

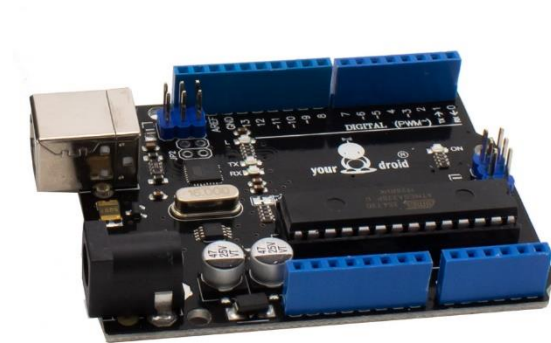
## yourDroid UNO R3 RFID Starter Kit



## Inhaltsverzeichnis

0.	Erste Schritte .....	3
1.	Grundlagen & Blink Beispiel .....	15
2.	Lauflicht mit 8 LEDs .....	21
3.	Taster .....	26
4.	DHT11 Temperatursensor .....	30
5.	RGB LED-Modul .....	35
6.	Buzzer .....	38
7.	LED mit Potentiometer dimmen .....	41
8.	Schrittmotor .....	44
9.	DS1302 Echtzeituhrmodul (RTC) .....	51
10.	Servomotor .....	56
11.	Neigungsschalter .....	60
12.	Fotowiderstand .....	64
13.	LM35 Temperatursensor .....	68
14.	Relais Modul .....	72
15.	Flammensensor .....	75
16.	Keypad Modul .....	78
17.	Joystick Modul .....	82
18.	Wasserstandsensor .....	85
19.	Soundmodul .....	88
20.	LED-matrix .....	92
21.	MFRC522 RFID Modul .....	96
22.	Infrarot-Fernbedienung .....	99
23.	Acht LEDs mit 74HC595 ansteuern .....	104

# 0. Erste Schritte



## Arduino IDE installieren


Die Arduino Integrated Development Environment ist die Entwicklungsumgebung und Software der Arduino Plattform.

In dieser Lektion lernen Sie Ihren Computer einzurichten, um Arduino Boards nutzen zu können.

Die Arduino Software erlaubt es Ihnen Arduinos und andere Entwicklungsplatinen zu programmieren. Die Software ist verfügbar für Windows, Mac und Linux. Der Installationsprozess ist für alle Plattformen unterschiedlich.

## Arduino IDE herunterladen

**Schritt 1:** <https://www.arduino.cc/en/software> aufrufen



### Arduino IDE 1.8.19

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.


Refer to the **Getting Started** page for Installation instructions.

**SOURCE CODE**

Active development of the Arduino software is **hosted by GitHub**. See the instructions for **building the code**. Latest release source code archives are available **here**. The archives are PGP-signed so they can be verified using **this** gpg key.

#### DOWNLOAD OPTIONS

**Windows** Win 7 and newer  
**Windows** ZIP file

**Windows app** Win 8.1 or 10 

**Linux** 32 bits  
**Linux** 64 bits  
**Linux** ARM 32 bits  
**Linux** ARM 64 bits

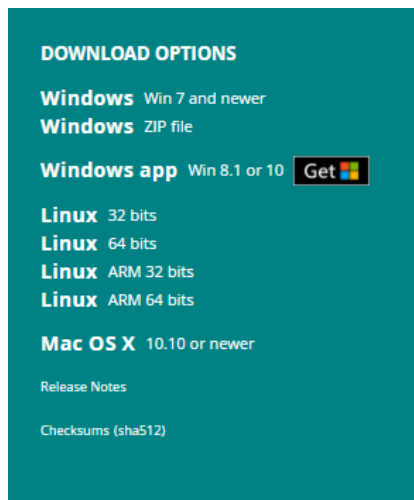
**Mac OS X** 10.10 or newer

[Release Notes](#)

[Checksums \(sha512\)](#)

*Auf dieser Webseite finden Sie immer die aktuelle Version. Die aktuelle Version ist vermutlich neuer als die aus dieser Anleitung.*

## Schritt 2: Passenden Installer herunterladen



„Win 7 and newer“ auswählen, falls Sie nicht die App aus dem Microsoft Store installieren möchten.

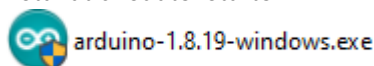
## Schritt 3: Download Starten



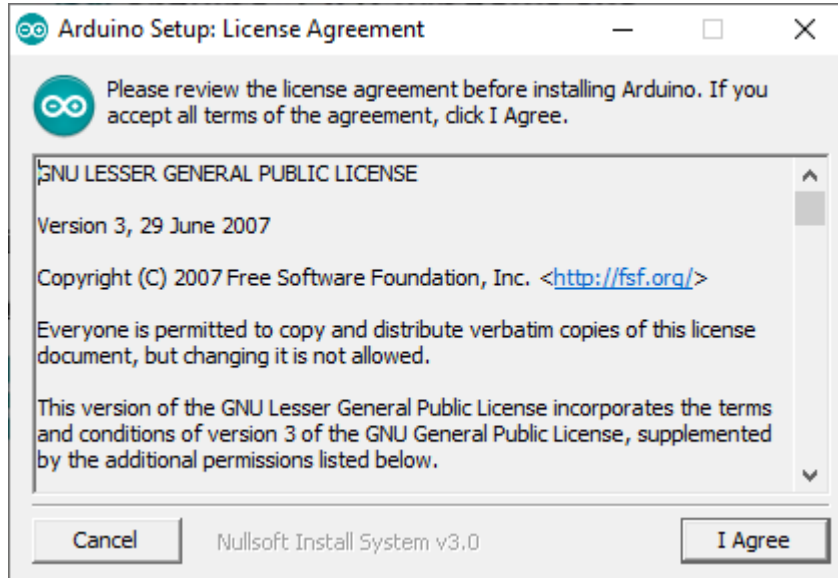
„Just Download“ auswählen.

