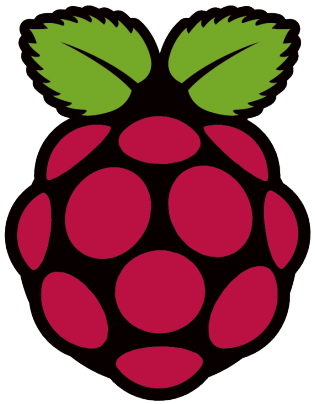
Programmieren mit dem Raspberry Pi



https://www.raspberrypi.com

Organisatorisches

Inhalt des Workshops

Tag 1:

- Einführung & Inbetriebnahme
- kurze Einführung in Python
- GPIO-Pins (Ampel)

Tag 2:

- Entfernungsmessung über Ultraschall (Blitzer)
- Gas-Sensor (Abgasmessung)

Ausblick: Forscher-AG-Projekt (Autonomes Fahren)

Der Raspberry Pi

Der Raspberry Pi

- Einplatinenrechner
- ca. 40 Euro
- Linux-Betriebssystem
- GPIO-Anschlüsse für Sensoren und Aktoren

Aufbau

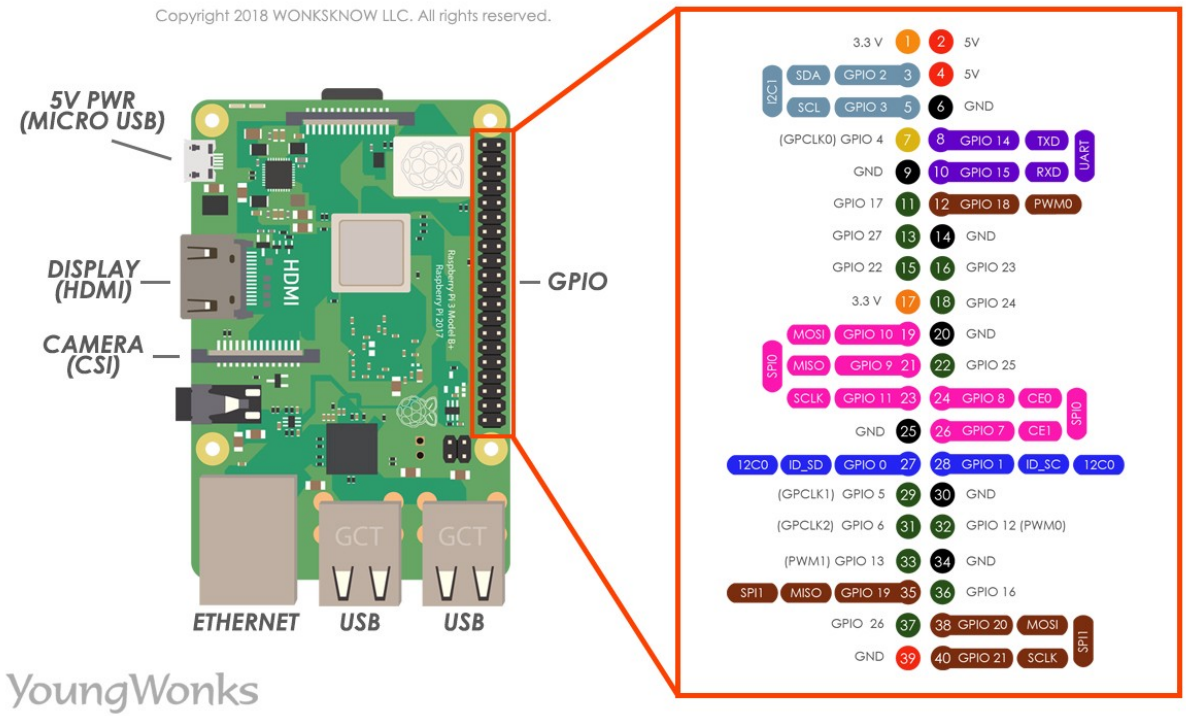
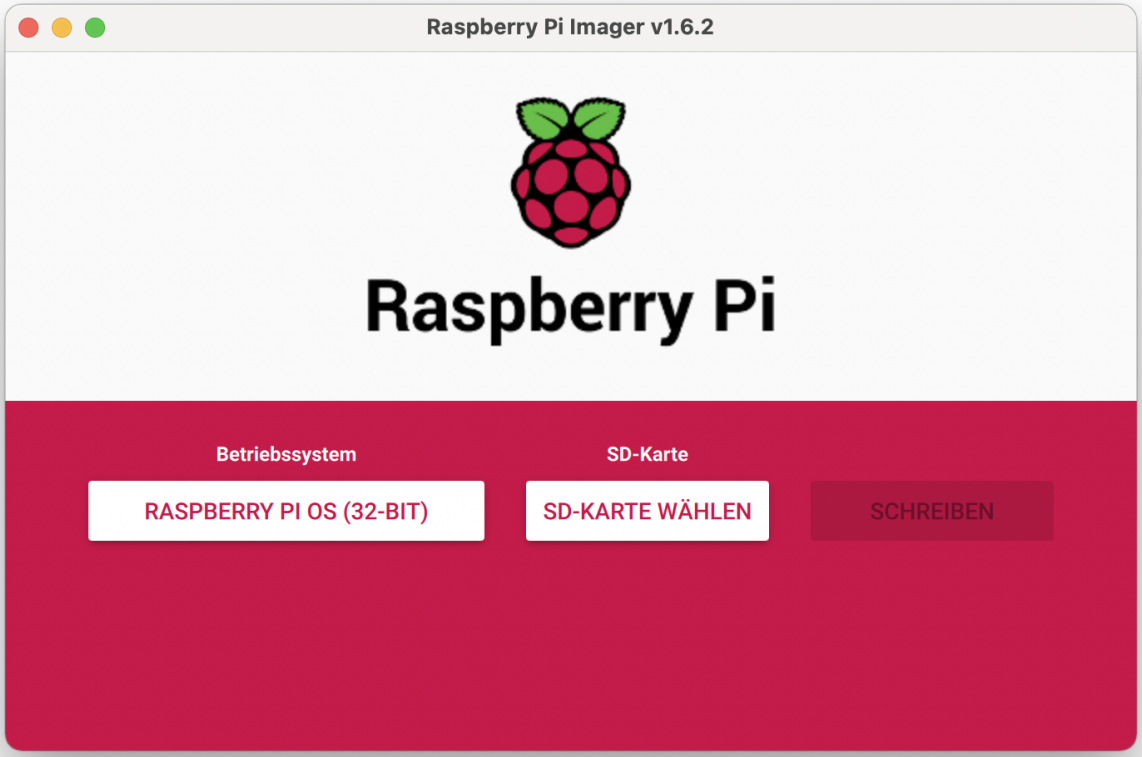


Abbildung: medium.com

SD-Karte vorbereiten I



Download: https://www.raspberrypi.com/software/

SD-Karte vorbereiten II

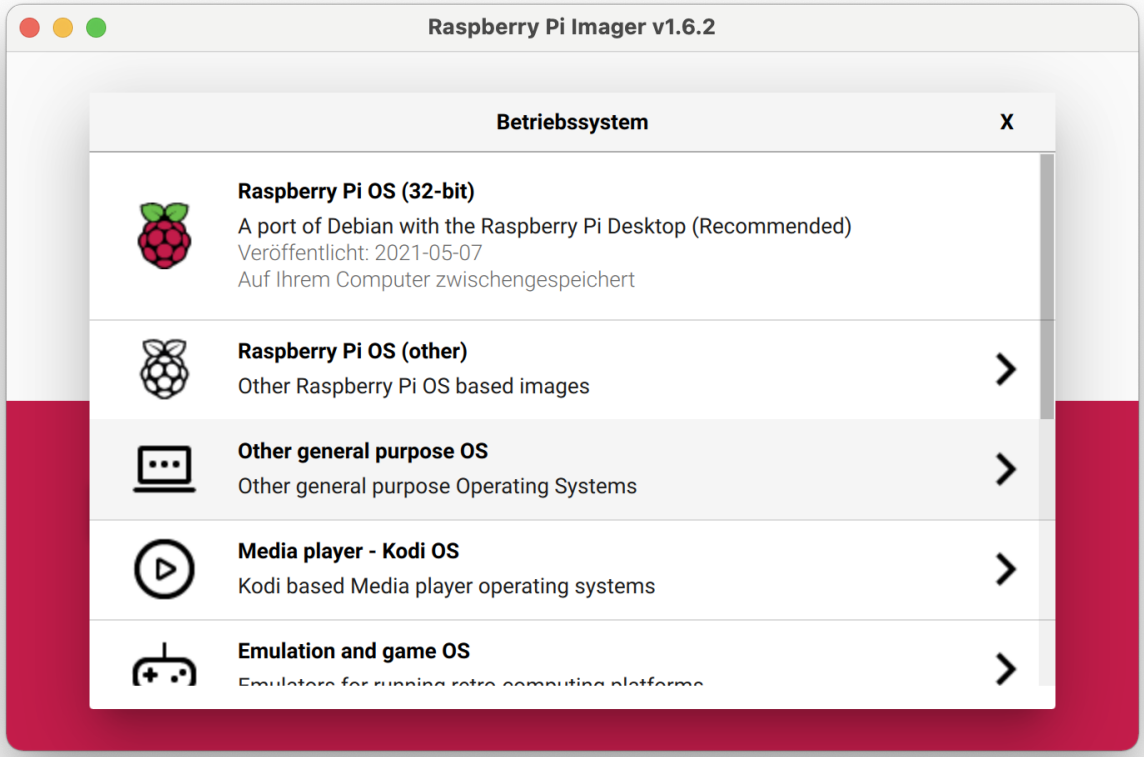


Abbildung: 1. Betriebssystem auswählen („Raspberry Pi OS (32-bit)“)

