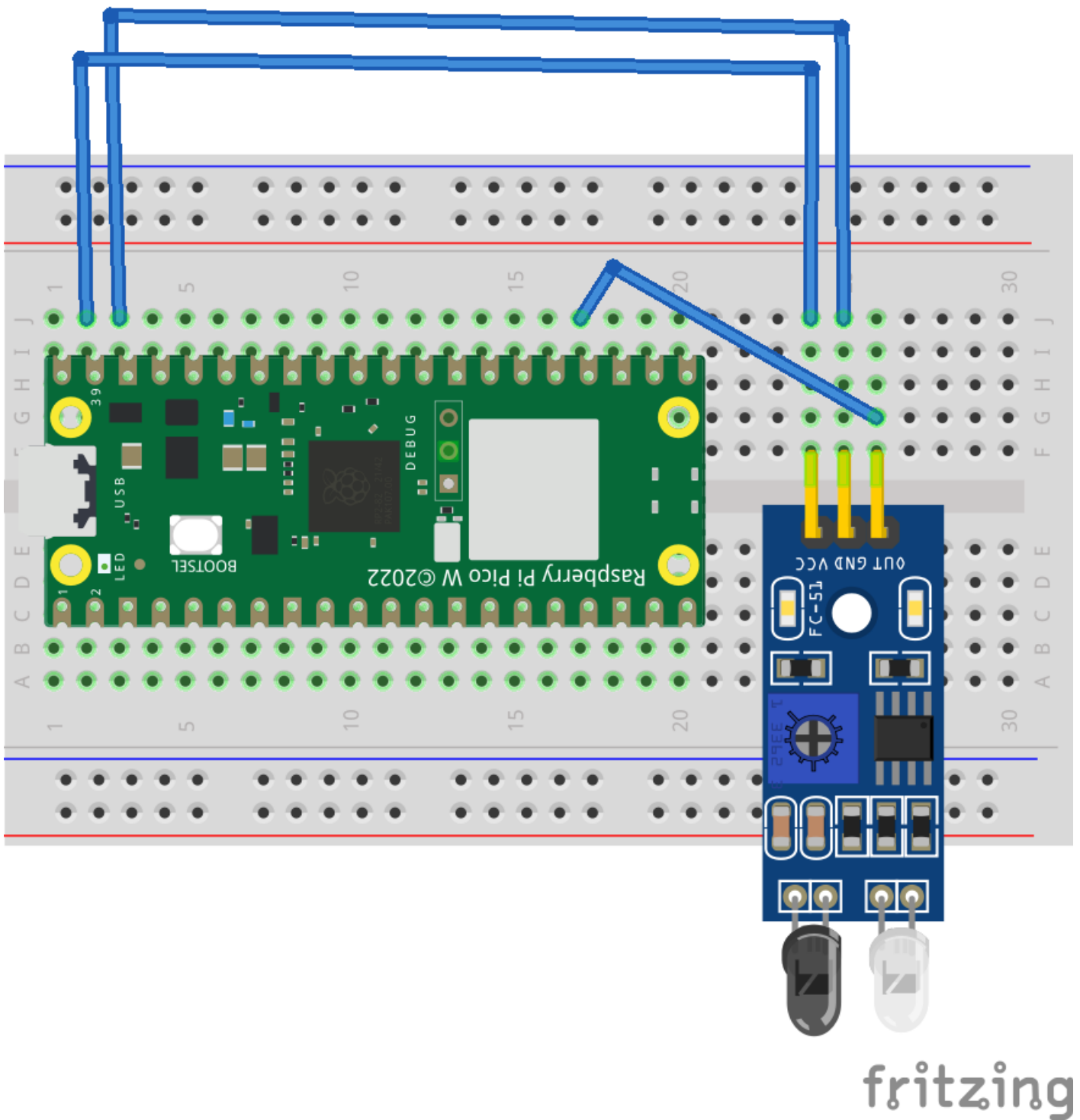
fritzing

[Die Softwarebibliothek für den Ultraschallsensor](#)

# Der Infrarotsensor



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



So wird der Infrarotsensor ausgelesen:

```
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.
```

# Codebeispiele

## Theo III

### Best Practise

#### Ausschalten der Motoren bei Unterbrechung des Programms

Anfangs wird man sehr viel an dem Roboterprogramm testen müssen. Dabei wird das Programm dann häufig abstürzen. Damit die Motoren nicht weiter in dem Zustand laufen, in dem sie dabei geschaltet waren, kann man mit einem `try/except` arbeiten:

```
try:
    # Hier läuft das Programm
except Exception as err:
    print(err) # Nötig, um Fehlermeldungen angezeigt zu bekommen.
    r.emergency_stop() # Roboter anhalten, hier ein Beispiel mit der robotlibrary.
    print("Robot stopped") # Damit es ganz deutlich ist.
except KeyboardInterrupt:
    r.emergency_stop()
    print("Keyboard interrupt")
```

#### Effizienter Code

Auch wenn die Rechenleistung der Picos ausreichen sollte, ergibt es Sinn, sich über Effizienz Gedanken zu machen, da schwer zu erkennen ist, ob manche Probleme durch Überlastung des Prozessors hervorgerufen werden.

```
while True:
    robot.drive()
    if us.get_dist() > min_distance:
        # stop or turn
        robot.stop()
```

In diesem Beispiel wird in der Schleife der Befehl `drive()` mit jedem Durchlauf aufgerufen, was nicht sonderlich effizient ist, da die Motoren weiterfahren, auch wenn das Programm gerade andere Befehle ausführt. Eine bessere Variante wäre diese:

```
robot.drive()
while us.get_dist() > min_distance:
    pass
robot.turn()
```

Hier wird nur die Entfernung zum nächsten Hindernis überprüft. Sobald der Roboter zu nahe gekommen ist, wird die Schleife beendet und der Code wird weiter ausgeführt.