## Installation unter Windows

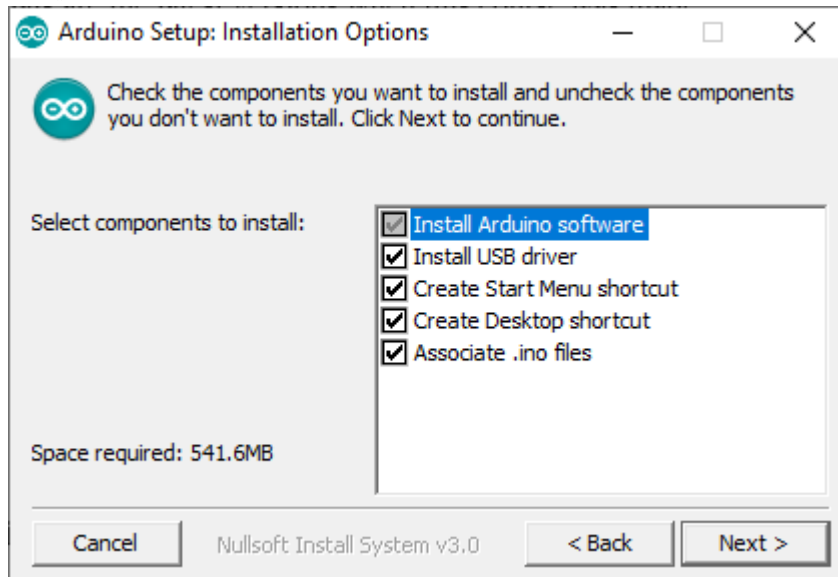
### 1. Installationsdatei starten



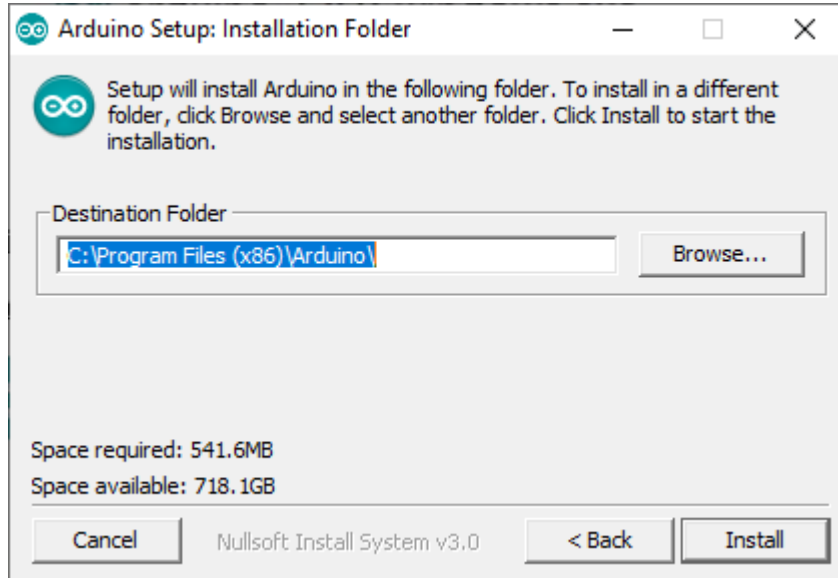
## 2. Lizenzbedingungen akzeptieren



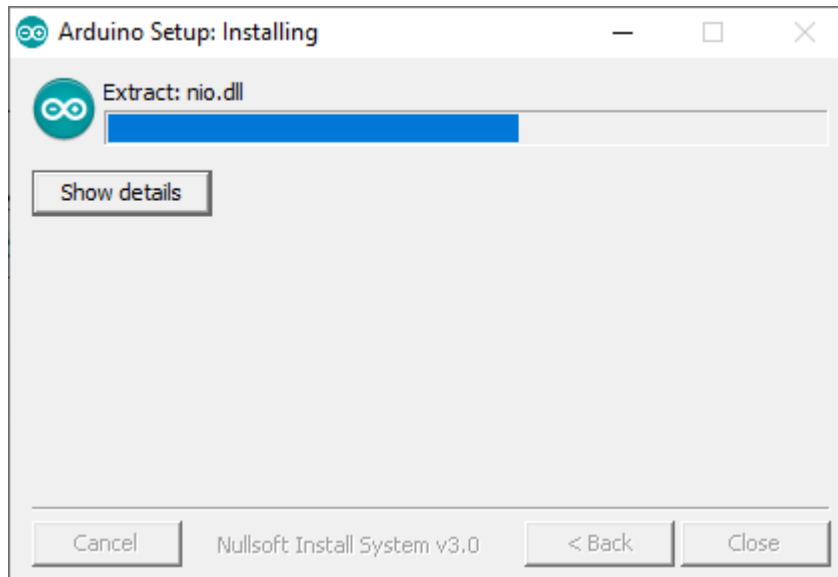
## 3. Auf „Next“ klicken



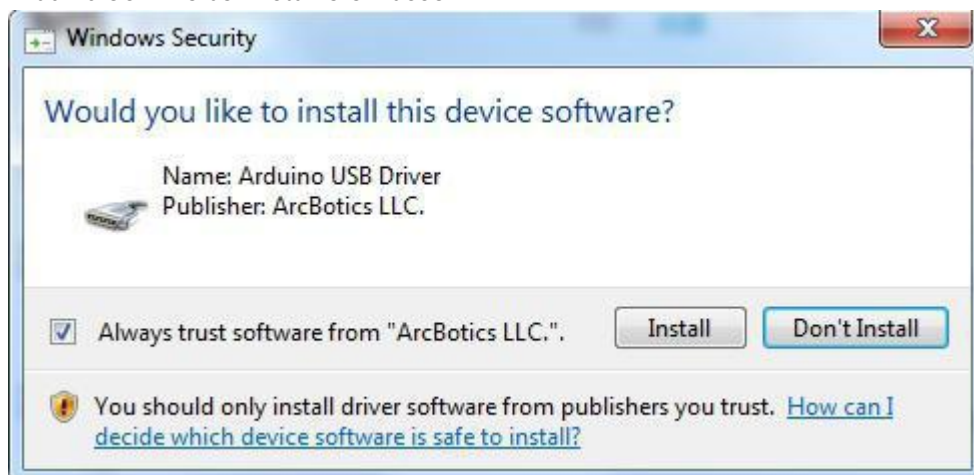
## 4. Installationspfad auswählen oder „Install“ drücken



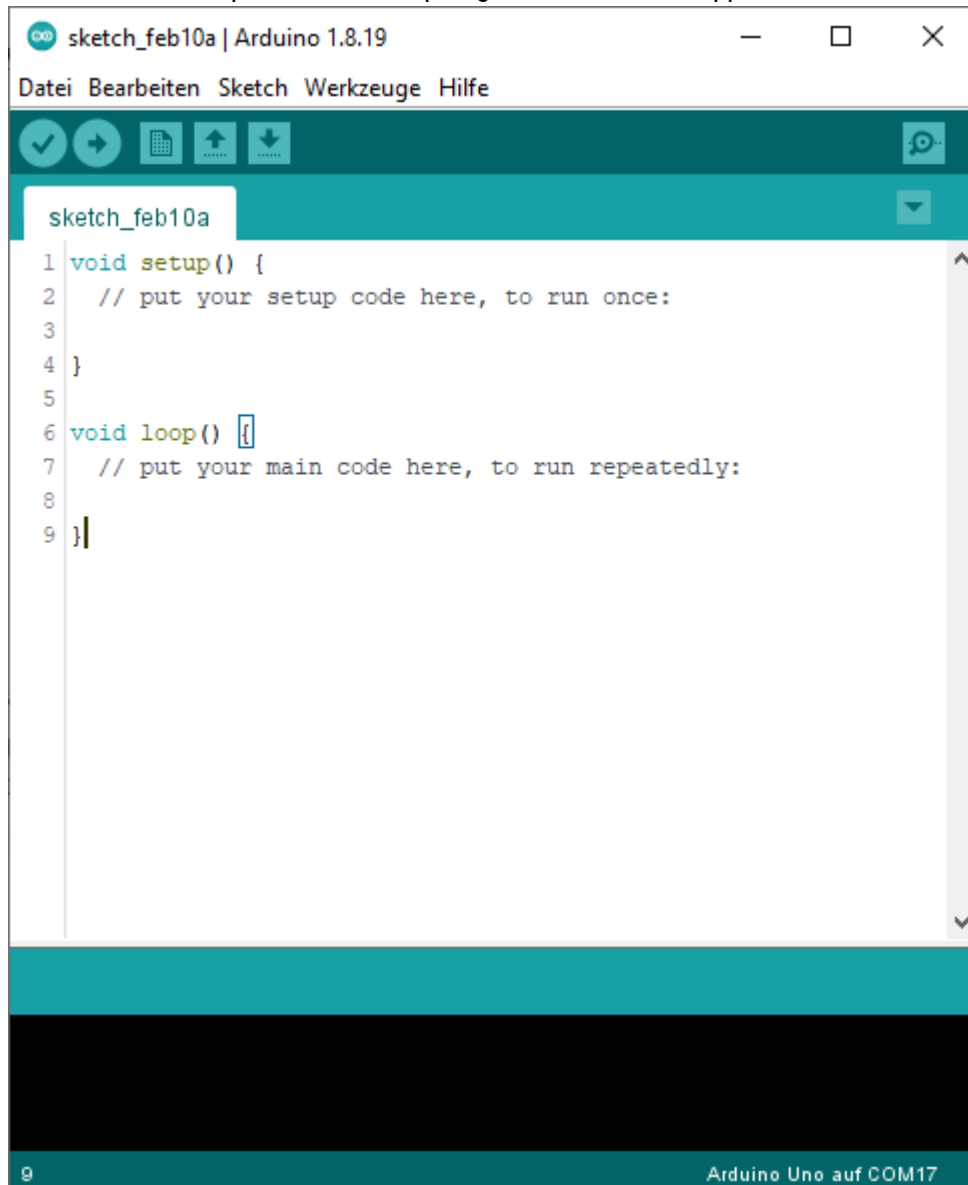
## 5. Installation abwarten



## 6. Arduino USB-Treiber installieren lassen



7. Zurück zum Desktop und die Verknüpfung der Arduino IDE doppelklicken



8. Das war die Installation unter Windows.

### Installation unter Linux

1. Package extrahieren
2. Install Script starten  
Entweder im entpackten Archiv die „install.sh“ starten oder im Terminal zum Dateipfad navigieren und „./install.sh“ eingeben
3. Die Arduino IDE kann nun gestartet werden. Auf dem Desktop finden Sie die Verknüpfung.

