DIY-Alarmanlage

Wenn du dich bewegst, wird es laut.

- Methodenkarte: DIY-Alarmanlage
- Installation Thonny
- Das Raspberry Pi Pico mit Thonny programmieren
- Zusammenbau
- Der Code
- Online-Ressourcen

Methodenkarte: DIY-Alarmanlage

Wenn du dich bewegst, wird es laut.

Zielgruppe	Dauer	Level	Gruppengröße
ab 10 Jahren	5 bis 10 Stunden	3	3 TN's

Kurzbeschreibung

Das Projekt „DIY-Alarmanlage“ vermittelt in Gruppenarbeit Grundlagen der Elektronik und Programmierung mit MicroPython. Ein Bewegungsmelder steuert einen MP3-DFPlayer, der Audiodateien abspielt. Die Teilnehmenden lernen Stromkreise, modulares Design und algorithmisches Denken kennen und diskutieren ethische Aspekte der Technologie.

> **Ziele**
> *- Grundlagen von Stromkreisläufen vermitteln*
> *- MicroPython vermitteln*
> *- Teamarbeit fördern*
> *- algorithmisches Denken stärken*
> *- ethische Aspekte digitaler Technologien reflektieren*

Material	Werkzeug
• Raspberry Pi Pico Set • Breadboard • Jumper, Dupont Kabel Set • MP3-DFPlayer • Lausprecher 3W 8Ohm • Bewegungssensor (PRI-Sensor)	• Lötstation/-set • Zangenset • Sortier-/Projektbox

Ablauf

1. **Vorbereitung:**
   Die Projektleitung bereitet alle Materialien vor (siehe QR-Code). Dabei ist eine genaue Verkabelung essenziell, besonders bei den Pins und Lautsprechern. Ein Demomodell sowie die Installation von Micropython und des Codes erleichtern die Einführung. Notwendige Software wie Thonny sollte auf einem Computer, Laptop oder Raspberry Pi 400 installiert werden, um die Programmierung zu ermöglichen. Je nach Alter, Vorwissen und Beziehungsarbeit kann die Workshopzeit reduziert werden (z.B. bei den Phasen Projektvorbereitung und Projektstart).
2. **Projektstart:**
   Gemeinsam werden Regeln für die Zusammenarbeit festgelegt. Jede Gruppe entscheidet sich für einen Namen und definiert Team-Rollen. Danach folgt eine thematische Einführung. Anschließend werden Anwendungen von Alarmanlagen und ihre Funktionen in einer Diskussion beleuchtet, unterstützt durch Recherche und Beispiele.
3. **Praktische Arbeit: Aufbau und Test:**
   Die Teilnehmenden bauen die Module eigenständig zusammen. Die Workshopleitung unterstützt bei Bedarf. Nach dem Zusammenbau werden die Funktionen getestet, zum Beispiel die Bewegungserkennung und Audioausgabe.
4. **Programmierung:**
   Die Gruppen schließen den Raspberry Pi Pico an, analysieren den vorhandenen Code und nehmen einfache Änderungen, etwa bei der Lautstärkeregelung, vor. Erweiterungen wie die zufällige Audiowiedergabe können implementiert werden, um die Programmierkenntnisse zu vertiefen.
5. **Reflexion:**
   Abschließend reflektieren die Gruppen über ihre Zusammenarbeit und den Projekterfolg. Verbesserungsvorschläge werden gesammelt und im Plenum besprochen. Die Projektleitung gibt abschließen ein Feedback und Anregungen für Folgeprojekte.

Autor*in: Shelly Pröhl (*Büro Berlin des JFF*)

Installation Thonny

Wir programmieren das Raspberry Pi Pico mit der Skriptsprache MicroPython in der kostenlosen Entwicklungsumgebung (IDE) Thonny. Dafür verwenden wir einen Computer oder Laptop.

Thonny	https://thonny.org/
Micropython	https://micropython.org/

Was ist eine IDE?