SD-Karte vorbereiten III

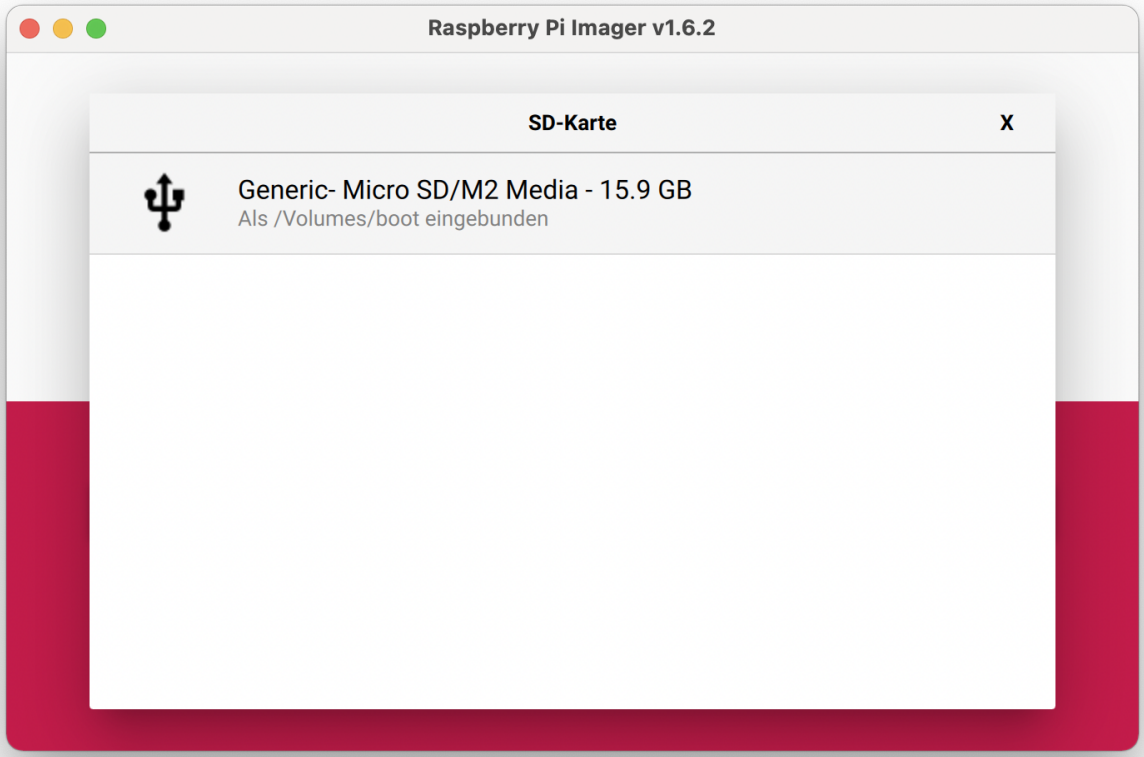


Abbildung: 2. SD-Karte auswählen

SD-Karte vorbereiten IV

Abbildung: 3. Image auf SD-Karte schreiben

Raspberry Pi OS – Der Desktop

Abbildung: Desktop von Raspberry Pi OS

Raspberry Pi einrichten – Konfiguration

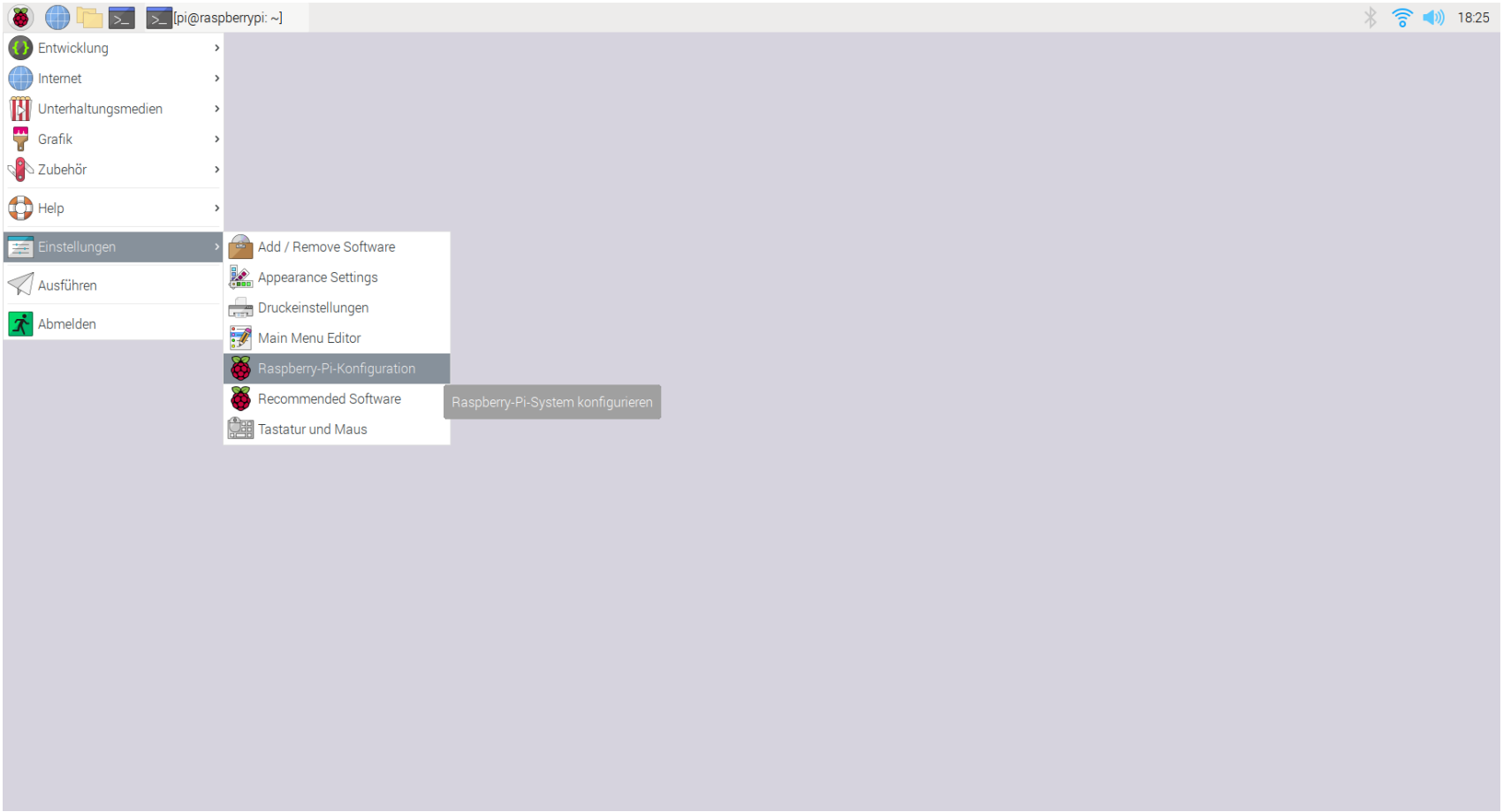


Abbildung: Raspberry-Pi-Konfiguration

Raspberry Pi einrichten – Sprache & Tastaturlayout

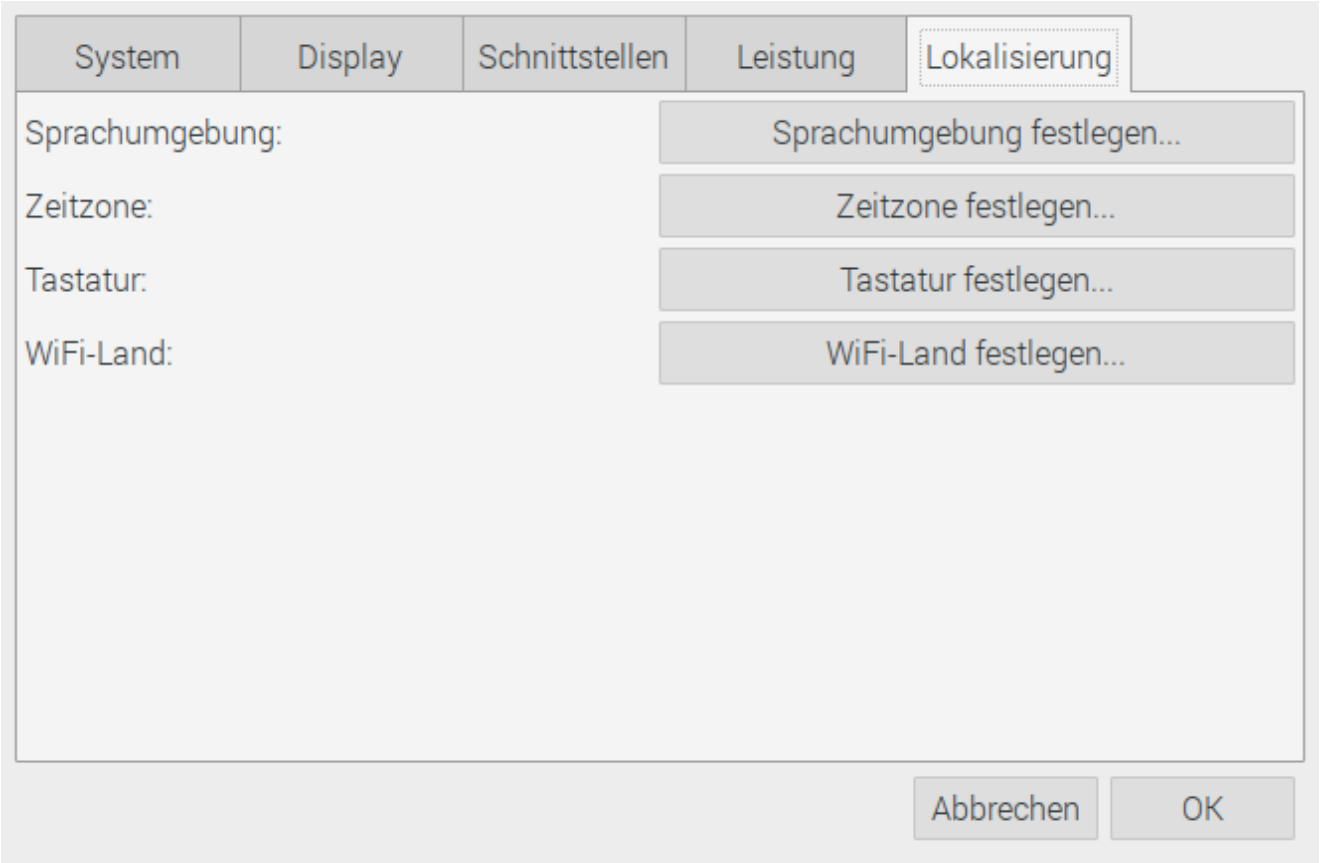


Abbildung: Lokalisierungsoptionen

Raspberry Pi einrichten – Schnittstellen

Abbildung: Schnittstellen

Kurze Einführung in Python

Thonny I

Abbildung: Thonny starten

Thonny II

Abbildung: Der Editor Thonny

Hallo Welt!

```
print("Hello World")
```

- `print` -Befehl zum Anzeigen von Text
- Text (*String*) in Anführungszeichen

Arbeiten mit Variablen

```
a = 3
b = 42
c = 11
mein_text = "Hallo Welt"

print(b)
print(a+b*10)
print(mein_text)
print(mein_text + "!")
```

- Speichern von Text und Zahlen in Variablen:
  `<Variablenname> = <Inhalt>`
- Abrufen des Inhalts: `<Variablenname>`
- Zusammenfügen von Strings: `+`
- Rechnen mit Zahlen: `+`, `-`, `*`, `/`, `//`

Kommentare

```
print("Hallo Welt") # Hier wird etwas ausgegeben
```

- Text nach einem # wird nicht zum Code gezählt
- Zur Dokumentation des Codes (wichtig!!!)

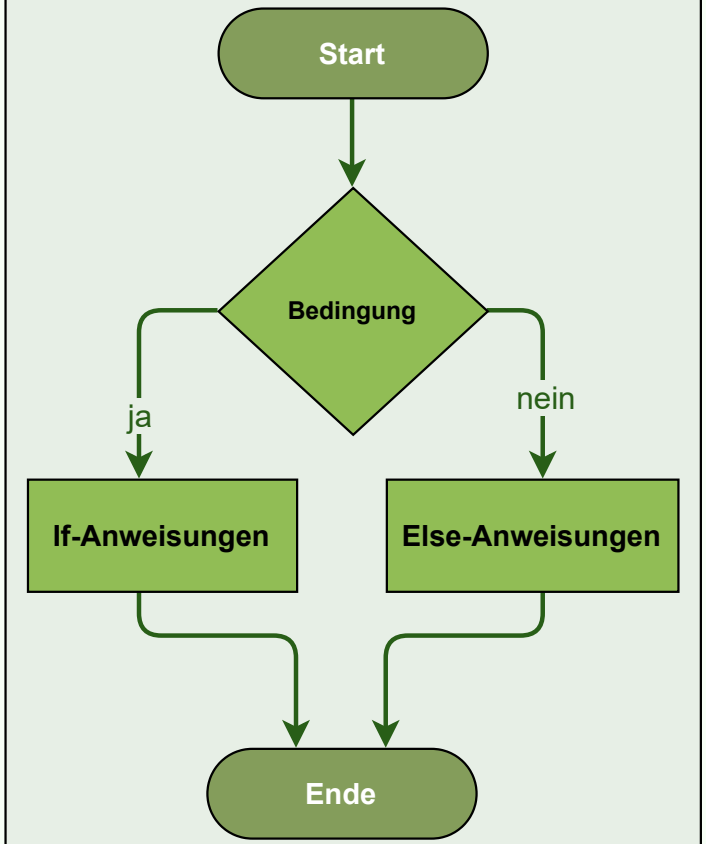
if-Abfrage

```
i = 7
if i < 5:
    print("i kleiner als 5")
elif i == 5:
    print("i gleich 5")
else:
    print("i groesser als
     ↪  5")
```

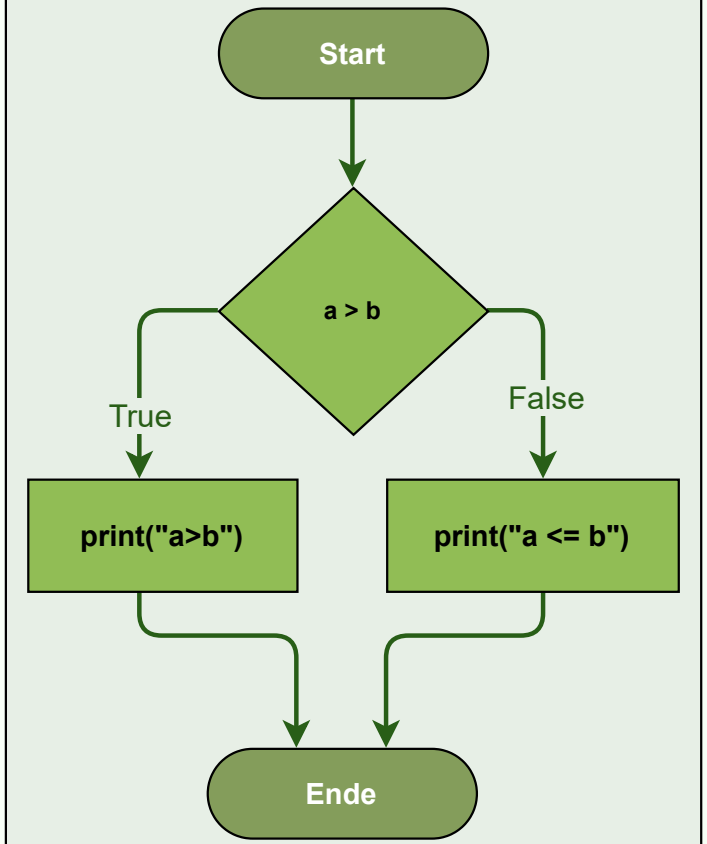
- Befehle nur ausführen, wenn *if*-Bedingung gilt
- Weitere Bedingung(en) mit *elif* (optional)
- Wenn keine Bedingung zutrifft: *else*-Fall ausführen (optional)
- Vergleich mit ==, !=, >, <, >=, <=
- Inhalt wird eingerückt (Tab)

If-Abfrage

Allgemein



Beispiel



Bei allen Diagrammen wurde das Deklarieren einiger Variablen aus Gründen der Übersicht weggelassen.