*Empfohlen: Wenn Sie ein Ubuntu System nutzen, können Sie stattdessen die Installation über das Software-Center von Ubuntu durchführen.*

## Arduino Installation unter MacOS

1. Die Installationsdatei ist im Zip-Format. Wenn Sie Safari nutzen, wird der Inhalt automatisch entpackt. Die Installationsdatei doppelklicken, falls noch nicht bereits installiert, werden Sie aufgefordert die Java Runtime Library zu installieren.
2. Die Arduino IDE kann nun gestartet werden.



## Bibliotheken hinzufügen und den seriellen Monitor öffnen

### Installieren zusätzlicher Arduino Bibliotheken

Sobald Sie sich mit der Arduino IDE vertraut gemacht haben und die integrierten Beispiele ausprobiert haben, können Sie mit Bibliotheken mehr Potenzial aus Ihrem Arduino Board schöpfen.

### Was sind Bibliotheken?

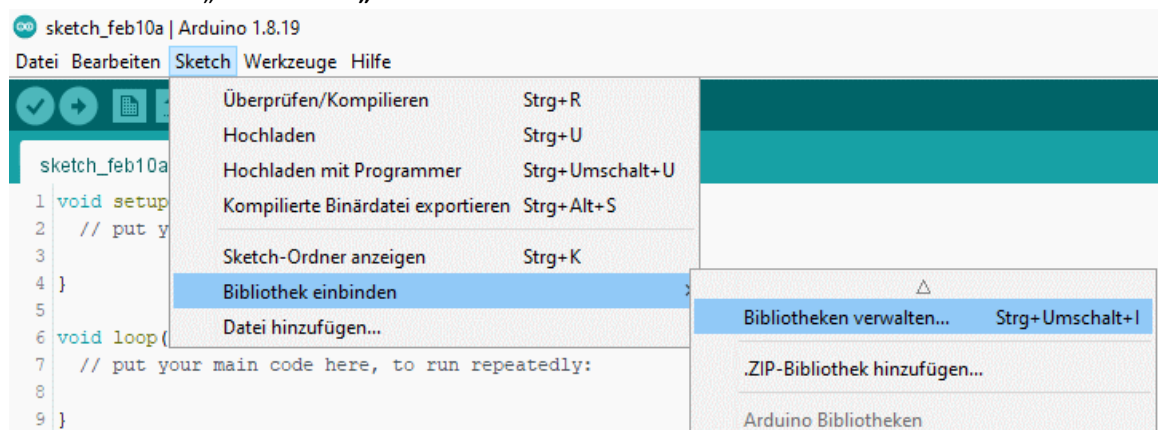
Bibliotheken sind Codesammlungen um Sensoren, Module, Displays oder Funktionen wesentlich einfacher programmieren zu können. Zum Beispiel vereinfacht die integrierte LiquidCrystal.h Bibliothek die Kommunikation zu LCD Displays, um Zeichenfolgen unkompliziert ausgeben können. Es gibt tausende zusätzliche Bibliotheken im Internet kostenfrei zum Herunterladen. Die integrierten Bibliotheken besitzen Beispiele, die Sie unter „Datei – Beispiele“ aufrufen können. Zusätzliche Bibliotheken müssen heruntergeladen und installiert werden.

### Wie installiere ich weitere Bibliotheken?

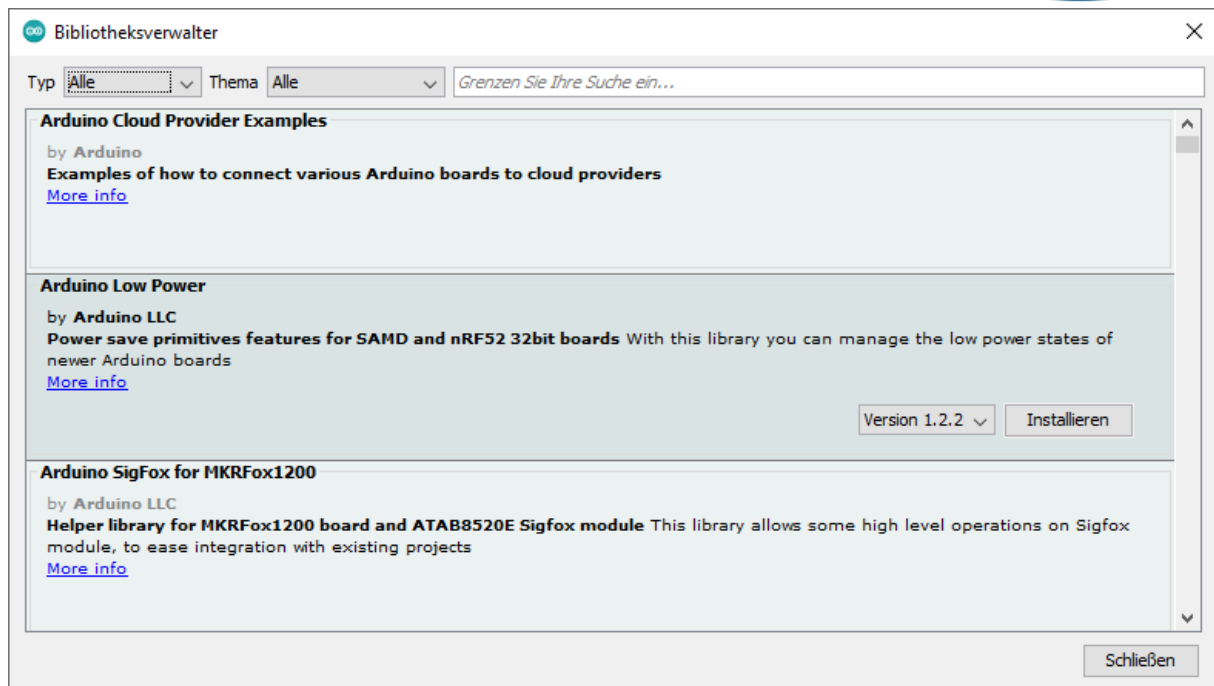
Für die Installation von weiteren Bibliotheken gehen wir hier auf die verschiedenen Möglichkeiten ein.