Eine IDE (Integrated Development Environment) ist eine Software, die euch beim Schreiben, Testen und Ausführen von Code (Programmen) unterstützt. Sie vereint viele hilfreiche Werkzeuge an einem Ort, darunter:

- **Texteditor:** Zum Schreiben und Bearbeiten von Code.
- **Debugger:** Zum Finden und Beheben von Fehlern im Code.
- **Terminal:** Zum Ausführen des Codes und Anzeigen von Ergebnissen.

Was ist Thonny?

Thonny ist eine einfache und benutzerfreundliche IDE, die speziell für Python entwickelt wurde. Sie eignet sich besonders gut für Einsteiger*innen, die das Programmieren gerade erst lernen. Thonny bietet eine übersichtliche Benutzeroberfläche und viele hilfreiche Funktionen, die den Einstieg ins Programmieren erleichtern. Es ist ein großartiges Tool, um erste Schritte mit Python und MicroPython zu machen.

drawing drawing

Installation von Thonny auf verschiedenen Betriebssystemen

Installation auf Windows

- **Gehe zur offiziellen Thonny-Website.**
- Klicke auf den Download für Windows.
- Lade die Installationsdatei herunter und öffne sie, wenn der Download abgeschlossen ist.

- Folge den Anweisungen des Installationsassistenten, um Thonny zu installieren.
- Sobald die Installation abgeschlossen ist, kannst du Thonny über das Startmenü öffnen.

Installation auf Linux (Ubuntu)

- **Öffne das Terminal** auf deinem Ubuntu-System.
- Gib den folgenden Befehl ein, um das Paket zu aktualisieren:

```
sudo apt update
```

- Installiere Thonny, indem du den folgenden Befehl eingibst:

```
sudo apt install thonny
```

- Warte, bis die Installation abgeschlossen ist. Danach kannst du Thonny im Anwendungsmenü finden und starten.

Installation auf macOS

- **Gehe zur offiziellen Thonny-Website.**
- Klicke auf den Download für macOS.
- Lade die .dmg-Datei herunter und öffne sie, wenn der Download abgeschlossen ist.
- Ziehe das Thonny-Symbol in den Programme-Ordner, um die Installation abzuschließen.
- Öffne Thonny, indem du es im Programme-Ordner findest oder über Spotlight suchst.

> **Übung**
>
> Versuche nach der Installation, siehe unten, über die IDE Thonny eine Bibliothek zu installieren, zum Beispiel die Bibliothek 'NeoPixel'.
>
> - Thonny öffnen ->
> - Menü -> Tools -> Manage Plug-ins
> - Suche nach 'neopixel' -> installieren

Das Raspberry Pi Pico mit Thonny programmieren

Um ein neues Raspberry Pi Pico zu programmieren, müssen wir es zunächst vorbereiten. Es mag anfangs nach vielen Schritten klingen, aber sobald ihr es einmal gemacht habt, geht der Rest richtig schnell! (□ ‘‿ ’□) Wir teilen den Prozess in drei Schritte auf:

1. **Installation von MicroPython auf dem Raspberry Pi Pico**
2. **Raspberry Pi Pico mit Thonny öffnen**
3. **Ein Programm auf dem Raspberry Pi Pico speichern/laden**