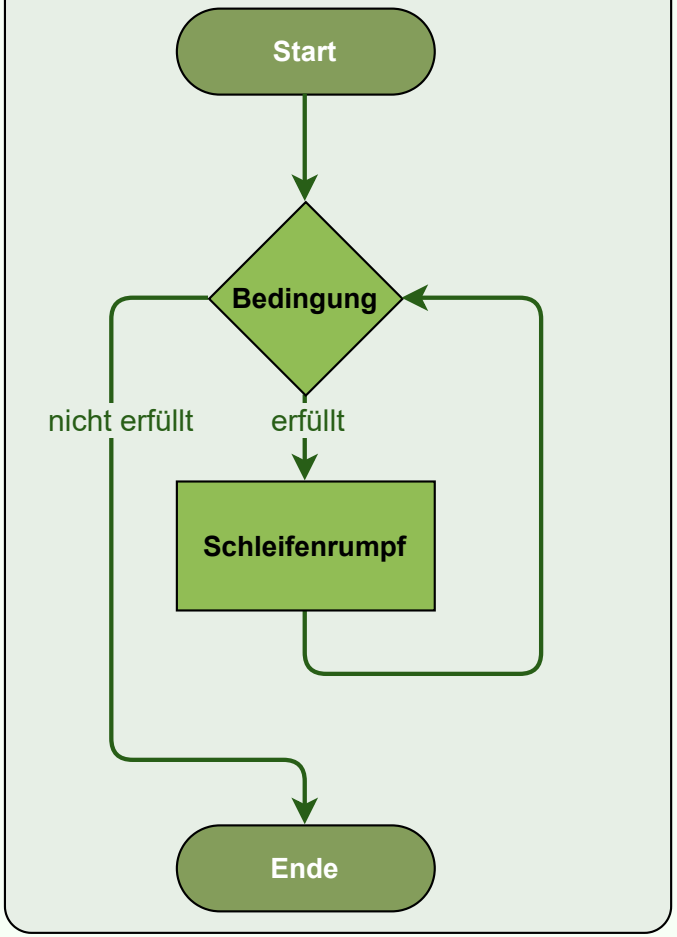
while-Schleife

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

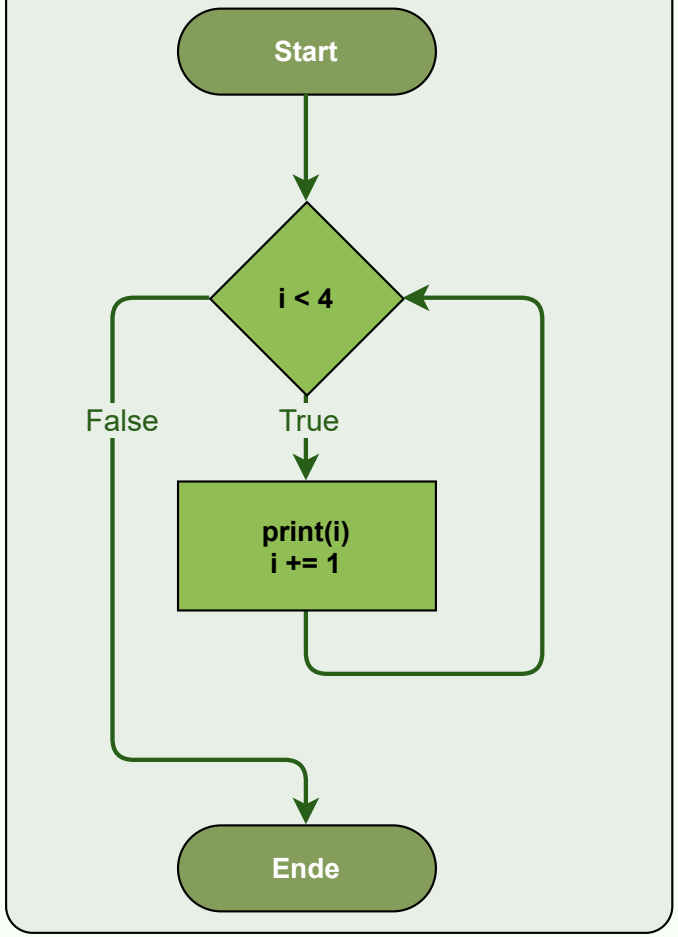
- Befehle wiederholt ausführen, solange die Bedingung gilt
- Inhalt wird eingerückt (Tab)

While Schleife

Allgemein



Beispiel



Bei allen Diagrammen wurde das Deklarieren einiger Variablen aus Gründen der Übersicht weggelassen.

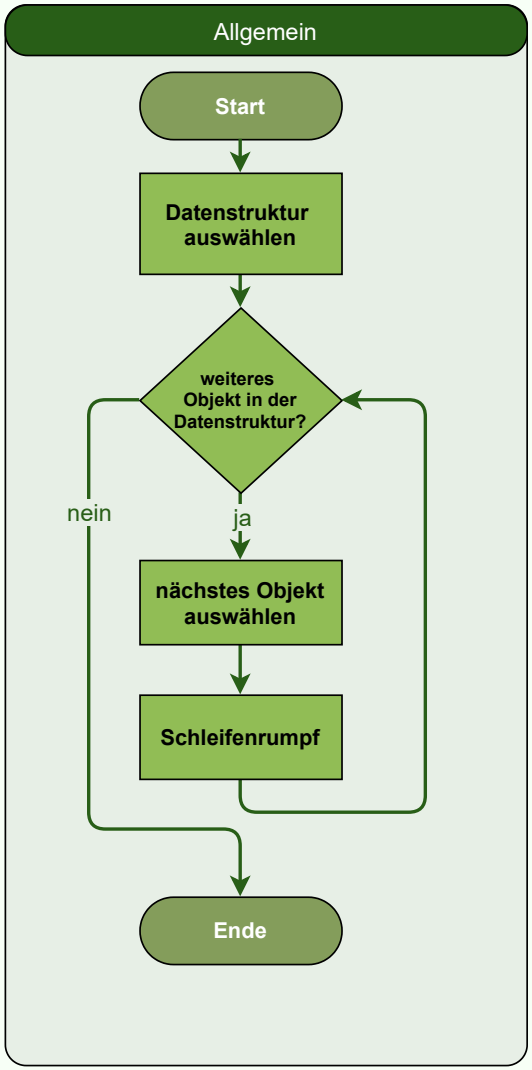
for-Schleife

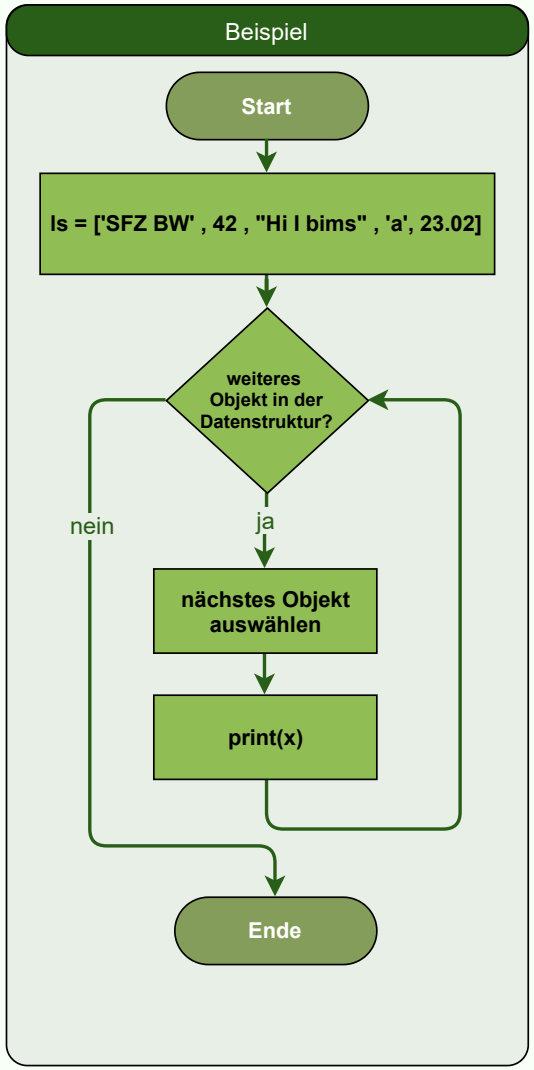
```
for i in range(10):
    print(i)

for i in range(10,20):
    print(i)
```

- Wird meist genutzt, wenn man etwas mit einer bestimmten Anzahl von Wiederholungen ausführen möchte
- In Python: Durchlaufen einer Liste
- Inhalt wird eingerückt (Tab)

For-Schleife





Bei allen Diagrammen wurde das Deklarieren einiger Variablen aus Gründen der Übersicht weggelassen.

sleep

```
# Bibliothek importieren
from time import sleep

print("Anweisung 1")
sleep(3) # 3 Sekunden warten
print("Anweisung 2")
```

- Programm für eine bestimmte Zeit „schlafen legen“
- Zeitangabe in Sekunden
- Bibliothek muss zuerst importiert werden

Stromkreis

Grundlagen von Stromkreisen:

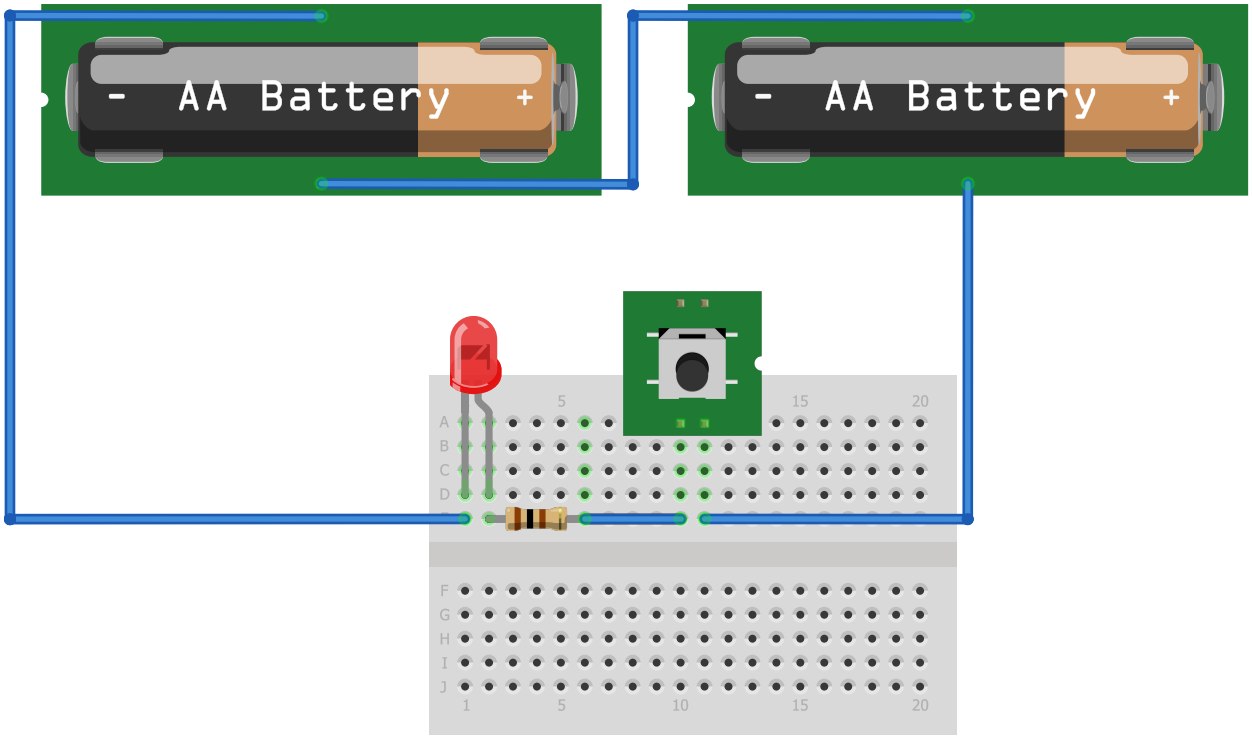


Abbildung: Beispielstromkreis

Grundlagen von Stromkreisen:

