
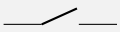
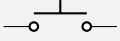
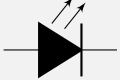






## Skill-Card: Schaltpläne lesen

Bei Schaltplänen sind alle Kabel waagerechte oder senkrechte Linien gezeichnet. Die damit verbundenen Bauteile haben besondere Symbole:

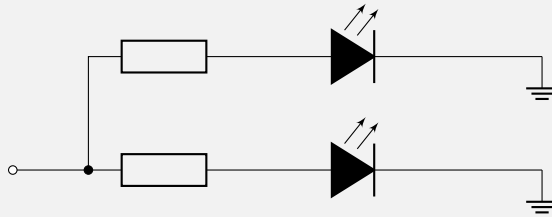
	Ein Widerstand (oft mit Wert)
	Ein normaler Schalter
	Ein Taster (Nur bei Betätigung geschlossen)
	Eine Leuchtdiode (LED)

Kreuzen sich zwei Linien, sind die Kabel nur dann miteinander verbunden, wenn die Kreuzung mit einem Punkt markiert wurde.

Die Bauteile müssen natürlich an Stromquellen angeschlossen werden. Dazu gibt es auch passende Schaltzeichen:

	Batterie (Langer Strich ist der Pluspol)
	Spannungs- oder Stromquelle.
	Pin mit abgehender Verbindung.
	Massenanschluss (Minuspol, alle sind untereinander verbunden)

Bei der folgenden Zeichnung sind zwei LEDs über einen Pin mit jeweils einem Widerstand angeschlossen.

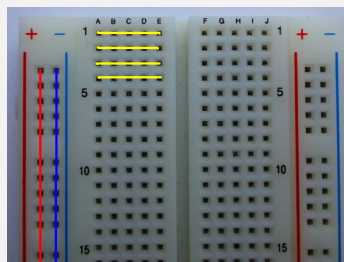


## Skill-Card: Anschluss Steckbrett

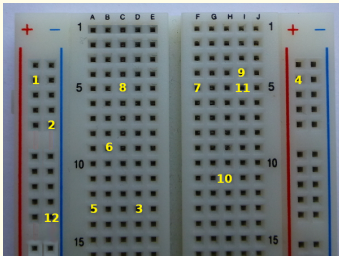
### Benötigtes Material

- Steckbrett
- Adapter mit Kabel

Für das Ausprobieren von einfachen Schaltungen eignet sich ein Steckbrett besonders, da hier die Elemente einfach verkabelt und wieder entfernt werden können. Da immer nur ein Bein eines Element in ein Loch passt, sind alle Löcher in einer Reihe links und rechts von der Mitte untereinander verbunden, wie mit gelb im folgenden Bild dargestellt. Zusätzlich sind Löcher ganz links bzw. rechts komplett untereinander verbunden (rot und blaue Linien), so dass sie sich für die Verteilung der Spannungspole eignen.

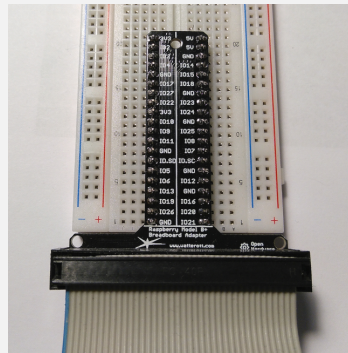
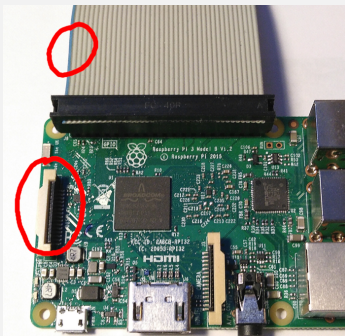


## Aufgabe



Gib an, welche der mit Nummern versehenen Löcher miteinander elektrisch verbunden ist.