### Bibliotheken mit der Arduino IDE installieren

Seit Version 1.8.0 gibt es den praktischen Library Manager mit dem Sie im Handumdrehen Bibliotheken suchen, herunterladen und installieren können. Öffnen Sie die Arduino IDE und klicken Sie im im Reiter „Sketch“ auf **„Bibliotheken verwalten...“**



Es öffnet sich der Library Manager, mit dem Sie Bibliotheken installieren, aktualisieren oder entfernen können. Gelegentlich werden Sie in der Arduino IDE unten rechts eine Meldung erhalten, dass für installierte Bibliotheken Aktualisierungen verfügbar sind.

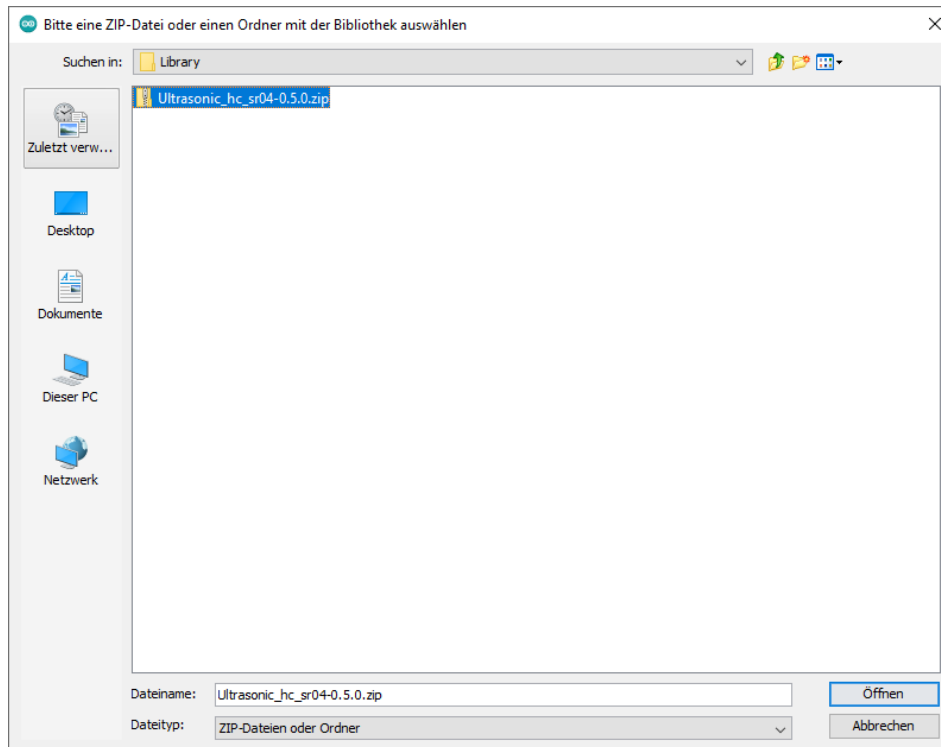


### .zip Bibliothek importieren

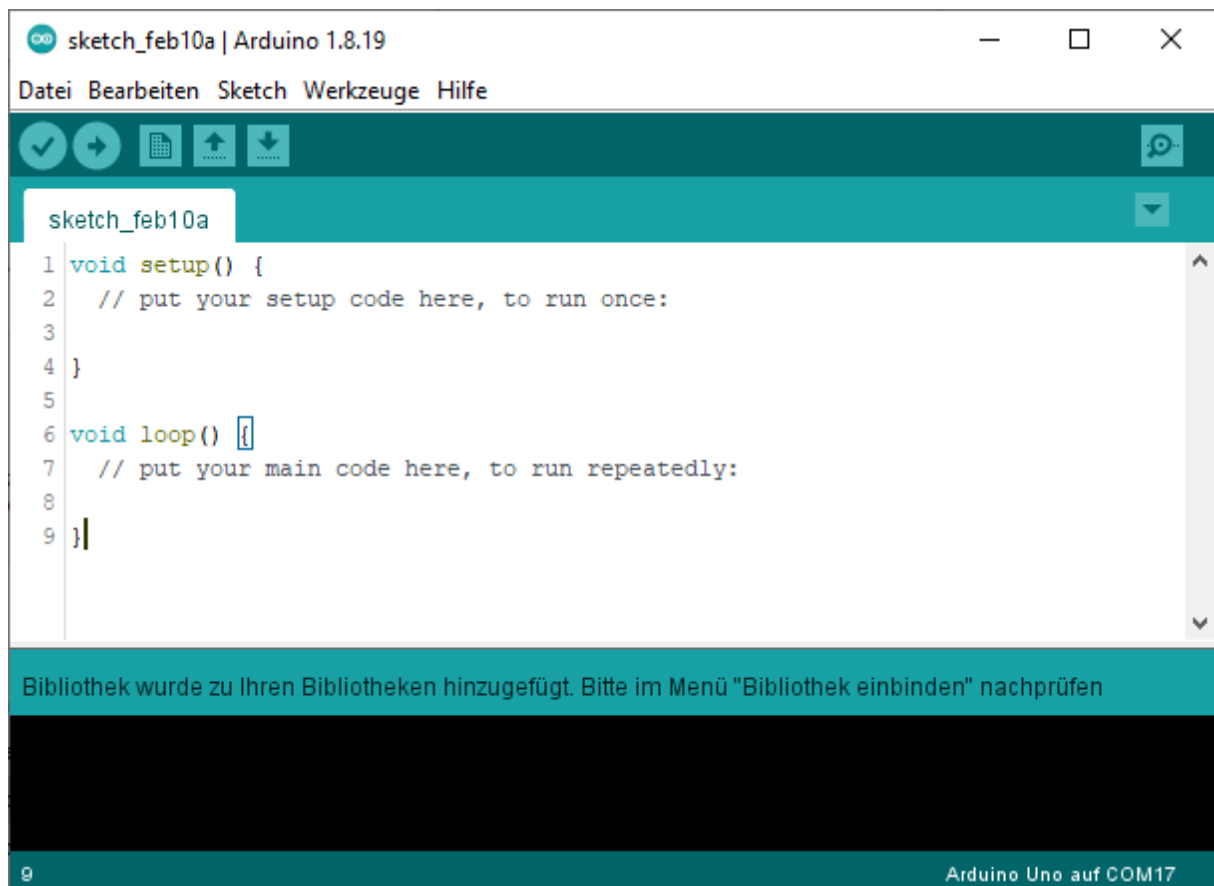
Es kann vorkommen, dass eine Bibliothek nicht im Bibliotheksverwalter zu finden ist. In diesem Fall muss man auf eine externe Quelle zurückgreifen und die Bibliothek importieren. Hierzu in der Arduino IDE unter „Sketch“ zu „Bibliothek einbinden“ navigieren und **„ZIP Bibliothek hinzufügen...“** anklicken.