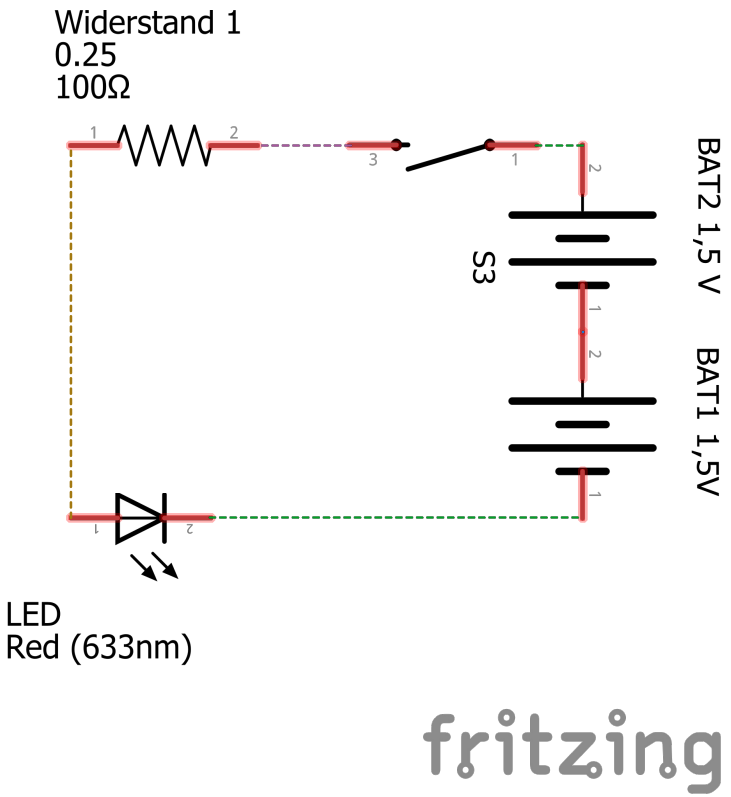


Abbildung: Beispielstromkreis

Vorwiderstand berechnen

Grundlagen von Stromkreisen:

Spannung die am Vorwiderstand abfallen soll:

$$U_{\text{Quelle}} = U_{\text{R}} + U_{\text{Diode}} \Longrightarrow U_{\text{R}} = U_{\text{Quelle}} - U_{\text{Diode}}$$

Außerdem folgt aus dem Datenblatt der Diode $I_{\text{Diode,max}}$ und mit $I_{Diode,max} = I_{R,max}$:

$$R = \frac{U_{\text{R}}}{I_{R,max}} = \frac{U_{\text{R}}}{I_{Diode,max}}$$

Widerstands-Farbcode

Widerstandsfarbcode mit 4 Ringen				
Farbe	**1. Ring (1. Ziffer)**	**2. Ring (2. Ziffer)**	**3. Ring (Multiplikator)**	**4. Ring (Toleranz)**
schwarz	-	0	1	-
braun	1	1	10	+/- 1%
rot	2	2	100	+/- 2%
orange	3	3	1'000	-
gelb	4	4	10'000	-
grün	5	5	100'000	-
blau	6	6	1'000'000	-
violett	7	7	-	-
grau	8	8	-	-
weiss	9	9	-	-
gold	-	-	0.1	+/- 5%
silber	-	-	0.01	+/- 10%

Abbildung: electronicsplanet.ch

GPIO-Pins

Pinout I

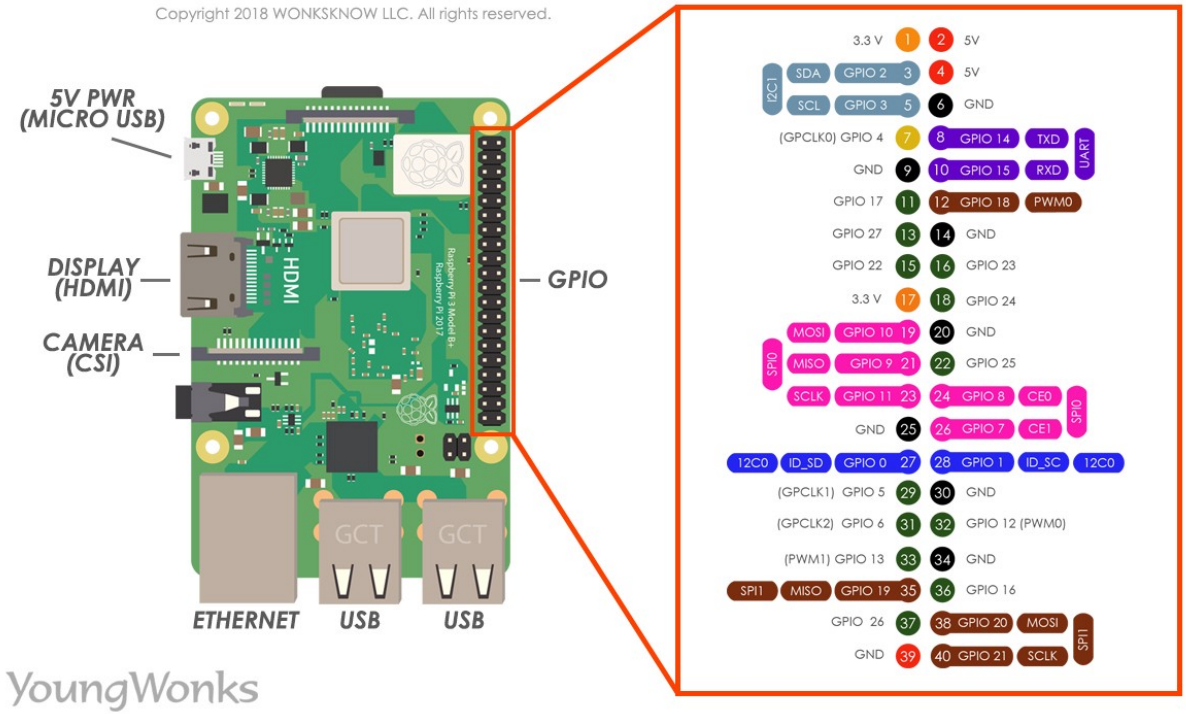


Abbildung: medium.com

Was fällt bei der Nummerierung der Pins auf?

Pinout II

Es gibt 2 Nummerierungen:

- eine Nummerierung von 0 - 40 (innen, **GPIO.BOARD**)
- GPIO-Nummerierung (außen, **GPIO.BCM**)

Blinkende LED – Aufbau

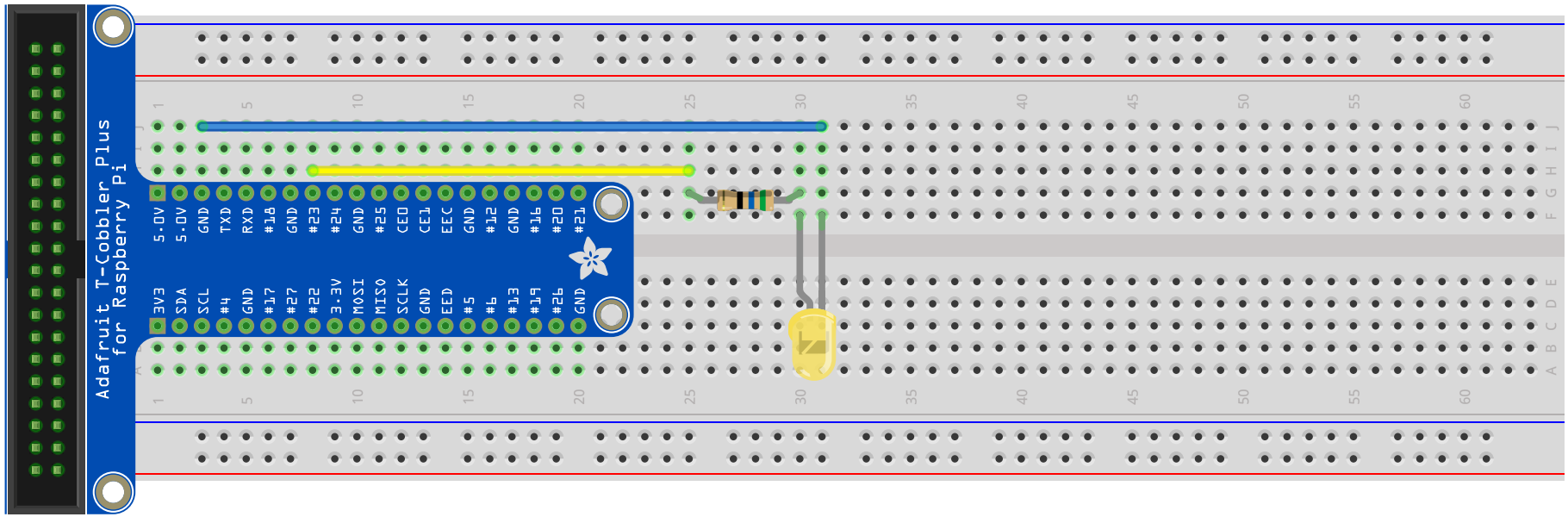


Abbildung: Aufbau der Blinkerschaltung

Blinkende LED – Pegeldiagramm

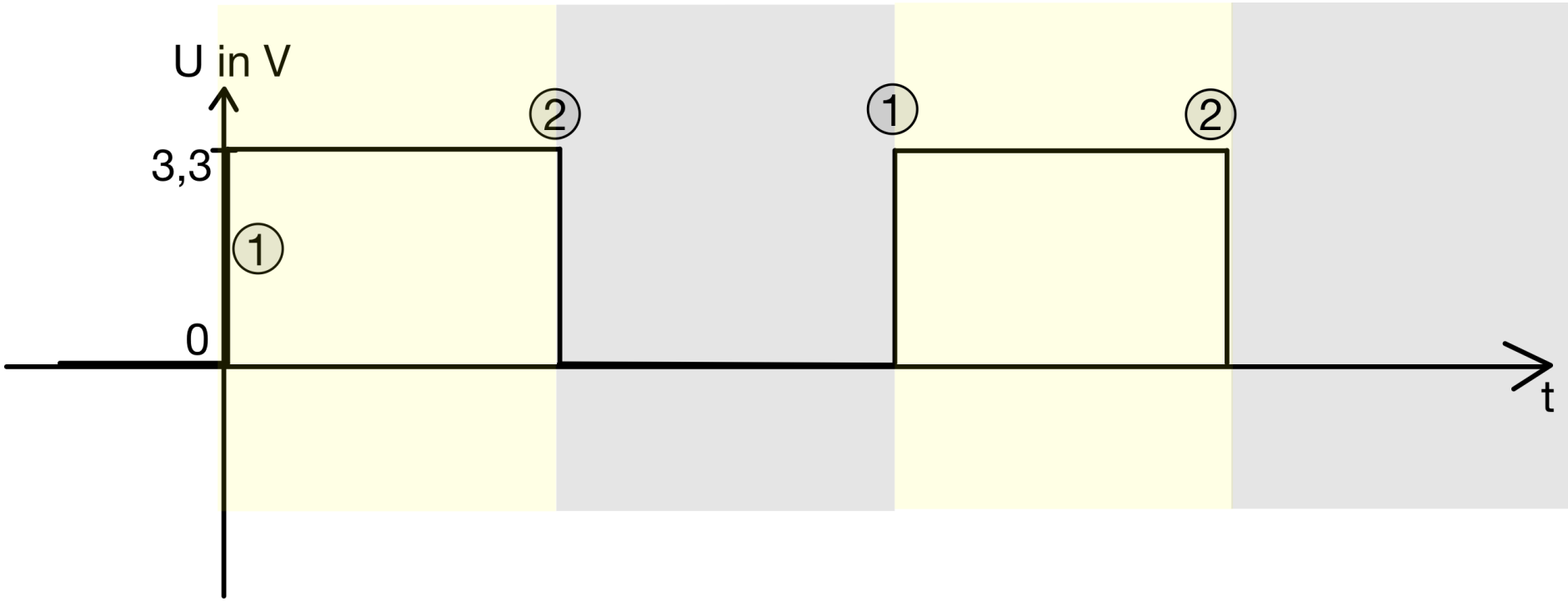


Abbildung: Pegel der Blinkerschaltung

Blinkende LED – Programmcode

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

GPIO.setup(23,GPIO.OUT)