Der 40-polige GPIO des Raspberry Pi kann für die Ausgabe von elektrischen Signalen bzw. als Eingang dafür genutzt werden. Damit diese Anschlüsse auf dem Steckbrett verfügbar sind, kann man sie z. B. mit einem Adapter dorthin übertragen. Damit die Anschlüsse nicht vertauscht werden hat das Flachbandkabel eine extra Markierung, oft rot oder blau. Diese Markierung wird auf der Seite genutzt, bei der der Pin mit der Nummer 1 ist. Beim Raspberry Pi ist dieses auf der Seite vom Raspberry Pi, auf der auch die SD-Karte eingesteckt wird, wie auf dem folgenden Bild zu sehen.



### Lösung

Die Punkte 2 und 12, 3 und 5 sowie 7 und 11 sind elektrisch miteinander verbunden.

## Skill-Card: LED

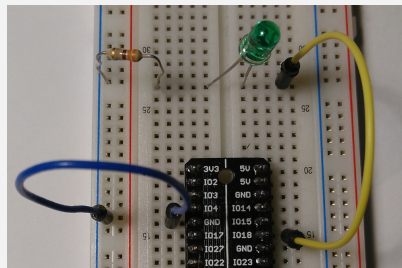
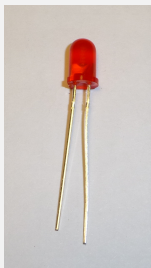
### Benötigtes Material

- Steckbrett
- LED
- 330  $\Omega$  Widerstand

### Benötigte Skill-Cards

- Anschluss Steckbrett
- Schaltpläne lesen

Beim Anschluss einer LED an den GPIO muss beachtet werden, dass diese einen Vorwiderstand benötigen, damit sie nicht durchbrennen. Außerdem ist die Anschlussrichtung bei der LED zu beachten. Wie auf dem Foto zu erkennen hat die LED zwei verschieden lange Beine. Das längere ist der Pluspol und muss auf der Seite des geschalteten Ports liegen, während das kürzere mit der Masse (GND) verbunden wird.



Im zweiten Bild und in der folgenden Schaltskizze ist aufge-

zeichnet, wie die LED mit einem  $330\ \Omega$  Widerstand an Pin 18 geschaltet ist.



Das folgende Python-Programm zeigt beispielhaft, wie man diese LED steuern kann:

```
from gpiozero import LED
from time import sleep

led = LED(18)

led.on()
sleep(2)
led.off()
```

Die Wahl des Pin für die LED kann in gewissen Rahmen frei getroffen werden. Bis auf 2, 3, 9, 10, 11, 14 und 15 können alle Pins gewählt werden, die mit einer Nummer versehen sind.

# Story-Card: Blinker

## Benötigtes Material

- Steckbrett
- LED
- 330  $\Omega$  Widerstand

## Benötigte Skill-Cards

- LED

Beim Blinker an einem Auto wird das Licht wiederholt an und aus geschaltet. Nach jedem Schaltvorgang muss etwas gewartet werden, damit man den Wechsel auch wahrnehmen kann.

```
from gpiozero import LED
from time import sleep
```

```
led = LED(18)
```

```
for i in range(0,20):
    led.on()
    sleep(0.5)
    led.off()
    sleep(0.5)
```



# Story-Card: Wechselblinker

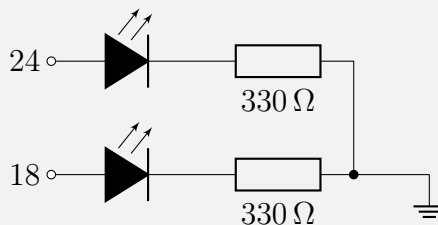
## Benötigtes Material

- Steckbrett
- Zwei LEDs
- Zwei  $330\ \Omega$  Widerstände

## Benötigte Skill-Cards

- LED

Wenn ein Bahnübergang geschlossen ist, wird dieses auch durch blinkende Lichtsignale angezeigt. Dabei befindet sich eins auf der linken und eines auf der rechten Seite. Nie sind beide gleichzeitig an, sondern abwechselnd immer nur eine Seite.



```
from gpiozero import LED  
from time import sleep
```

```
led = LED(18)  
led2 = LED(24)
```

```
for i in range(0,20):  
    led.on()  
    led2.off()  
    sleep(0.5)  
    led.off()  
    led2.on()  
    sleep(0.5)
```