## Fehlertoleranter Code

Die Sensoren, die wir benutzen, liefern nicht immer zuverlässige und korrekte Ergebnisse. Daher kann man sich nicht darauf verlassen, dass **eine** Messung ausreicht. Ist man auf genauere Ergebnisse angewiesen, kann es sinnvoll sein, die Ergebnisse von Sensormessungen (insbesondere des Ultraschallsensors) zu filtern. Dazu kann gehören, Extremwerte, die im vorliegenden Fall unwahrscheinlich sind, zu ignorieren oder Mittelwerte von mehreren Messungen zu bilden. Ein Beispiel für die Glättung der Entfernungswerte aus dem Ultraschallsensor:

```
us = Ultra(16)
dist_values = deque([0,0,0,0,0],5)
while True:
    d = us.get_dist()
    dist_values.append(d)
    d = sum(dist_values)/len(dist_values)
    print(f"Entfernung: {d} cm")
```

Eine andere Methode muss für das Auslesen des Infrarotsensors gefunden werden. Hier könnte man z.B. eine kurze Wartezeit einbauen und dann abfragen, ob der Sensor immer noch denselben Wert liefert wie bei Auslösung der Reaktion.

## Überschreiben von Methoden aus der Bibliothek

Möchte man die Funktion einer Methode aus der Roboterbibliothek (robotlibrary) verändern, kann man die Methode überschreiben (Fachbegriff aus der objektorientierten Programmierung). Dafür wird die Klasse `Robot` vererbt, wie in dem Codebeispiel angegeben. Möchte man nun z.B. die Methode `set_speed()` verändern, dann definiert man sie einfach neu. Ist eine Methode nicht in der abgeleiteten Klasse (in diesem Fall `MyRobot`) definiert, dann wird die Methode der Elternklasse (`Robot`) genommen. Probiere es aus, indem du das vorliegende Programm einmal mit und einmal ohne die Definition von `set_speed()` aufrufst.

```
from robotlibrary.robot import Robot
from time import sleep, sleep_ms

class MyRobot(Robot):
```

```

def __init__(self):
    super().__init__(False, None)
    print("start")

def set_speed(self, x):
    print(f"Child method set_speed. Value: {x}")

def main():
    try:
        robot = MyRobot()
        robot.set_speed(90)
        while True:
            sleep(1)
    except KeyboardInterrupt:
        robot.emergency_stop()

if __name__ == "__main__":
    # execute only if run as a script
    main()

```

## Verändern der Funktion der Fernbedienung

Eine Ableitung funktioniert nicht, wenn man die Methode `read()` verändern möchte, da von der Bibliothek immer die Elternklasse aufgerufen würde. Man kann aber vor der Initialisierung der Klasse `Robot` eine neue Methode definieren und diese als Parameter übergeben. Hier ist ein Beispiel, wie es geht.

```

# Example for how to override the read() method that gets input from the remote control.
# Define your own method.
def my_read(buffer: memoryview):
    print("my_read called.")

def main():
    try:
        # Use the name of your method as a parameter when initialising the robot object.
        r = Robot(True, my_read)
        r.set_speed(100)
        while True:
            sleep(1)
    except KeyboardInterrupt:

```

```

        r.emergency_stop()

if __name__ == "__main__":
    # execute only if run as a script
    main()

```

## Den Ultraschallsensor im Hintergrund laufen lassen

In komplexeren Programmen kann es lästig werden, immer wieder die Entfernung zum nächsten Hindernis zu überprüfen. Dieses Beispiel erläutert, wie man das Problem löst, indem man die Entfernungsmessung in den Hintergrund verlegt.

```

from robotlibrary.robot import Robot
from time import sleep, sleep_ms
import uasyncio as asyncio

##### Your class definition
class MyRobot(Robot):
    def __init__(self):
        super().__init__(False, None) # Call the original constructor.
        print("start")

    # With this method defined here, the robot will not drive as the speed is not set in this
    function.
    def set_speed(self, x):
        print(f"Child method set_speed. Value: {x}")

##### End of class definition
# Define functions for your program
async def monitor_dist():
    global distance
    while True:
        if robot.get_dist() < distance:
            react_to_obstacle()
            await asyncio.sleep_ms(100)

def react_to_obstacle():
    global distance
    print("danger")

```

```

robot.random_spin(300)
robot.set_forward(True)
robot.set_speed(80)

async def my_program():
    robot.set_speed(90)
    while True:
        print("driving")
        await asyncio.sleep_ms(100)

async def main():
    asyncio.create_task(monitor_dist())
    await my_program()

##### Initialize the robot and start the program.
robot = MyRobot()
distance = 20
if __name__ == "__main__":
    # execute only if run as a script
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("Emergency stop")
        robot.emergency_stop()

```

## Beschleunigung mit Entfernungsmessung

Diese Methode ist nur nötig, wenn man nicht asyncio benutzt. Beschleunigt man den Roboter langsam mit der Methode `set_speed`, dann kann er in der Zeit bis zum Erreichen der Geschwindigkeit keine Entfernungsmessung durchführen. Dies ist ein Beispiel, wie man beides erreichen kann:

```

obstacle_detected = False
new_speed = 100
speed_now = 0
min_distance = 15
while speed_now <= new_speed and not obstacle_detected:
    #Set the speed for the motors, f. ex. motor.set_speed(speed_now)

```

```
    utime.sleep_ms(10+int(speed_now/2))
    speed_now += 1
    if us.get_dist() < min_distance: # Adjust the code to your needs.
        obstacle_detected = True
if obstacle_detected:
    # Stop or turn or whatever
    obstacle_detected = False
else:
    # keep going
    pass
```