while True:
    GPIO.output(23,True)
    sleep(1)
    GPIO.output(23,False)
    sleep(1)
```

Blinkende LED – Aufbau

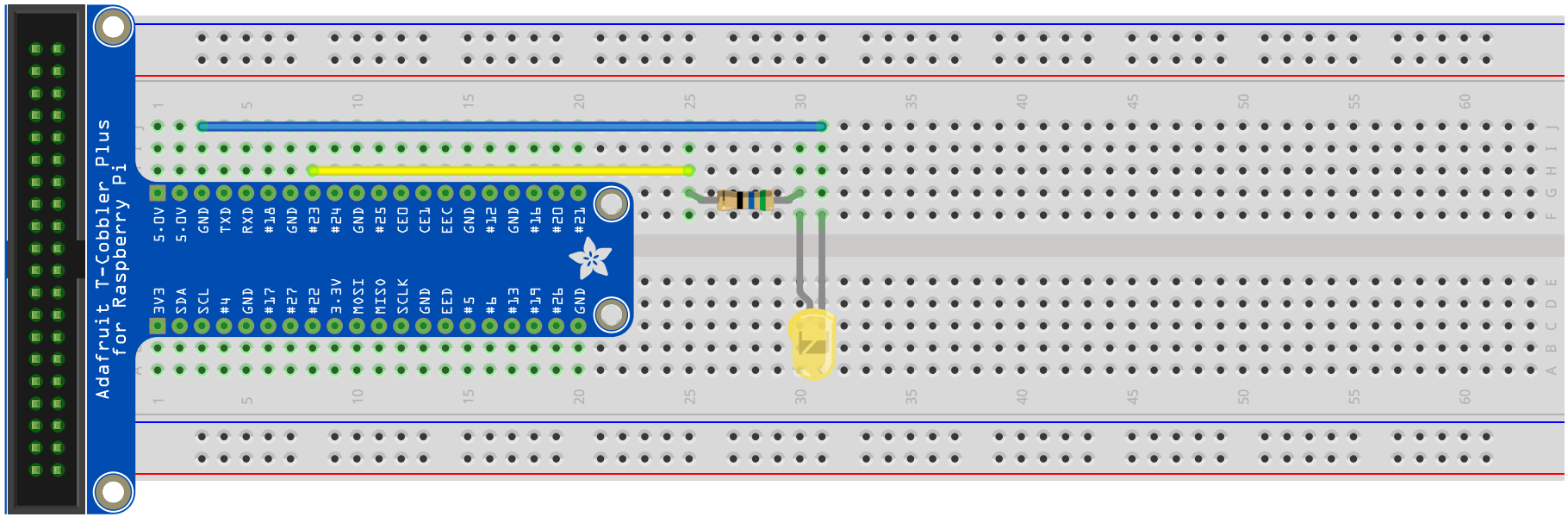


Abbildung: Aufbau der Blinkerschaltung

Ampel

Aufgabe:

Baue eine Ampelschaltung, die die folgenden Farben durchläuft:
Rot → Rot+Gelb → Grün → Gelb → Rot → usw.

Dafür werden folgende Bauteile benötigt:

- 3 LEDs (Rot, Gelb, Grün) mit Vorwiderstand (bzw. fertiges Ampel-Modul)
- Jumper-Kabel
- Breadboard

Ampel – Aufbau

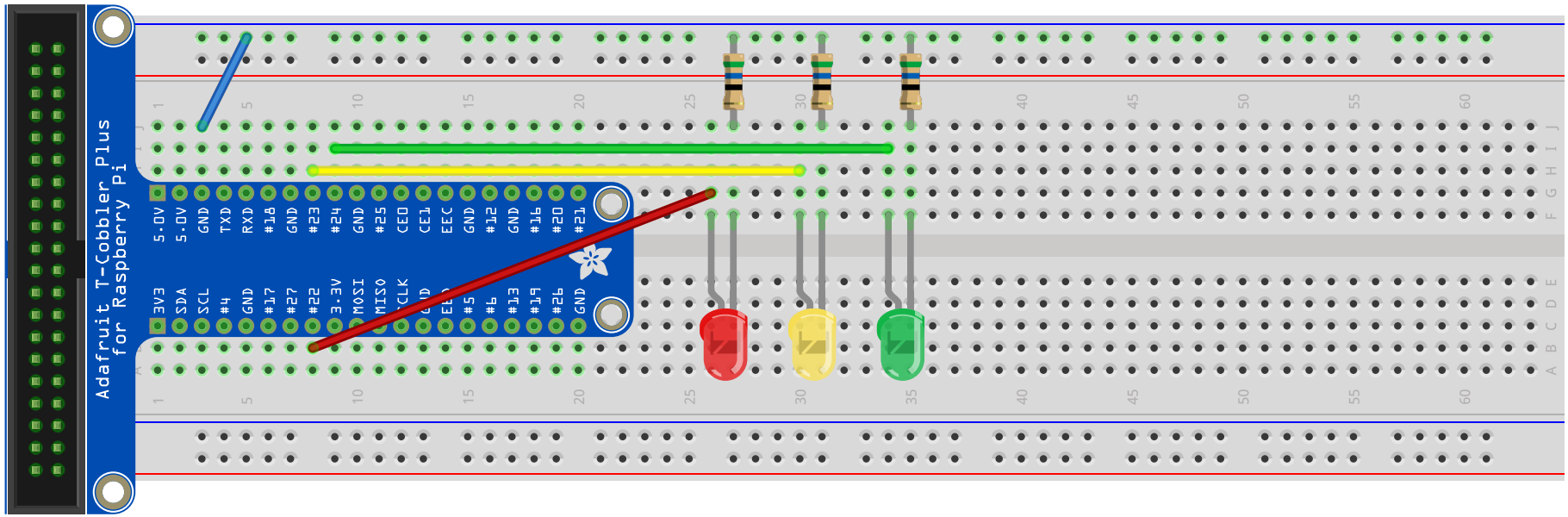


Abbildung: Aufbau der Ampelschaltung

Ampel – Programmcode I

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

rot = 22
gelb = 23
gruen = 24

GPIO.setup([rot,gelb,gruen],GPIO.OUT)
```

Ampel – Programmcode II

```
while True:
    # rot:
    GPIO.output(rot,True)
    sleep(1)
    # rot + gelb:
    GPIO.output(gelb,True)
    sleep(1)
    # gruen:
    GPIO.output([rot,gelb],False)
    GPIO.output(gruen,True)
    sleep(1)
    # gelb:
    GPIO.output(gruen,False)
    GPIO.output(gelb,True)
    sleep(1)
```

Ampel – Programmcode III

```
    # rot:
    GPIO.output(gelb,False)
    GPIO.output(rot,True)
```

Erweiterung: Fußgängerampel

Aufgabe:

Erweitere deine Ampelschaltung zu einer Fußgängerampel. Diese soll, sobald sie grün wird, auf eine Tastereingabe warten und das Programm daraufhin fortsetzen (also die Ampel auf rot schalten, kurz warten, wieder auf grün schalten, auf eine weitere Eingabe warten usw.)

Dafür werden folgende zusätzliche Bauteile benötigt:

- 2 LEDs (Rot, Grün) mit Vorwiderstand
- 1 Taster
- 1 Widerstand (10 kΩ)
- Jumper-Kabel
- Breadboard

Fußgängerampel – Aufbau

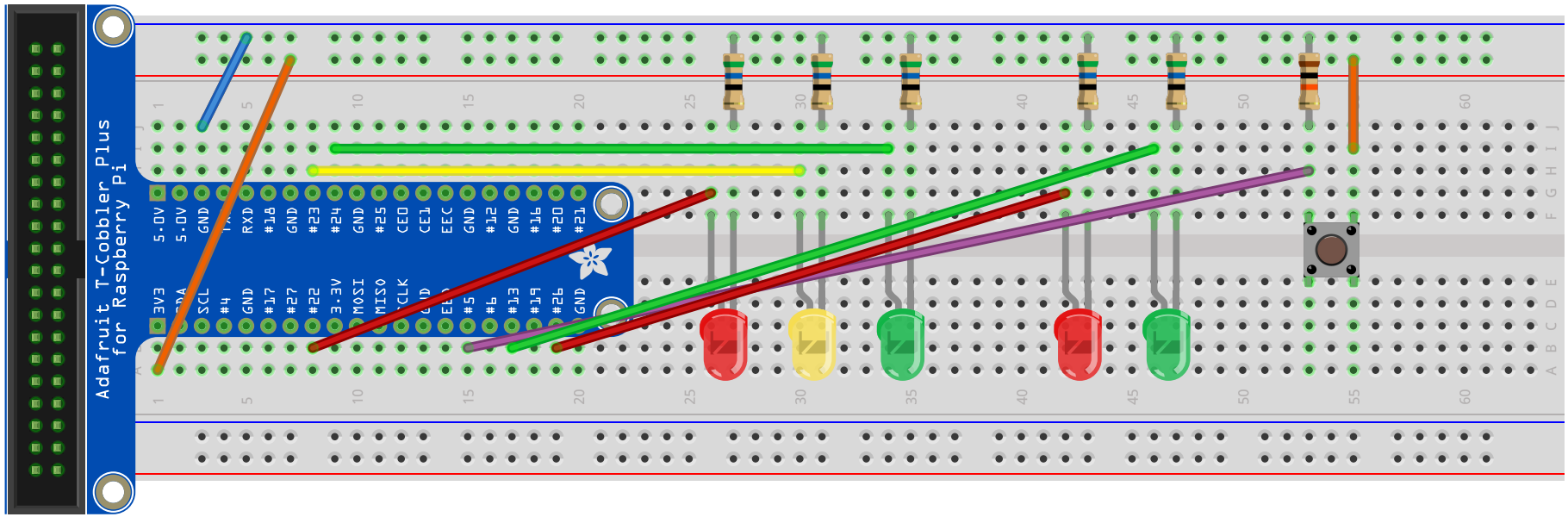


Abbildung: Aufbau der Fußgängerampel

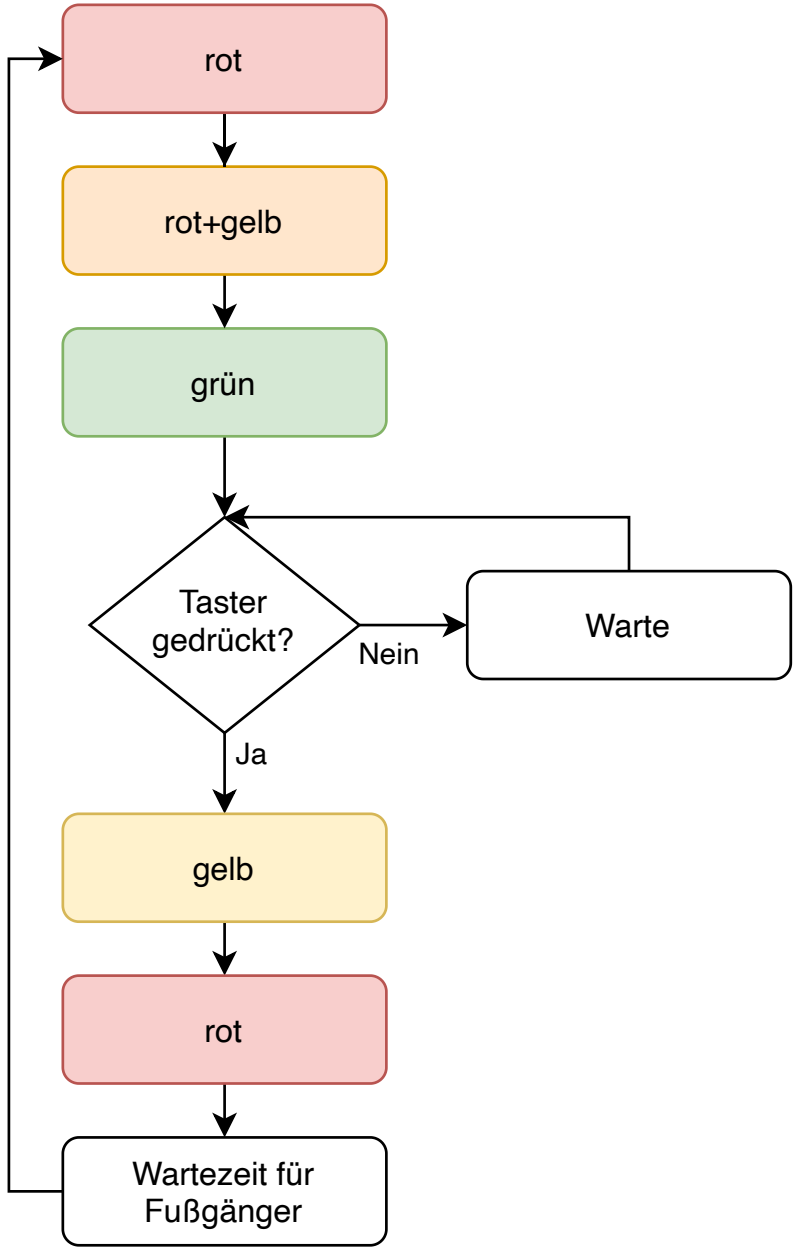


Abbildung: Programmablaufplan der Fußgängerampel

Fußgängerampel – Programmcode I

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

rot = 22
gelb = 23
gruen = 24

rot_fg = 26
gruen_fg = 13

taster = 5
```

Fußgängerampel – Programmcode II

```
GPIO.setup([rot,gelb,gruen,rot_fg,gruen_fg],GPIO.OUT)
GPIO.setup(taster,GPIO.IN)

while True:
    # rot:
    GPIO.output(rot,True)
    sleep(1)
    # rot + gelb:
    GPIO.output(gelb,True)
    sleep(1)
    # gruen:
    GPIO.output([rot,gelb],False)
    GPIO.output(gruen,True)
```

Fußgängerampel – Programmcode III