# Skill-Card: Taster oder Schalter

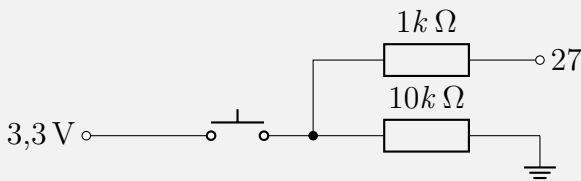
## Benötigtes Material

- Steckbrett
- Taster oder Schalter
- $1\text{ k}\Omega$  Widerstand
- $10\text{ k}\Omega$  Widerstand

## Benötigte Skill-Cards

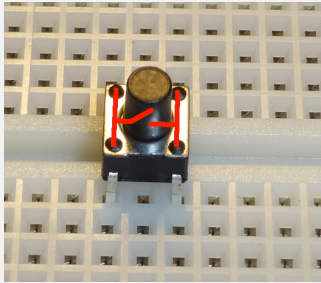
- Anschluss Steckbrett
- Schaltpläne lesen

Jeden Pin am GPIO, den man als Ausgang z. B. für eine LED nutzt, kann man auch als Eingang benutzen. Es muss nur sichergestellt sein, dass dieser entweder mit der Masse, was einer 0 entspricht, oder mit  $3,3\text{ V}$  verbunden sein, was einer 1 entspricht. Aus diesem Grund sind auch für einen Taster bzw. Schalter drei Anschlüsse und zwei Widerstände nötig:



Viele Taster haben vier Beine, mit denen sie auf einem Steckbrett befestigt werden können. In solchen Fällen sind immer zwei Beine direkt miteinander verbunden und der eigentliche Taster

befindet sich zwischen diesen Paaren, wie auch im folgenden Bild dargestellt.



Die Ansteuerung des Tasters in einem Programm ist wie folgt dargestellt. Dabei ist der Taster an Pin 27 angeschlossen. Die Wahl des Pin ist genauso wie bei der LED fast frei wählbar.

```
from gpiozero import Button
from time import sleep
```

```
button = Button(27)
```

```
for i in range(0,20):
    print (button.value)
    sleep(0.5)
```

## Skill-Card: Immer reagieren können

### Benötigte Skill-Cards

- Taster oder Schalter
- LED

Die Umsetzung der Situation, eine LED blinken soll, bis ein Taster gedrückt wird könnte wie folgt aussehen:

```
from gpiozero import Button
from gpiozero import LED
from time import sleep

button = Button(27)
led = LED(18)

while button.value == False:
    led.on()
    sleep(3)
    led.off()
    sleep(3)
```

Dieses hat aber den Nachteil, dass der Taster nur alle 6 Sekunden überprüft wird. So hat man als Anwender nur dann die Chance das Blinken zu unterbrechen, wenn man den Button in dem Moment drückt, indem die LED wieder eingeschaltet werden soll.

Programme die jederzeit auf Eingaben reagieren realisieren dies mithilfe von Threads. Auch Python kann mit Threads arbeiten, die Programmierung erfordert aber einiges Können. Des-

halb hier eine Möglichkeit, mit der man die gleiche Wirkungsweise nachstellen kann:

```
from gpiozero import Button
from gpiozero import LED
from time import sleep

button = Button(27)
led = LED(18)

count = 0
while button.value == False:
    if count == 0:
        led.on()
    if count == 30:
        led.off()
    if count == 60:
        count = -1
    sleep(0.1)
    count = count + 1
```

Hier wird ein Zähler genutzt, der alle 0,1 Sekunden erhöht wird. Deshalb wird die Schleife entsprechend oft wiederholt und bei jeder dieser Wiederholungsschritte ist es möglich, auf die Eingabe des Buttons reagieren zu können.