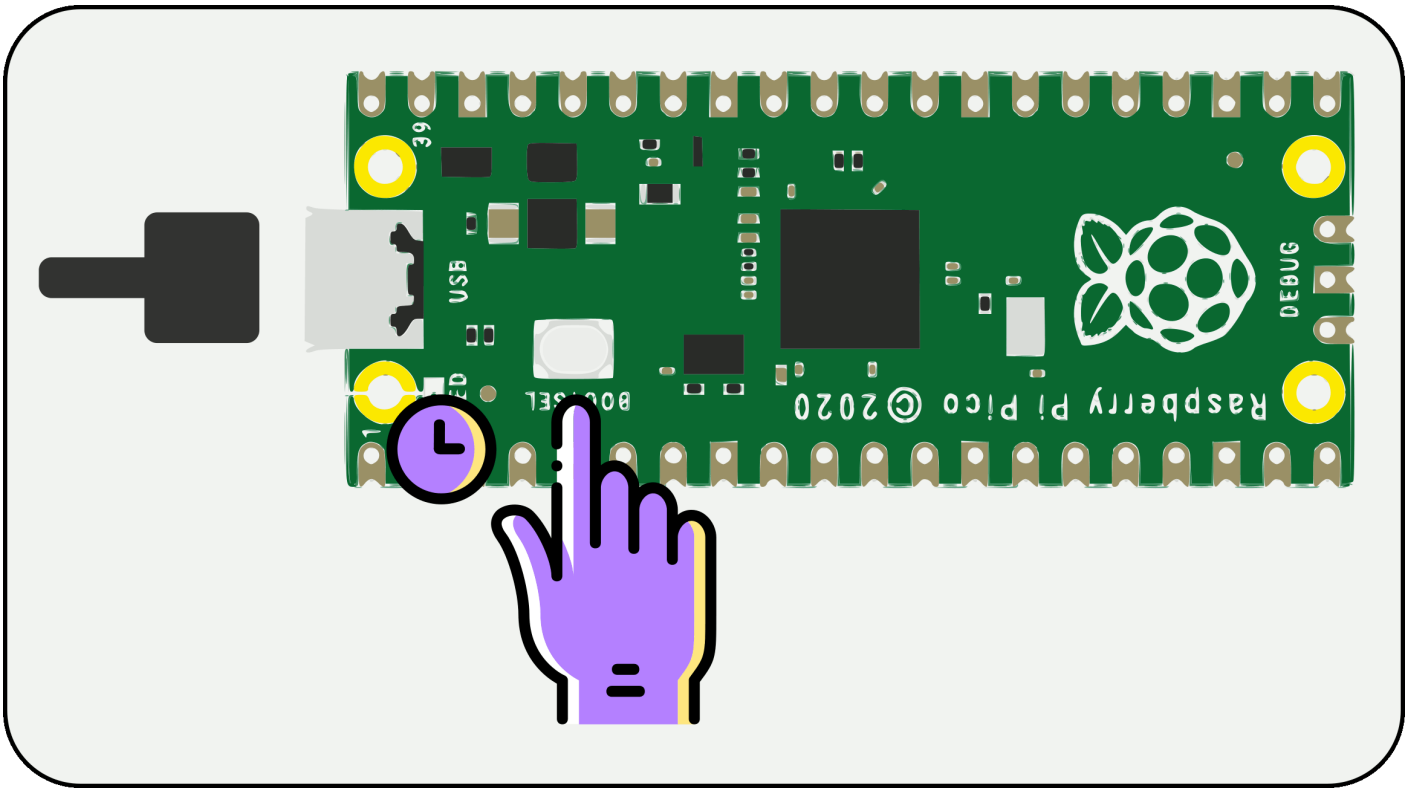
Installation von Micropython auf dem Raspberry Pi Pico

- **Ladet euch Micropython herunter:**

  Besucht die MicroPython-Website und ladet die passende UF2-Datei für das Raspberry Pi Pico herunter.

  (Zum Beispiel v1.24.0 (2024-10-25) .uf2)
- **Schließt das Raspberry Pi Pico an den Computer an:**
  Verwendet ein Micro-USB-zu-USB-A-Kabel. Haltet dabei den BOOTSEL-Knopf beim Raspberry Pi Pico gedrückt, während ihr das USB-Kabel anschließt.
  (siehe Animation)
- **Kopiert die UF2-Datei auf das neue Laufwerk:**
  Sobald das Pico als neues Laufwerk auf eurem Computer erscheint, zieht die heruntergeladene UF2-Datei per Drag-and-Drop in dieses Laufwerk. Dadurch wird MicroPython auf dem Raspberry Pi Pico installiert.