```
    # warte, bis Taster gedrueckt wird:
    while(GPIO.input(taster) == False):
        sleep(0.2)

    # gelb:
    GPIO.output(gruen,False)
    GPIO.output(gelb,True)
    sleep(1)
    # rot:
    GPIO.output(gelb,False)
    GPIO.output(rot,True)
    sleep(5) # Wartezeit fuer Fussgaenger
```

Entfernungsmessung über Ultraschall (Blitzer)

Entfernungsmessung – Was benötigen wir?

- Ultraschallsensor (HC-SR04)
- Logik-Level-Shifter
  → *alternativ:* Widerstände (330 Ω, 10 kΩ)
- Jumper-Kabel
- Breadboard

Entfernungsmessung – Prinzip

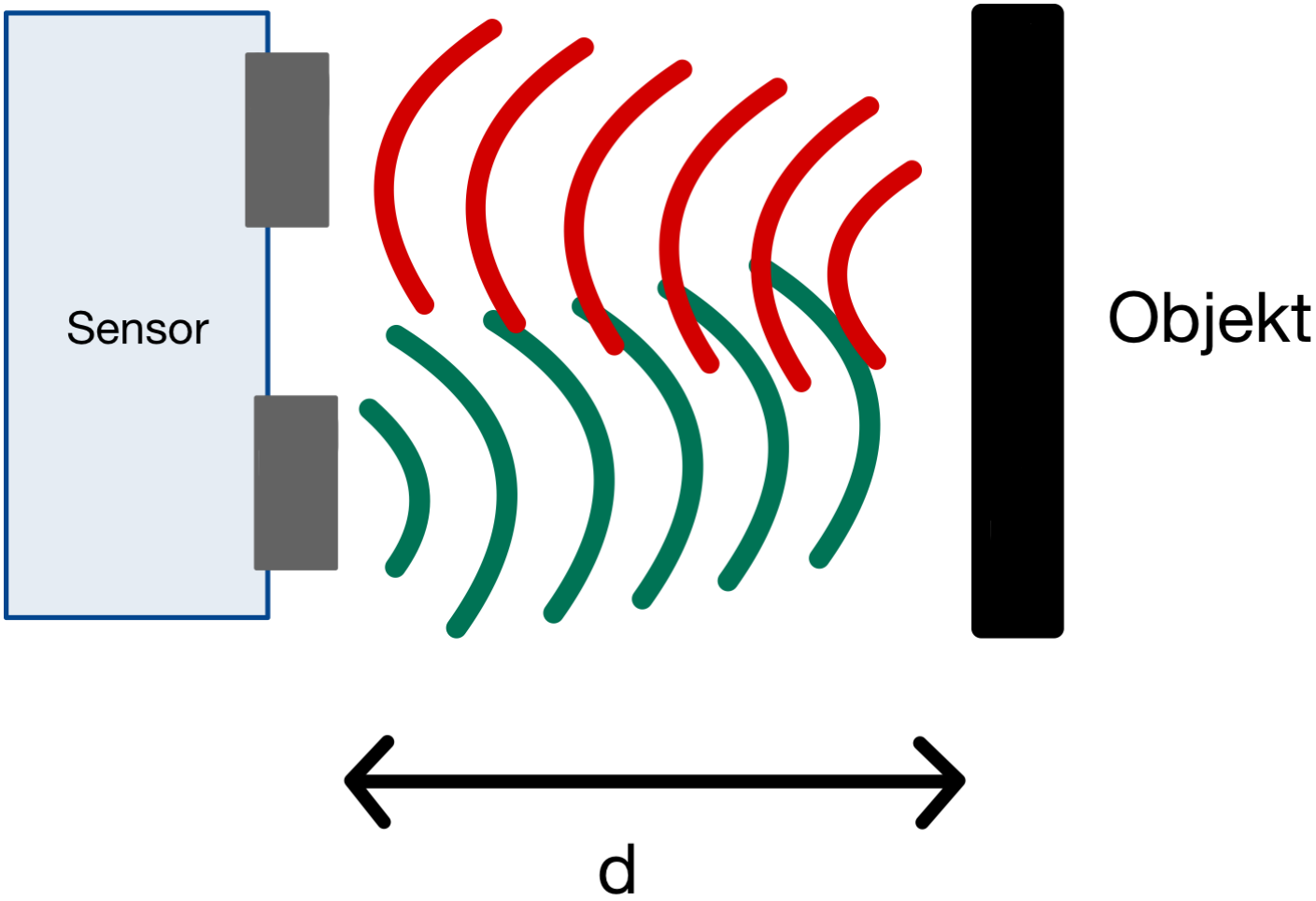


Abbildung: Ultraschall-Entfernungsmessung

Entfernungsmessung – Aufbau

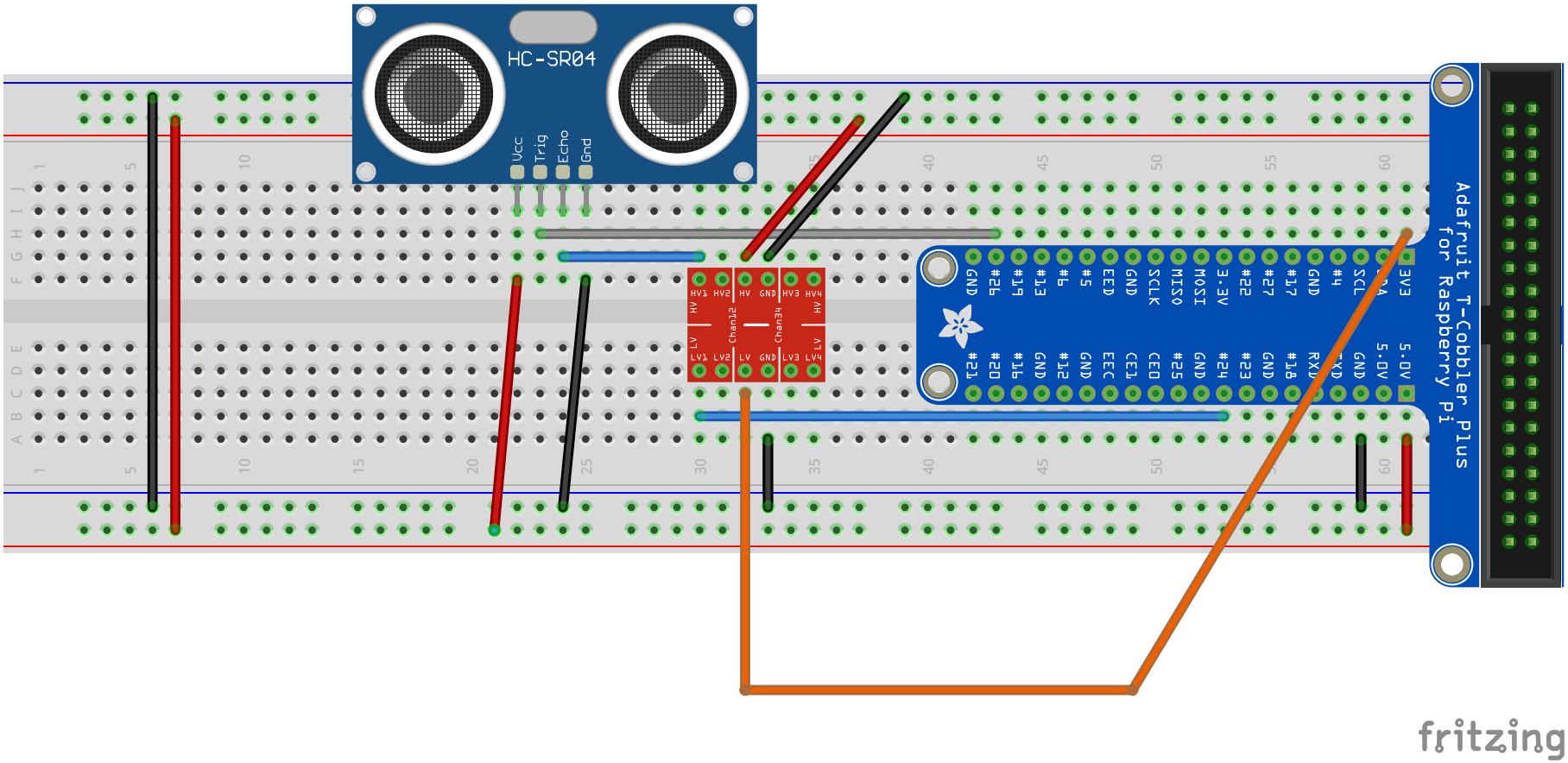


Abbildung: Aufbau des Entfernungsmessers

Entfernungsmessung – Aufbau (alternativ)

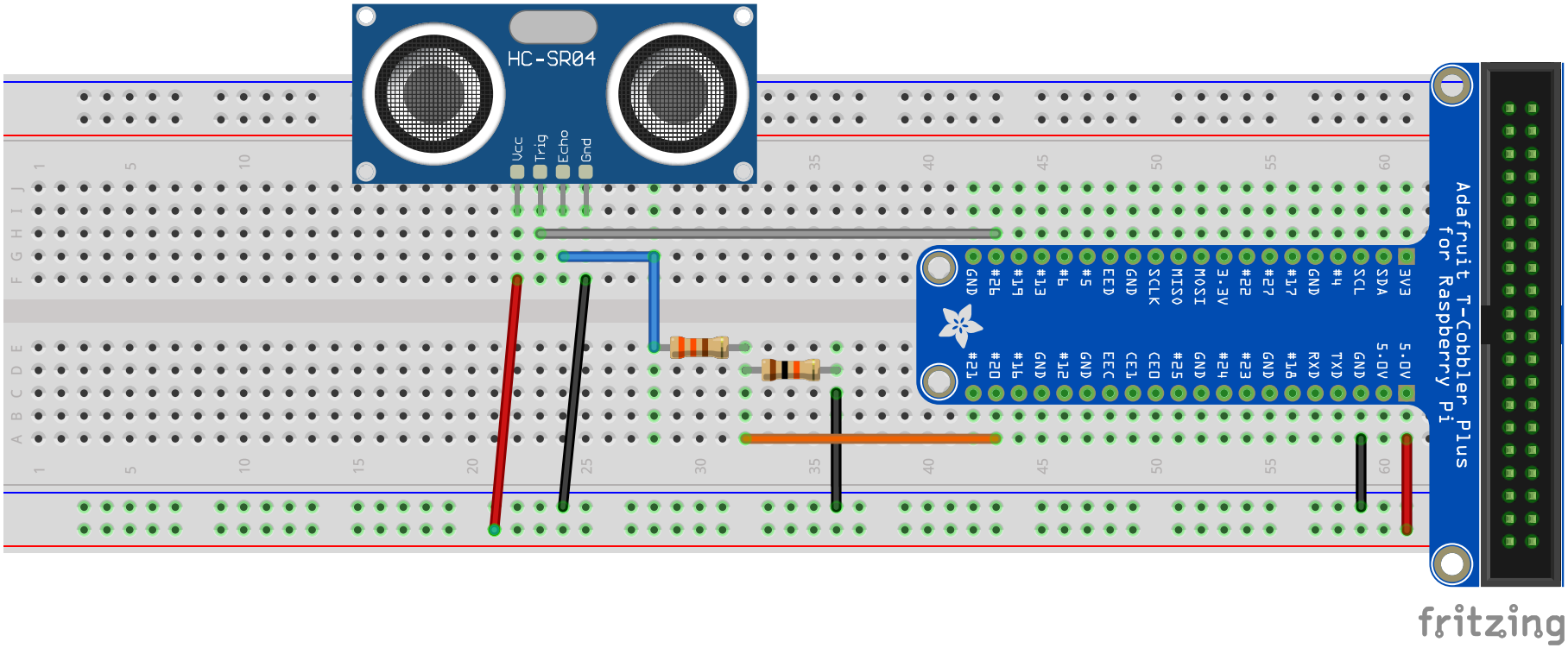


Abbildung: Aufbau des Entfernungsmessers mit Spannungsteiler

Entfernungsmessung – Programmcode I

```
#Bibliotheken einbinden
import RPi.GPIO as GPIO
import time

#GPIO Modus (BOARD / BCM)
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

#GPIO Pins zuweisen
GPIO_TRIGGER = 26
GPIO_ECHO = 24

#Richtung der GPIO-Pins festlegen (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
```

Entfernungsmessung – Programmcode II

```
def get_distance():
    # setze Trigger auf HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # setze Trigger nach 0.01ms auf LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartZeit = time.time()
    StopZeit = time.time()