## Skill-Card: Helligkeitssensor

### Benötigtes Material

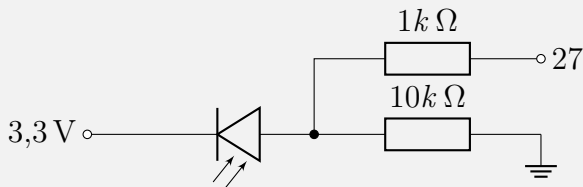
- Helligkeitssensor

### Benötigte Skill-Cards

- Taster oder Schalter

Es gibt zwei Möglichkeiten Helligkeitssensoren einzusetzen: Nutzt man sie als einfachen Schalter, wird nur geprüft ob genügend Licht darauf fällt oder eben nicht. Dieser Fall wird hier vorgestellt. Wenn man weiter differenzieren will, wie hell bzw. dunkel es ist, muss mit einem Analog/Digital-Wandler gearbeitet werden.

Da der Helligkeitssensor nur als Schalter genutzt werden soll, erfolgt die Programmierung analog zu der des Schalters bzw. Tasters. Auch die Beschaltung ist sehr ähnlich.



Dabei kommt eine Photodiode als Sensor zum Einsatz. Dabei ist auf die Polung zu achten.

## Skill-Card: Helligkeit einer LED

### Benötigtes Material

- LED

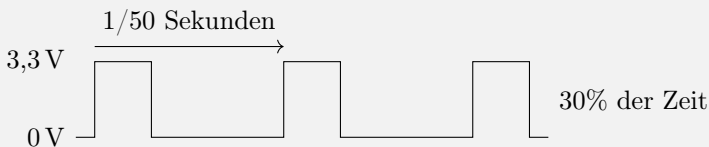
### Benötigte Skill-Cards

- LED

Alle Pins am GPIO des Raspberry Pi sind digital. Das bedeutet, dass sie bei ihnen nur zwei verschiedene Zustände möglich sind: An/Aus bzw. 1/0 oder Wahr/Falsch. Oft genug gibt es aber Fälle in denen Zwischenwerte gewünscht werden. So soll eine LED nicht mit voller Helligkeit leuchten.

Diese Zwischenwerte am GPIO können nicht direkt realisiert werden, sondern nur mit einem Trick. Wenn bei LEDs immer nur für kurze Zeit der Strom fließt und dieses sehr oft hintereinander wiederholt, so nimmt unser Auge es so wahr, als würde die LED etwas schwächer leuchten. Dieses nennt man Pulsweitenmodulation.

In der unteren Zeichnung ist der zeitliche Verlauf der Spannung dargestellt: Bei 50 Hz wird die Zeitspanne, in der der Strom an und abgeschaltet wird 50 mal in der Sekunde wiederholt.





Dieses lässt sich in Python mit Hilfe von PWMLED nutzen. In diesem Beispiel wird die LED zuerst mit 50% Helligkeit, dann mit voller und dann mit 30% Helligkeit angesteuert.

```
from gpiozero import PWMLED
from time import sleep

pled = PWMLED(18)
pled.value = 0.5 # 50% Helligkeit
pled.on()
sleep(1)
pled.value = 1
sleep(1)
pled.value = 0.3
sleep(1)
pled.off()
```

## Skill-Card: RGB-LED

### Benötigtes Material

- RGB-LED

### Benötigte Skill-Cards

- Helligkeit einer LED

Eine RGB-LED besteht aus drei einzelnen LEDs in den Farben rot, grün und blau, die in einem Gehäuse zusammengefasst sind. Werden diese drei Farben gemischt, erhält man weißes Licht. Dadurch, dass man die Intensität(Helligkeit) der einzelnen LEDs steuert, kann man alle anderen Farben darstellen.

Angeschlossen wird die RGB-LED einmal mit der Masse und jede Farbe mit einem Ausgabepin am GPIO des Raspberry Pi. Welcher Pin welche Farbe ist, muss dem Datenblatt der LED entnommen werden. In der Regel ist der längste Pin bei dieser Form der LED der Massenanschluss. Wie bei einer normalen LED muss auch hier auf passende Vorwiderstände geachtet werden, die zwischen Pin am GPIO und dem Pin für die jeweilige Farbe angeschlossen werden müssen.