Sie werden aufgefordert mit dem Dateixplorer eine ZIP Bibliothek zu öffnen. Navigieren Sie zum entsprechenden Dateipfad und wählen Sie eine ZIP Bibliothek aus.



Wenn die Bibliothek erfolgreich eingebunden wurde, erscheint im Hauptfenster folgende Nachricht:



Nun Können Sie, wie aufgefordert, unter „Sketch -> Bibliotheken einbinden“ nach unten scrollen um zu schauen ob die Bibliothek aufgelistet wird.

**Hinweis: Die Beispiele unter „Datei – Beispiele“ tauchen erst nach einem Neustart der Arduino auf.**

### Bibliotheken manuell installieren

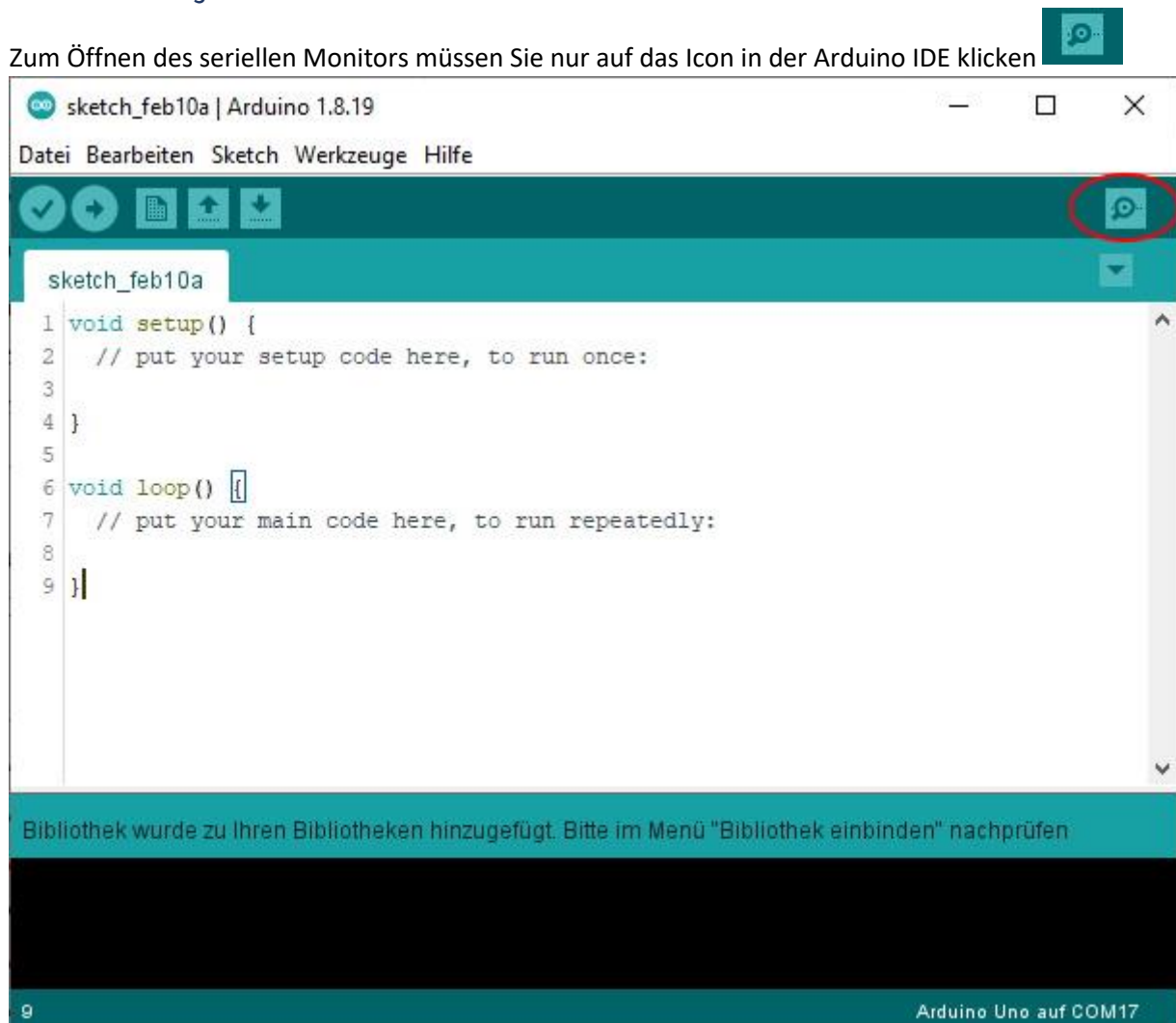
Um Bibliotheken manuell zu installieren, müssen Sie die .ZIP Datei entpacken und den Ordner nach „Dokumente/Arduino/Libraries“ kopieren. Wichtig ist, dass der Ordner wie die Bibliothek heißt und eine .cpp und .h Datei enthält. Nach einem Neustart befindet sich die Bibliothek nun ebenfalls unter „Sketch – Bibliotheken einbinden“ in der Liste.