> Während der Installation oder dem Laden von MicroPython auf das Raspberry Pi Pico trennt sich das Laufwerk automatisch vom Computer. Dies zeigt an, dass die Installation abgeschlossen ist. Zieht auf keinen Fall das Kabel während dieses Prozesses vom Pico ab, da dies die Installation unterbrechen und zu Fehlern führen könnte.
>
> Sicherheit geht vor – lasst das Pico in Ruhe arbeiten! (□ ^ ᵕ ^□)



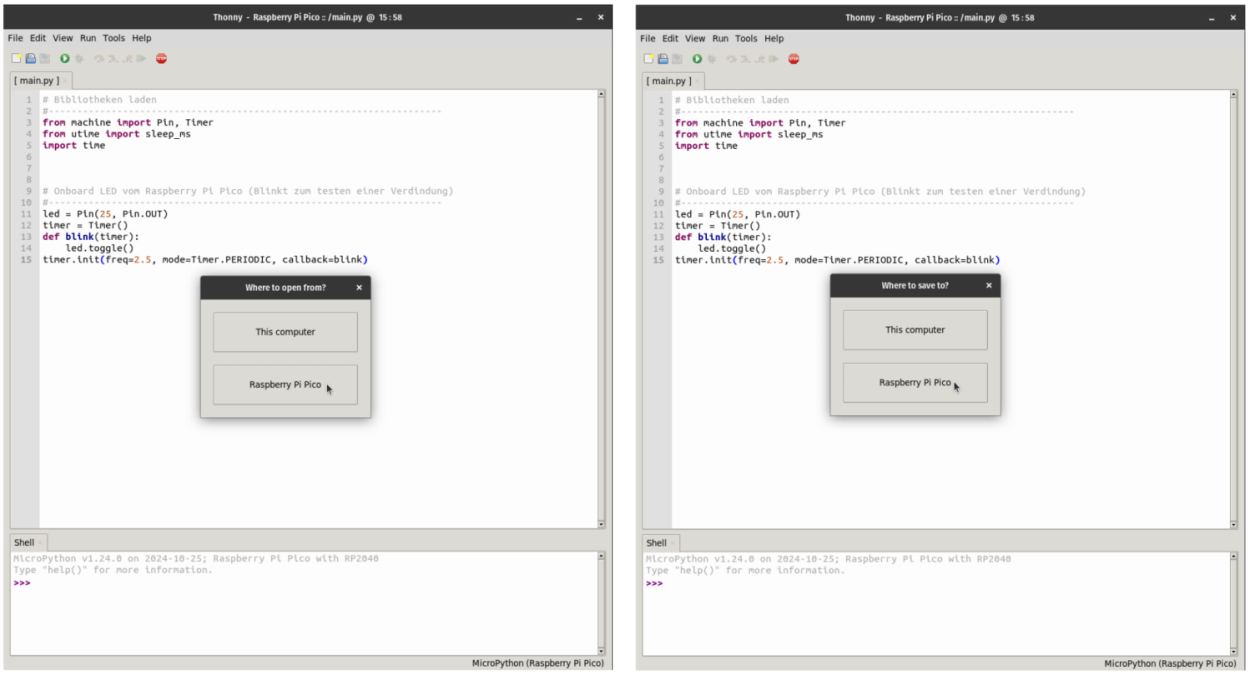
Mit Thonny ein Programm auf dem Raspberry Pi Pico speichern/laden

Nachdem ihr Thonny geöffnet habt, könnt ihr über das Menü die Option **Dateien/Files** auswählen, um Dateien zu verwalten.