In Python lässt sich am besten mit der RGBLED arbeiten, in der alle drei LEDs zusammengefasst werden. Für die Farben müssen in der Reihenfolge rot, grün und blau die Nummer der GPIO-Pin angegeben werden. Die einzelnen Farbanteile können in Werten zwischen 0 und 1 angegeben werden.

```
from gpiozero import RGBLED
```

```
from time import sleep

led = RGBLED(16,20,21)
led.color = (1, 1, 0) # LED leuchtet gelb
sleep(1)
led.color = (1, 0.5, 0) # LED leuchtet orange
sleep(1)
led.off()
```

## Skill-Card: Motor

### Benötigtes Material

- Motor
- Motortreiber

### Benötigte Skill-Cards

- Anschluss Steckbrett

Die meisten Elektromotoren benötigen eine höhere Spannung und Stromstärke, als sie der Raspberry Pi an den schaltbaren GPIO-Pins liefern kann. Auch wird die Laufrichtung des Motors durch die Polung der Stromrichtung beeinflusst. Die Polung der GPIO-Pins lässt sich aber nicht ändern. Deshalb muss man einen Motortreiberchip wie z. B. den L293D einsetzen, mit dem man bis zu zwei Motoren steuern kann.

An den Motortreiber wird einmal die Strom-/Spannungsquelle für den Motor angeschlossen, sowie die beiden Pole der Motoren. Zur Steuerung gibt es für jeden Motor drei Pins, die mit dem GPIO verbunden werden müssen. Einer für jede Richtung (Forward/Backward), sowie ein weiterer (Enable) mit dem man die Geschwindigkeit regeln kann. Des weiteren muss auch die Masse des GPIO mit dem Chip verbunden werden.

Im `gpiozero`-Paket gibt es zur Steuerung den `Motor`, dem man die beiden Pins für Forward und Backward mit angeben muss. Wenn man den Anschluss für Enable nicht direkt mit 3,3 V verbunden hat, lässt sich dieser als zusätzlicher Parameter angeben. Nur in dem Fall ist es möglich, die Geschwindigkeit des

Motors zu regeln, der ansonsten immer mit voller Leistung arbeiten würde.

```
from gpiozero import Motor
from time import sleep

motor = Motor(16,20, enable = 21)
motor.forward(0.5) # Motor läuft mit halber
    Geschwindigkeit
sleep(2)
motor.backward() # Motor läuft mit voller
    Geschwindigkeit
sleep(1)
motor.stop()
```

## Skill-Card: Schrittmotor

### Benötigtes Material

- Schrittmotor
- Motortreiber

### Benötigte Skill-Cards

- Motor

Ein normaler Elektromotor für Gleichstrom bewegt sich solange, wie er von außen Strom bekommt. Ein Schrittmotor dreht sich hingegen nur um eine gewisse Stellung weiter. Er hat keinen Kommutator, der die Stromrichtung in den Spulen und damit das Magnetfeld ändert. Stattdessen besitzt der Schrittmotor in der einfachsten Bauform zwei Spulenpaare, bei denen die Stromrichtung von außen gesteuert werden muss. Dabei kann die Achse des Motors immer um  $90^\circ$  gedreht werden.

Will man einen Schrittmotor einmal um  $360^\circ$  drehen, so müssen folgende Schritte durchgeführt werden:

- Spulenpaar A kurz Strom für vorwärts
- Spulenpaar B kurz Strom für vorwärts
- Spulenpaar A kurz Strom für rückwärts
- Spulenpaar B kurz Strom für rückwärts

Um aus dieser Stellung genau wieder zurück zu gehen muss die Reihenfolge geändert werden:

- Spulenpaar A kurz Strom für rückwärts
- Spulenpaar B kurz Strom für vorwärts
- Spulenpaar A kurz Strom für vorwärts
- Spulenpaar B kurz Strom für rückwärts

Angeschlossen werden muss ein Schrittmotor an einen Motortreiber so, dass das eine Spulenpaar den ersten Motor darstellt und das andere Spulenpaar den zweiten. Da die Geschwindigkeit im Programm durch die Wartezeit zwischen den Schritten beeinflusst werden kann, sollten die Enable Pin direkt mit 3,3 V verbunden werden.