    # speichere Startzeit
    while GPIO.input(GPIO_ECHO) == 0:
        StartZeit = time.time()
```

Entfernungsmessung – Programmcode III

```
# speichere Ankunftszeit
while GPIO.input(GPIO_ECHO) == 1:
    StopZeit = time.time()

# Zeit Differenz zwischen Start und Ankunft
TimeElapsed = StopZeit - StartZeit

# mit der Schallgeschwindigkeit (34300 cm/s)
 ↪  multiplizieren
# und durch 2 teilen, da hin und zurueck
distanz = (TimeElapsed * 34300) / 2

return distanz
```

Entfernungsmessung – Programmcode IV

```
while True:
    entfernung = get_distance()
    print ("Gemessene Entfernung = "
     ↪  +str(int(entfernung))+ " cm")
    time.sleep(1)
```

Quelle: tutorials-raspberrypi.de (17.10.2021)

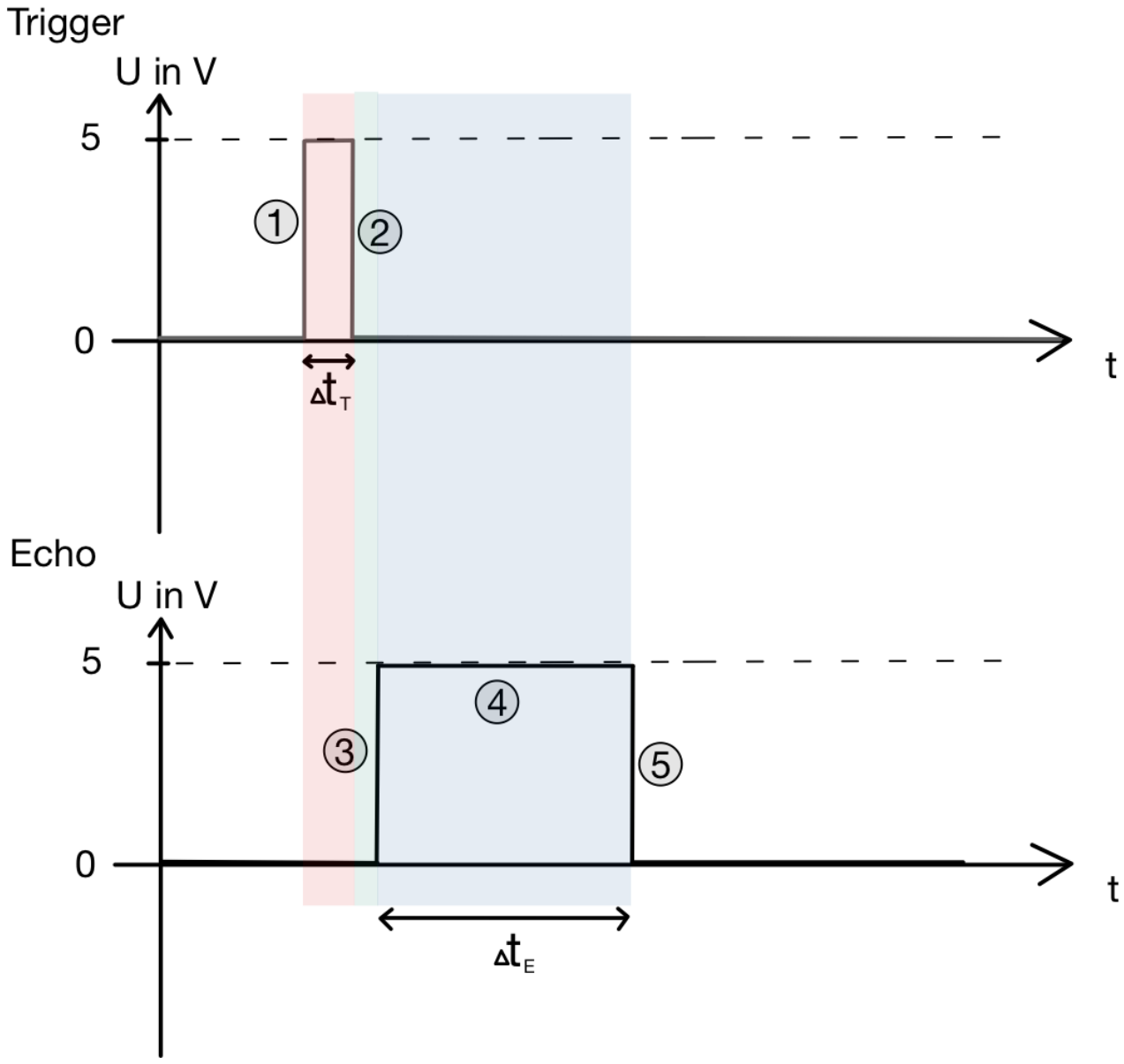


Abbildung: Pegel während einer Entfernungsmessung

Entfernungsmessung – Herleitung

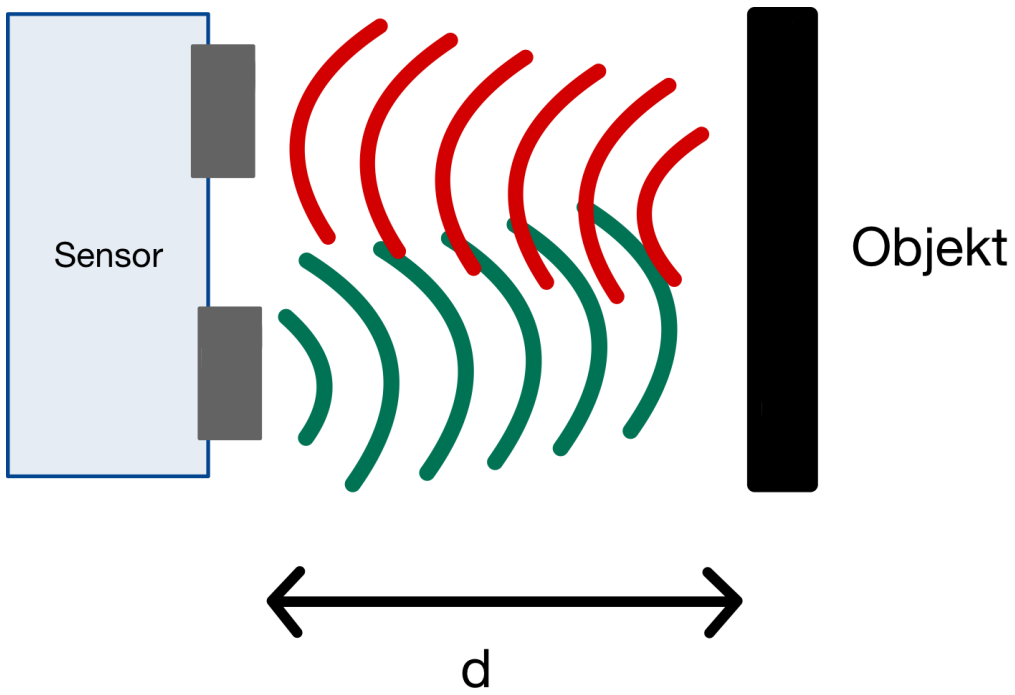


Abbildung: Ultraschall-Entfernungsmessung

$$2 \cdot d = v_{\text{Schall}} \cdot \Delta t_E \quad \Longrightarrow \quad d = \frac{v_{\text{Schall}} \cdot \Delta t_E}{2}$$

Geschwindigkeitsmessung – Aufgabe

Aufgabe:

Ermittle aus den eingelesen Entfernungen die Geschwindigkeit eines sich bewegenden Objektes.

Geschwindigkeit - Definition

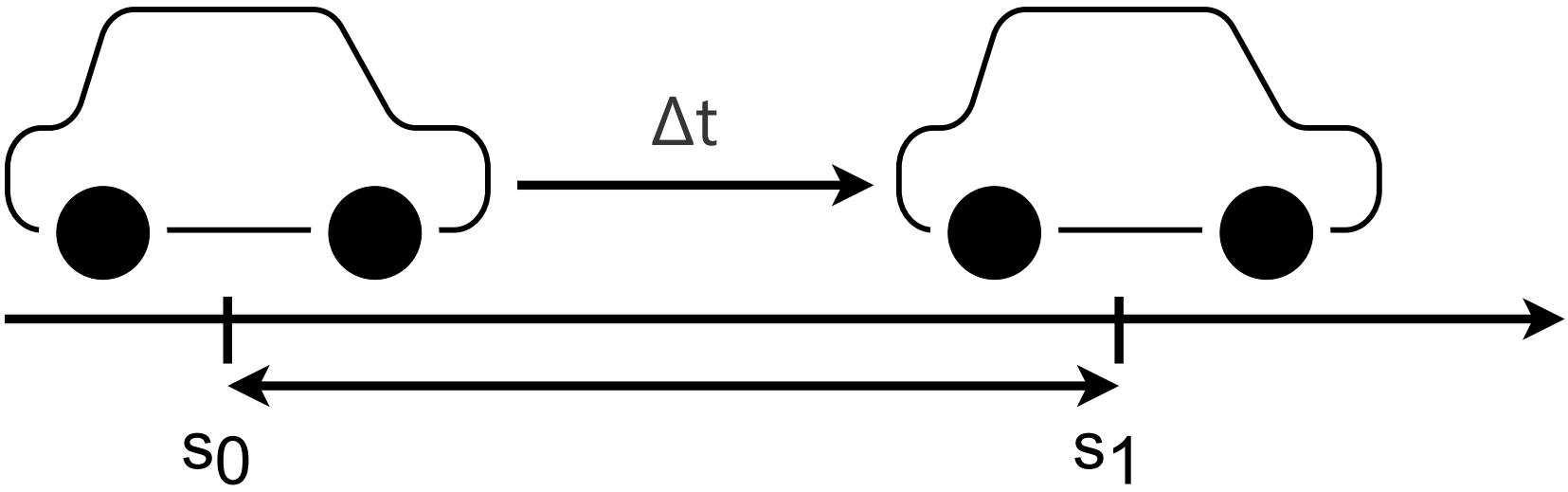


Abbildung: Auto fährt in der Zeitspanne Δt die Strecke Δs

$$v = \frac{\Delta s}{\Delta t} = \frac{s_1 - s_0}{t_1 - t_0}$$

Geschwindigkeit - Beispiel

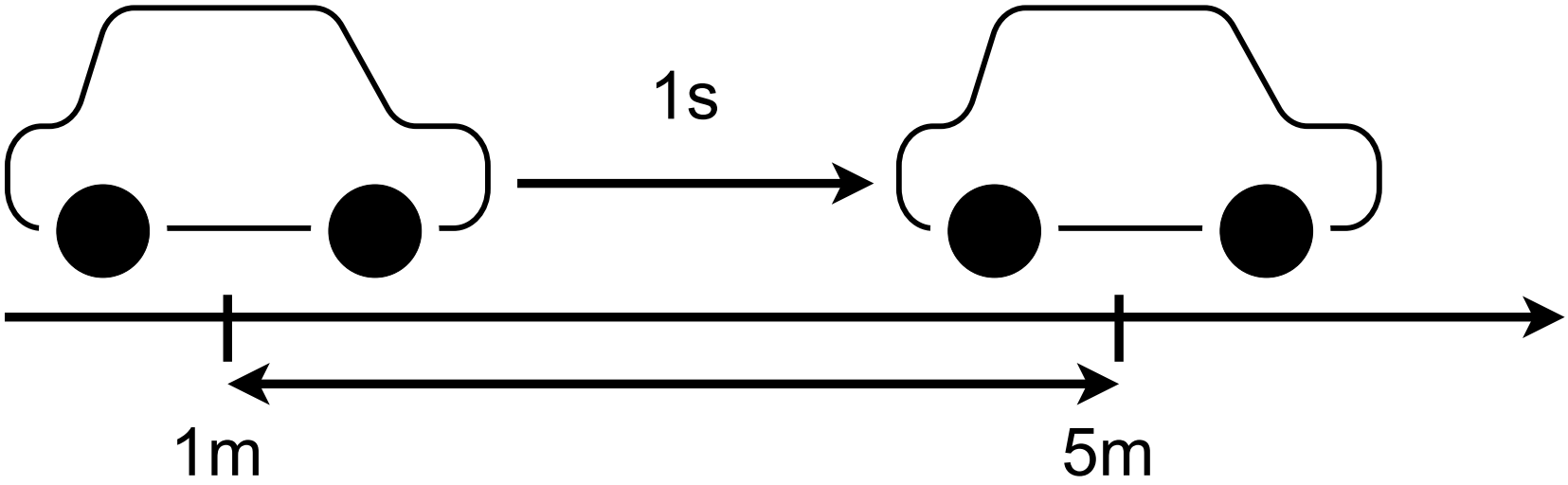


Abbildung: Auto fährt in einer Zeitspanne von 1 s die Strecke 5 $m - 1$ $m = 4$ m

$$v = \frac{\Delta s}{\Delta t} = \frac{5\ m - 1\ m}{1\ s} = 4\ \frac{m}{s}$$

Geschwindigkeitsmessung – Programmcode I