- Mit der Option **Öffnen/Open** öffnet sich ein Dialog, in dem ihr auswählen könnt, ob ihr eine Datei von eurem Computer oder direkt vom Raspberry Pi Pico laden möchtet.
- Mit der Option **Speichern/Save** öffnet sich ein Dialog, in dem ihr auswählen könnt, wo ihr euer Programm speichern möchtet – entweder auf dem Computer oder auf dem Raspberry Pi Pico.

So könnt ihr eure Programme einfach verwalten und sicherstellen, dass sie immer an der richtigen Stelle gespeichert sind!

> Achtet beim Speichern auf dem Raspberry Pi Pico darauf, dass die Datei den Namen **main.py** hat. Nur mit diesem Dateinamen erkennt das Pico euer Programm automatisch und führt es nach dem Starten aus. ᓚ(^. .^)



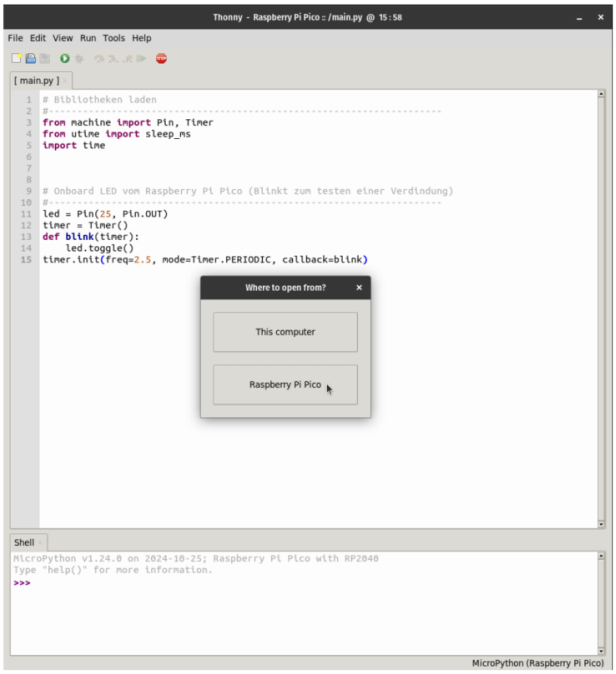
Raspberry Pi Pico mit Thonny öffnen

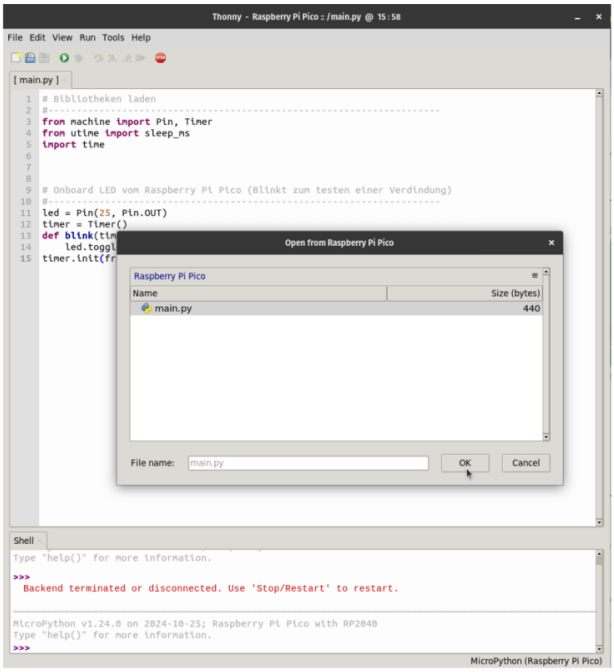
Normalerweise kommuniziert Thonny automatisch mit dem Raspberry Pi Pico, sobald es angeschlossen ist. Wenn ihr Thonny geöffnet habt, geht wie folgt vor:

1. Wählt im Menü die Option **Dateien/Files** aus.
2. Im neuen Dialog könnt ihr das **Raspberry Pi Pico** als Speicherort auswählen.
3. Anschließend könnt ihr eure **main.py**-Datei auf dem Pico finden, auswählen und öffnen.