Im Pythonpaket `gpiozero` gibt es für den Schrittmotor keine Elemente, so dass man hier selber aus zwei Motoren einen Schrittmotor bauen muss. Alternativ gibt es unter <https://kurzelinks.de/6zg4> eine Bibliothek, die man dafür nutzen kann. Diese lässt sich mit einem ULN2003-Motortreiber nutzen. Die Eingänge `In1` bis `In4` sind dann die Pins, die im Beispielprogramm angesteuert werden.

## Skill-Card: Servomotor

### Benötigtes Material

- Servomotor

### Benötigte Skill-Cards

- Anschluss Steckbrett

Ein Servomotor stellt eine besondere Art von Motoren dar. Bei diesen Motoren kontrolliert eine Elektronik die Position der Drehachse. Über diese wird auch gesteuert, in welche Position sich der Motor drehen soll. So werden sie z. B. bei der Lenkung Modellfahrzeugen eingesetzt.

In Verbindung mit dem Raspberry Pi wird oft ein SG90 Servomotor genutzt. Dieser hat drei Anschlüsse: Rot für 5 V und braun oder schwarz für Masse. Über den dritten Anschluss (gelb, orange oder braun) wird der Motor gesteuert.

In Python lässt sich ein Servomotor mit den `gpiozero` Paket nutzen. Im folgenden Beispiel ist die Steuerung am Pin 17 angeschlossen.

```
from gpiozero import Servo

myCorrection=0.45
maxPW=(2.0+myCorrection)/1000
minPW=(1.0-myCorrection)/1000

servo = Servo(17, min_pulse_width=minPW,
              max_pulse_width=maxPW)
```



Durch den Korrekturwert<sup>a</sup> lässt sich einstellen in welchem Winkelbereich der Servomotor sich drehen soll. Ohne diese Korrektur dreht er sich nur ungefähr 45 Grad von der Mitte aus zu jeder Seite anstatt die gewünschten 90.

Das Einstellen der Winkel lässt sich dann über folgendes verwirklichen:

```
servo.mid() # Zur Mitte  
servo.min() # Maximale rechten Stellung  
servo.max() # Maximale linken Stellung  
servo.value = 0.2 # Zwischenwert
```

Als Zwischenwerte können alle Zahlen zwischen 1 für Links und -1 für Rechts angegeben werden. Die 0 ist dabei die Mittelstellung.

---

<sup>a</sup>Aus <https://kurzelinks.de/g7q8>

## Skill-Card: AD-Wandler

### Benötigtes Material

- AD-Wandler

### Benötigte Skill-Cards

- Anschluss Steckbrett

Die GPIO-Pins als Eingang am Raspberry Pi sind digital und können daher nur zwischen zwei Zuständen wechseln: Entweder liegt keine Spannung (0 V) oder eine Spannung (3,3 V) an. Werte dazwischen werden einem dieser beiden Möglichkeiten zugeordnet. Wenn man diese Zwischenwerte aber messen möchte muss man über einen AD-Wandler gehen, der die analoge Spannung in einen digitalen Wert umwandelt.

Der MCP3208 ist ein solcher AD-Wandler, der über das SPI-Protokoll seine Werte an den Raspberry Pi weitergeben kann. Dazu muss er aber an die Pins des GPIO angeschlossen werden, die für SPI vorgesehen sind. Außerdem muss in der Konfiguration das SPI-Protokoll aktiviert werden.

Am MCP3208 können an bis zu acht Kanälen verschiedene analoge Werte gemessen werden.