### Serieller Monitor

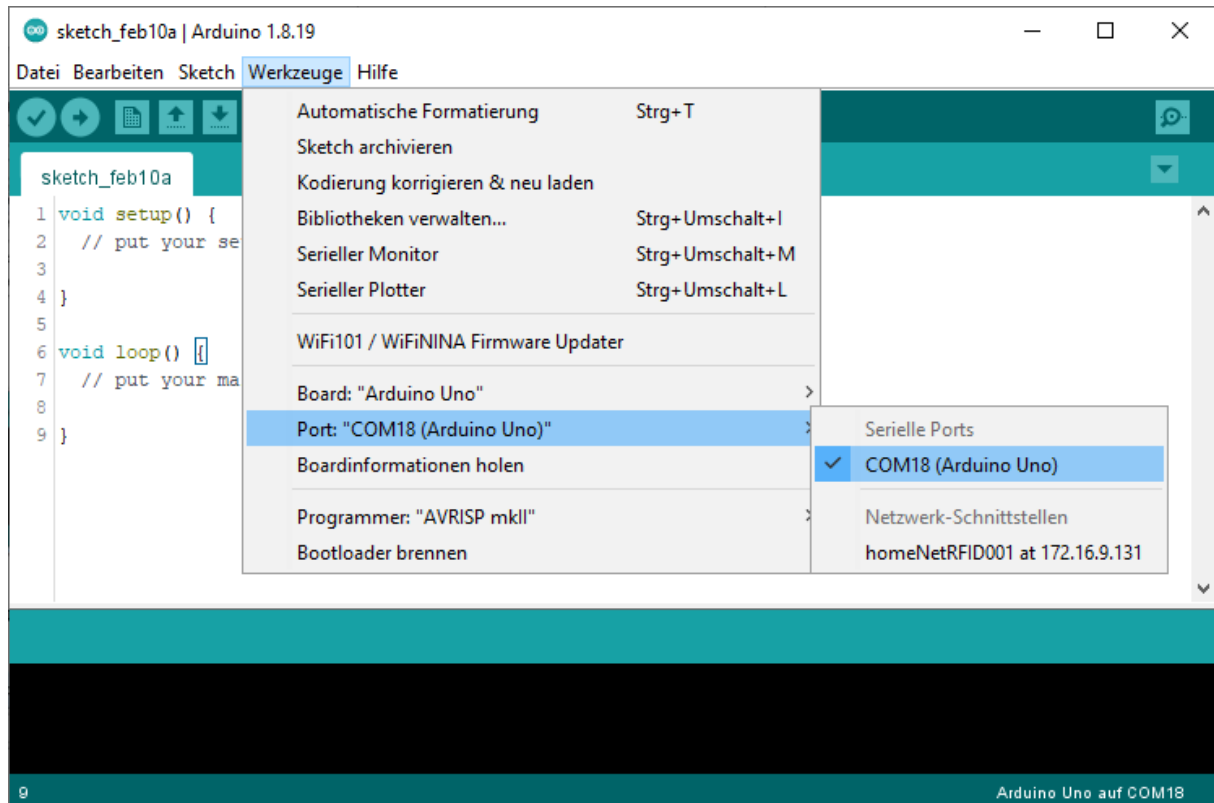
Der serielle Monitor ist ein wichtiger Teil der Arduino IDE, um mit angeschlossenen Microcontrollern kommunizieren und sich auf Fehlersuche begeben zu können. Mit diesem Werkzeug können Sie ganz schnell Messwerte ausgeben lassen, dem Arduino Befehle erteilen oder Programmfunktionen auf Fehler überprüfen.