So könnt ihr sicherstellen, dass euer Programm korrekt geladen und ausgeführt wird!

> Nachdem ihr eure **main.py** zum Öffnen ausgewählt habt, könnt ihr das Programm in Thonny bearbeiten. Wenn ihr Änderungen vornehmt und auf **Speichern** klickt, wird das Programm automatisch auf dem Raspberry Pi Pico aktualisiert und gespeichert. So bleiben eure Änderungen direkt auf dem Pico erhalten! ¯_(□)_/¯





Programm starten/stoppen

Nachdem ihr ein Programm erfolgreich auf das Raspberry Pi Pico gespeichert oder übertragen habt, könnt ihr es ausführen und bei Bedarf stoppen:

- **Starten:** Klickt in der Symbolleiste auf den runden **grünen Button**, um das Programm auf dem Raspberry Pi Pico zu starten. Alternativ könnt ihr die **Taste F5** drücken.
- **Stoppen:** Klickt in der Symbolleiste auf den **roten Stop-Button**, um das Programm zu stoppen. Alternativ könnt ihr die Tastenkombination **Strg + F2** verwenden.

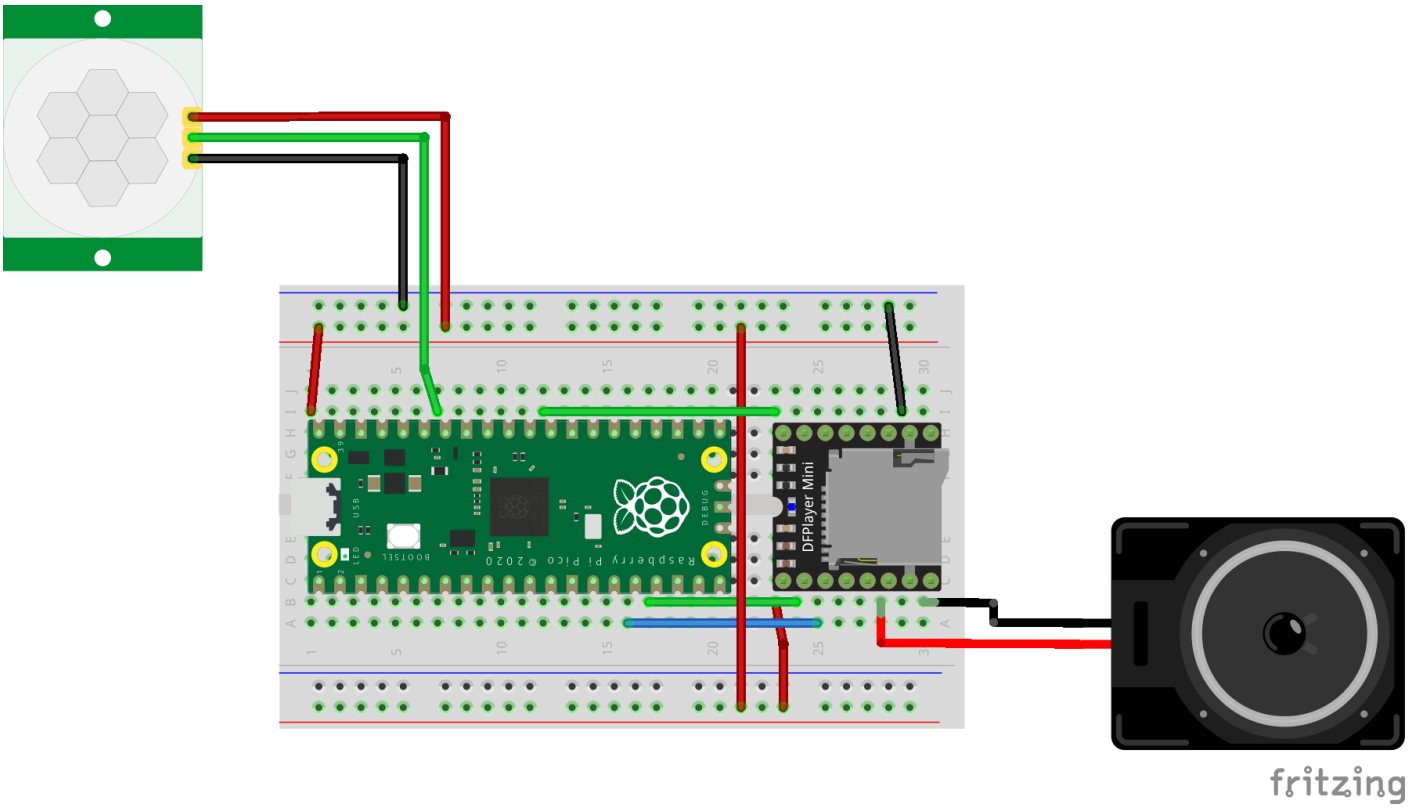
Buttons in Thonny zur Steuerung des Programmes auf dem Raspberry Pi Pico