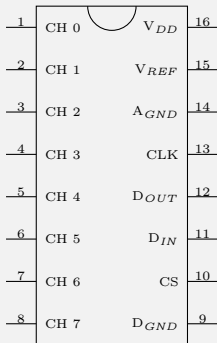
In das `gpiozero`-Paket liefert dafür auch eine passende Klasse `MCP3208` mit der diese Eingänge abgefragt werden können. Dazu muss aber zuerst der Eingang/Kanal (0 bis 7) gewählt werden. Die Werte die man erhält liegen zwischen 0 und 1. Im

folgenden Beispiel werden die ersten beiden Kanäle nacheinander abgefragt:

```
from gpiozero import MCP3208

ad = MCP3208()
ad.channel = 0 # Wählt den ersten Kanal
print(ad.value) # Zwischen 0 und 1
ad.channel = 1
print(ad.value)
```

Die Pin-Belegung des MCP3208 und wie diese mit dem GPIO verbunden werden müssen steht hier:



A/D	GPIO
9 (D <sub>GND</sub> )	9 (GND)
10 (CS)	24 (CE0)
11 (D <sub>IN</sub> )	19 (MOSI)
12 (D <sub>OUT</sub> )	21 (MISO)
13 (CLK)	23 (SCLK)
14 (A <sub>GND</sub> )	9 (GND)
15 (V <sub>REF</sub> )	1 (3,3 V)
16 (V <sub>DD</sub> )	1 (3,3 V)

## Skill-Card: RFID-Leser

### Benötigtes Material

- RFID-Leser

### Benötigte Skill-Cards

- Anschluss Steckbrett

Mit dem RC522 wird ein Modul angeboten, mit dem man RFID (Radio Frequency Identification) Karten auslesen kann. Dieses Modul funktioniert über das SPI-Protokoll, dass in der Konfiguration des Raspberry Pi aktiviert werden muss. Durch das SPI ist auch vordefiniert, welche Pins gesetzt werden müssen.

Auf dem Raspberry Pi muss zuvor aber noch ein Paket für Python installiert werden. Dieses geht am besten durch folgenden Aufruf:

```
pip3 install pi-rc522
```

Nun kann man mit folgendem Programm die IDs der RFID-Karten auslesen, die vor den RFID-Reader gehalten werden.

```
from pirc522 import RFID
from time import sleep
import RPi.GPIO as GPIO

cardReader = RFID(pin_mode=GPIO.BCM)

while True:
    (error, data) = cardReader.request()
```

```
if not error:
    print ("Karte_gefunden")
    (error, uid) = cardReader.anticoll()
if not error:
    print ("UID:_" + str(uid[0]) + "," +
          str(uid[1]) + "," + str(uid[2]) + "," +
          str(uid[3]))
    sleep(1)
```

## Skill-Card: 4x4 Tastenfeld

### Benötigtes Material

- Tastenfeld

### Benötigte Skill-Cards

- Anschluss Steckbrett

Ein 4x4-Tastenfeld hat acht Anschlüsse, von denen vier als Eingänge und 4 als Ausgänge genutzt werden. Die einen werden repräsentieren die Spalten, die anderen die Zeilen.

Zur Abfrage wird nacheinander immer an einen Eingang eine Spannung angelegt und dann alle Ausgänge abgefragt. Durch das Drücken einer Taste liegt dann an einem Ausgang eine Spannung an. Aus der Kombination von Eingang und Ausgang kann man dann die Taste identifizieren.

Zur Einbindung in Python kann eine Vorlage genutzt werden, die unter <https://kurzlinks.de/o4gh> zu erreichbar ist. Diese muss man im entsprechenden Verzeichnis speichern. Die Vorlage ist eigentlich noch nicht fertig, kann aber schon genutzt werden.

Damit lässt sich dann das Tastenfeld wie folgt nutzen. Hierbei ist zu beachten, dass immer alle Tasten angegeben werden, die aktuell gedrückt sind, bzw. auch eben keine.

```
from keypad_gpiozero import MatrixKeypad
```

```
kp = MatrixKeypad(
```

```
    cols=[19, 26, 16, 20],  
    rows=[21, 14, 15, 18],  
    labels=["123A", "456B", "789C", "*0#D"],  
)  
  
for taste in kp.value:  
    print (taste)
```

Die hier im Beispiel angegebenen Pins müssen so abgeändert werden, wie das eigene Tastenfeld angeschlossen ist.