```
#Bibliotheken einbinden
import RPi.GPIO as GPIO
import time

#GPIO Modus (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#GPIO Pins zuweisen
GPIO_TRIGGER = 26
GPIO_ECHO = 24

#Richtung der GPIO-Pins festlegen (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
```

Geschwindigkeitsmessung – Programmcode II

```
def get_distance():
    # setze Trigger auf HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # setze Trigger nach 0.01ms aus LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartZeit = time.time()
    StopZeit = time.time()

    # speichere Startzeit
    while GPIO.input(GPIO_ECHO) == 0:
        StartZeit = time.time()
```

Geschwindigkeitsmessung – Programmcode III

```
    # speichere Ankunftszeit
    while GPIO.input(GPIO_ECHO) == 1:
        StopZeit = time.time()

    # Zeit Differenz zwischen Start und Ankunft
    TimeElapsed = StopZeit - StartZeit

    # mit der Schallgeschwindigkeit (34300 cm/s)
     ↪  multiplizieren
    # und durch 2 teilen, da hin und zurueck
    distanz = (TimeElapsed * 34300) / 2

    return distanz
```

Geschwindigkeitsmessung – Programmcode IV

```
try:
    while True:
        # Bestimmung der Distanz -> t=0
        entfernung1 = get_distance()
        # Warte eine Sekunde ab
        time.sleep(1)
        # Bestimmung der Distanz -> t=1
        entfernung2 = get_distance()

        geschwindigkeit = entfernung1 - entfernung2

        print ("Gemessene Geschwindigkeit = "
         ↪  +str(int(geschwindigkeit))+ "cm/s")
        time.sleep(1)
```

Geschwindigkeitsmessung – Programmcode V

```
except KeyboardInterrupt:
    GPIO.cleanup()
```

Blitzer – Aufgabe

Aufgabe:

Kannst du dein Programm erweitern, sodass eine LED aufblinkt, wenn eine bestimmte Geschwindigkeit überschritten wird? (Hierzu benötigst du eine LED mit passendem Vorwiderstand)

Blitzer – Aufbau

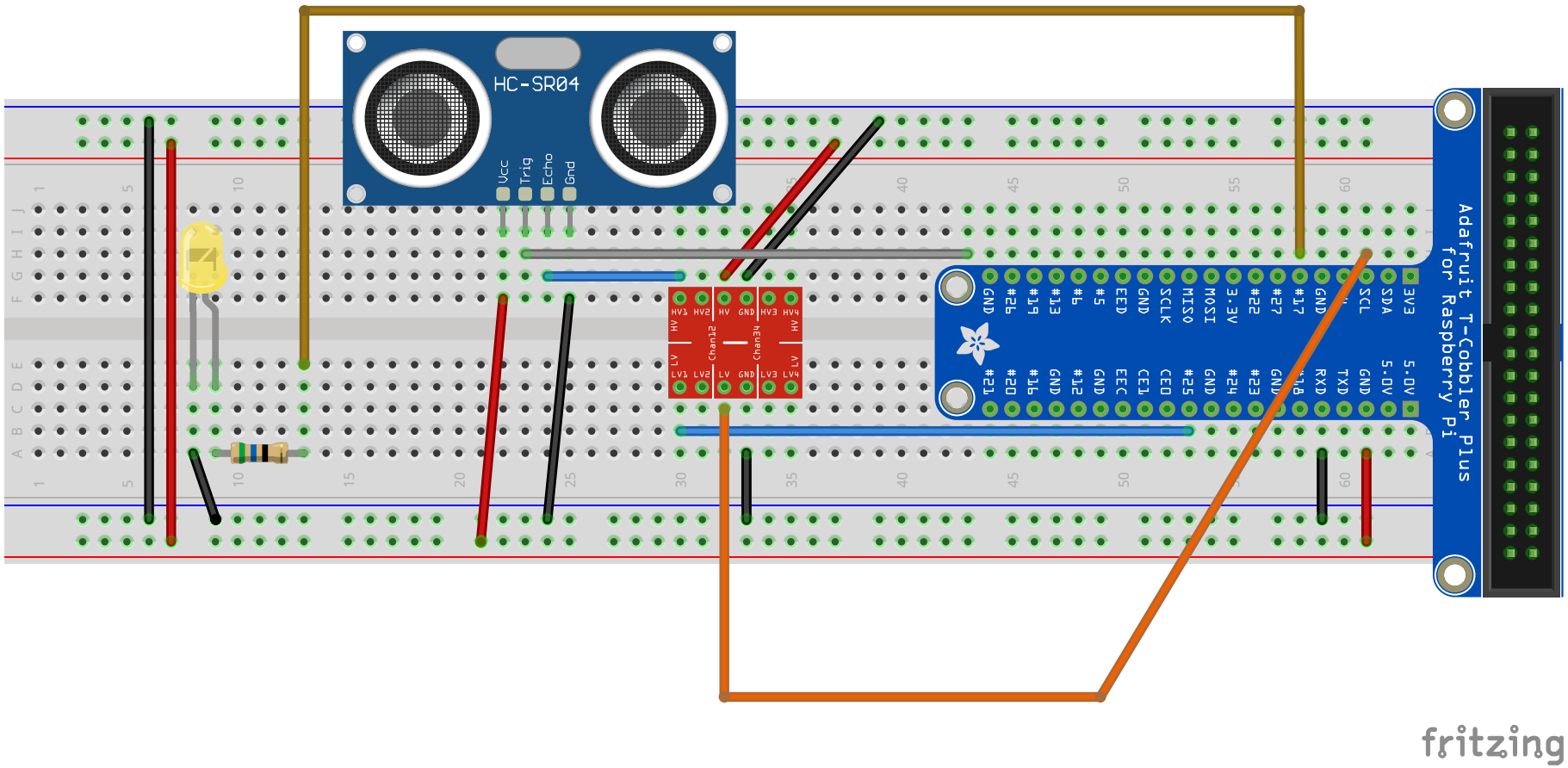


Abbildung: Aufbau des Blitzers

Blitzer – Programmcode I

```
#Bibliotheken einbinden
import RPi.GPIO as GPIO
import time

#GPIO Modus (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#GPIO Pins zuweisen
GPIO_TRIGGER = 26
GPIO_ECHO = 24
led = 17

#Richtung der GPIO-Pins festlegen (IN / OUT)
GPIO.setup([GPIO_TRIGGER,led], GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
```

Blitzer – Programmcode II

```
def get_distance():
    # ... siehe vorherige Folien ...
    return distanz


try:
    while True:
        # Bestimmung der Distanz -> t=0
        entfernung1 = get_distance()
        # Warte eine Sekunde ab
        time.sleep(1)
        # Bestimmung der Distanz -> t=1
        entfernung2 = get_distance()
```

Blitzer – Programmcode III

```
geschwindigkeit = entfernung1 - entfernung2

print ("Gemessene Geschwindigkeit = "
 ↪  +str(int(geschwindigkeit))+ "cm/s")

# LED blinken lassen bei Ueberschreitung der
 ↪  Geschwindigkeit:
if geschwindigkeit > 30:
    GPIO.output(led,True)
    sleep(0.2)
    GPIO.output(led,False)

time.sleep(1)
```

Blitzer – Programmcode IV

```
except KeyboardInterrupt:
    GPIO.cleanup()
```

Gas-Sensor (Abgasmessung)

Abgasmessung – Was benötigen wir?

- Gassensor (MQ-135)
- Analog-Digital-Wandler (Adafruit - ADS 1115)
- Logik-Level-Shifter
- Jumper-Kabel
- Breadboard

Abgasmessung – Aufbau

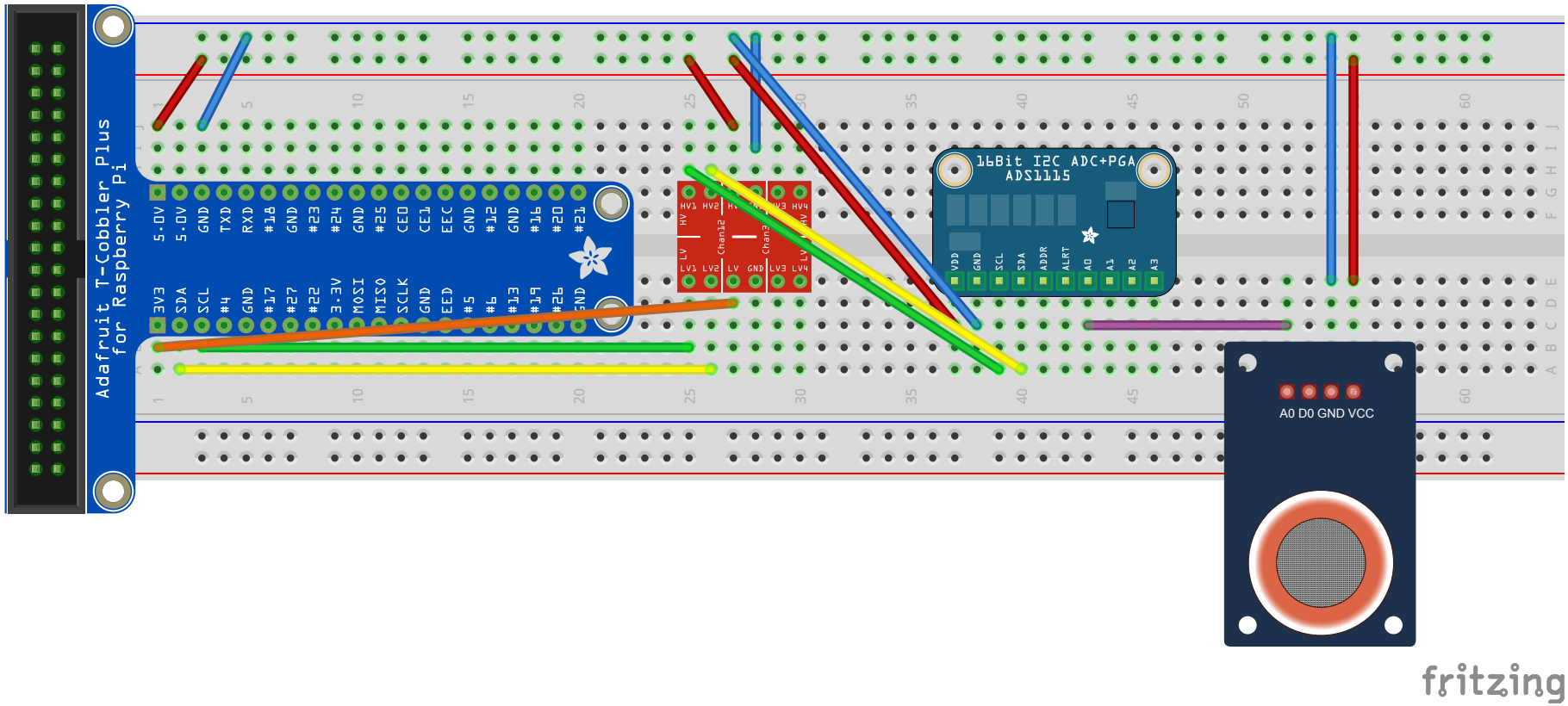


Abbildung: Aufbau des Gassensors

Abgasmessung – Programmcode

```
from time import sleep
import Adafruit_ADS1x15

adc = Adafruit_ADS1x15.ADS1115()

while True:
    # Sensor auslesen:
    value = adc.read_adc(0,gain=1)
    # Wert ausgeben:
    print(value)
    sleep(1)
```

Abgasmessung – Aufgabe

Aufgabe:

Erweitere das Programm so, dass es bei Überschreitung eines Grenzwerts (1200) eine Warnung ausgibt.

Abgaswarnung – Programmcode

```
from time import sleep
import Adafruit_ADS1x15

adc = Adafruit_ADS1x15.ADS1115()

while True:
    # Sensor auslesen:
    value = adc.read_adc(0,gain=1)
    # Wert ausgeben:
    print(value)
    # Wert ueberpruefen:
    if(value > 1200):
        print("Rauch erkannt!")
    sleep(1)
```

Zusammenfassung & Ausblick I

Was haben wir gelernt?

- Einrichtung des Raspberry Pi
- Grundlegende Bedienung
- Python-Grundlagen
- Ansteuern von GPIO-Pins
- Auslesen von Sensoren
  - Entfernungssensor
  - Gas-Sensor

Zusammenfassung & Ausblick II

Wie kann es weitergehen?

- Forscher-AG-Projekt
- Teilnahme an Wettbewerben wie Jugend forscht

→ **Bei Interesse gerne melden!**

Zusammenfassung & Ausblick III

Beispiel-Projekt: Messwerte-Erfassung am Wasserrad