Programm
starten

Programm
stoppen

Zusammenbau

Das Projekt hat vier Phasen, welche euch hier genauer beschrieben werden.



> **Beispiel-Verkabelung** (siehe Abbildung oben)**:**
> In der Abbildung werden rote, schwarze und grüne Kabel (Jumper) verwendet, die jeweils eine spezifische Funktion haben:
>
> - Rot (Power/PWR): Liefert Energie an das Bauteil, damit es funktioniert.
> - Schwarz (Ground/GRD): Schließt den Stromkreis und leitet überschüssige Energie ab.
> - Grün (Data): Überträgt die Daten zwischen dem Raspberry Pi Pico und dem Bauteil.

1) Stromversorgung einrichten:

- Verbinde den **5V-Pin** des Raspberry Pi Pico mit der **positiven Leiste** (rote Linie) des Breadboards.
- Verbinde einen **GND-Pin** des Raspberry Pi Pico mit der **negativen Leiste** (blaue Linie) des Breadboards.
  Jetzt können alle Bauteile auf dem Breadboard mit Strom versorgt werden.

2) Bewegungssensor anschließen:

- Stecke ein Jumper-Kabel vom **VCC-Pin** des Relaismoduls in die **positive Leiste** des Breadboards.
- Verbinde den **GND-Pin** des Relaismoduls mit der **negativen Leiste** des Breadboards.
- Schließe den **IN-Pin** des Relaismoduls mit einem Jumper-Kabel an den **GPIO-Pin GP28** des Raspberry Pi Pico an.

3) MP3 DFPlayer anschließen:

- Stecke ein Jumper-Kabel vom **VCC-Pin** des Relaismoduls in die **positive Leiste** des Breadboards.
- Verbinde den **GND-Pin** des Relaismoduls mit der **negativen Leiste** des Breadboards.
- Verbinden des **Busy-Pins** des DFPlayer mit **GPIO-Pin GB22** des Picos
- **UART**-Konfiguration für DFPlayer
  - **TX-Pin** des DFPlayer mit dem **RX-Pin GB17** des Picos verbinden
  - **RX-Pin** des DFPlayer mit dem **TX-Pin GB16** des Picos verbinden

> **UART** steht für **"Universal Asynchronous Receiver Transmitter" (dt. Universeller asynchroner Empfängersender)**. Es ist eine Methode, mit der elektronische Geräte miteinander **kommunizieren** können, indem sie Daten seriell (also **nacheinander, Bit für Bit**) über zwei Drähte senden: einen zum Senden (**TX**) und einen zum Empfangen (**RX**).

> Stell dir vor, **zwei Freunde** wollen miteinander sprechen, aber sie haben nur **eine Leitung**, über die sie abwechselnd sprechen können. Sie müssen sich einigen, **wie schnell** sie sprechen und **wann** sie anfangen und aufhören, damit sie sich verstehen.

4) Lautsprecher anschießen:

- Der DFPlayer Mini verfügt über eingebaute Verstärker an den **Pins SPK_1 und SPK_2**, sodass du den Lautsprecher direkt ohne zusätzlichen Verstärker anschließen kannst.

Der Code

Hier ist ein Beispielcode für eure Programmierung in Micropython in der IDE Thonny auf dem Raspberry Pi Pico.

```
import time
import random
from machine import Pin, UART
from dfplayer import DFPlayer

# HC-SR501 Bewegungsmelder
pir_sensor = Pin(28, Pin.IN)  # Verbinde den Bewegungsmelder mit Pin 28

# Onboard-LED des Raspberry Pi Pico
led_pico = Pin(25, Pin.OUT)  # Onboard-LED an Pin 25

# Busy-Pin des DFPlayer Mini
busy_pin = Pin(22, Pin.IN)  # Verbinde den Busy-Pin des DFPlayer mit Pin 22

# UART-Konfiguration für DFPlayer
uart = UART(0, baudrate=9600, tx=Pin(16), rx=Pin(17))

# DFPlayer MP3 Player
dfplayer = DFPlayer(uart_id=0, tx_pin_id=16, rx_pin_id=17)  # Initialisiere DFPlayer mit UART Pins

time.sleep(2)  # Warte, bis der DFPlayer vollständig bereit ist

# Setze die Lautstärke einmalig
print("Setze Lautstärke...")
dfplayer.volume(10)  # Lautstärke auf 15 setzen (Bereich 0-30)
time.sleep(0.5)

# Funktion zum Abspielen einer zufälligen MP3-Datei
def play_random_mp3():
    track_number = random.randint(1, 4)  # Wähle eine zufällige Datei im Bereich 001-004
    time.sleep(0.2)  # Kurze Pause nach der Zufallsauswahl
    print(f"Spiele MP3-Datei {track_number} ab...")
```

```
    dfplayer.play(1, track_number)  # Spiele die Datei im Ordner ./01/
    time.sleep(0.2)  # Zusätzliche kurze Pause für Stabilität
    print("MP3-Wiedergabe gestartet.")

# Hauptprogramm
playing = False
waiting_for_reset = False
reset_timeout = 5  # Timeout für die Bewegungserkennung in Sekunden
reset_start_time = 0  # Startzeit für das Timeout

while True:
    # Prüfen, ob der Sensor keine Bewegung mehr meldet, bevor eine neue erkannt werden kann
    if not playing and not waiting_for_reset and pir_sensor.value() == 1:
        print("Bewegung erkannt!")
        led_pico.value(1)  # LED einschalten
        play_random_mp3()
        playing = True  # Setzt den Status auf "abspielend"

    if playing:
        # Überprüfe den Status des Busy-Pins und steuere die LED entsprechend
        if busy_pin.value() == 0:  # Busy-Pin ist LOW, wenn eine Datei abgespielt wird
            print("DFPlayer spielt MP3-Datei ab.")
        else:
            print("DFPlayer ist im Leerlauf.")
            led_pico.value(0)  # LED ausschalten
            playing = False  # Wiedergabe ist abgeschlossen
            waiting_for_reset = True  # Warte auf die Beendigung der Bewegungserkennung
            reset_start_time = time.time()  # Startzeit für das Timeout setzen

    # Überprüfen, ob das Timeout abgelaufen ist oder der Sensor keine Bewegung mehr meldet
    if waiting_for_reset:
        if pir_sensor.value() == 0:
            print("Bereit für neue Bewegungserkennung.")
            waiting_for_reset = False
        elif time.time() - reset_start_time > reset_timeout:
            print("Timeout erreicht. Bereit für neue Bewegungserkennung.")
            waiting_for_reset = False

    time.sleep(0.1)  # Kurze Pause, um Sensor nicht zu überlasten
```

Online-Ressourcen

/