### Eine Verbindung herstellen

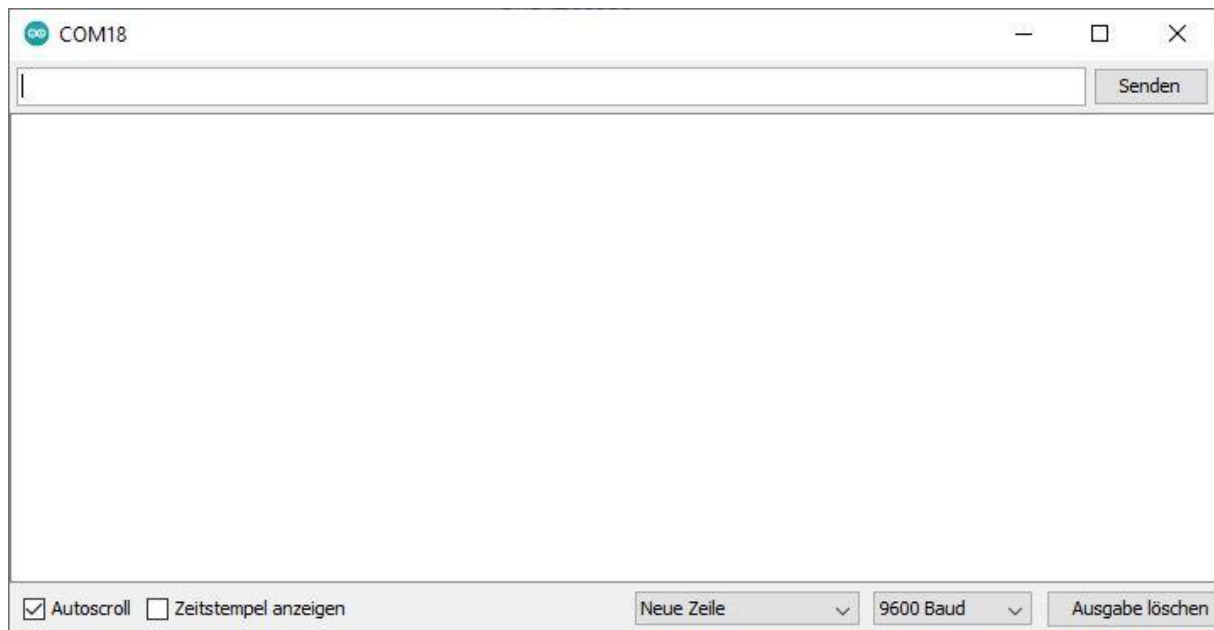
Zum Öffnen des seriellen Monitors müssen Sie nur auf das Icon in der Arduino IDE klicken



Die Port-Auswahl vom seriellen Monitor funktioniert wie beim Hochladen von Arduino Codes.

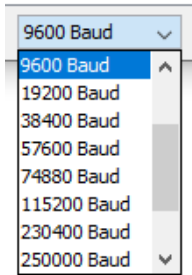


Einmal geöffnet sieht er ungefähr so aus:



## Einstellungen des seriellen Monitors

Der serielle Monitor der Arduino IDE hat nur wenige Einstellungen, die aber für die meisten Anwendungen ausreichen dürften. Die wichtigste Einstellung ist die baud rate.



Mit der baud rate wird die **Datenrate in Bit pro Sekunde** für die serielle Datenübertragung festgelegt. Wenn die baud rate falsch eingestellt ist, werden die Daten nicht korrekt angezeigt und die Kommunikation wird nicht funktionieren. **Autoscroll** ist sehr praktisch, kann aber bei Bedarf ausgeschaltet werden.

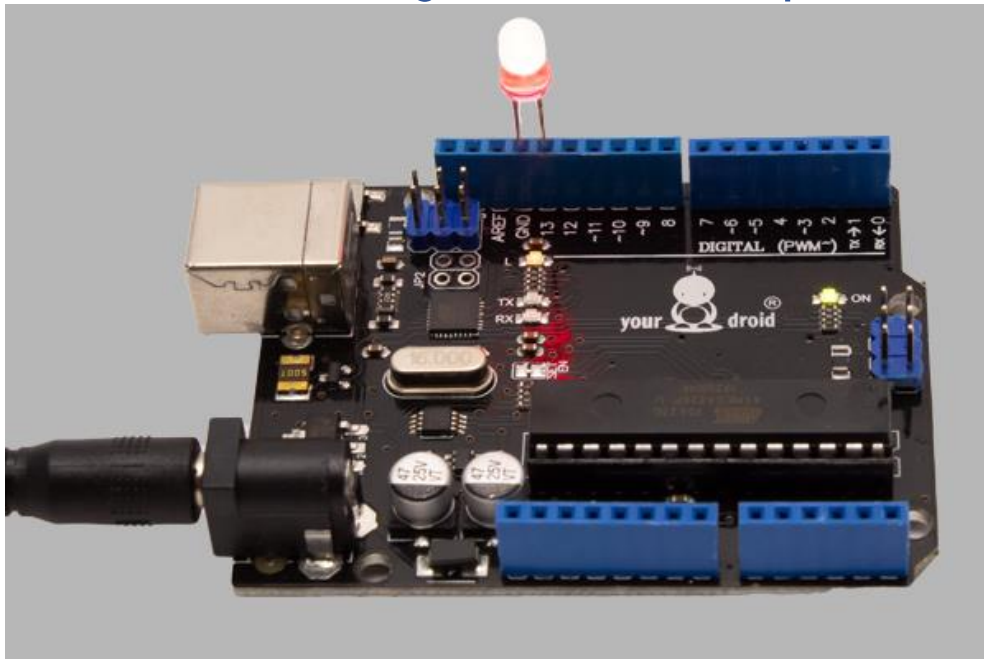
#### *Vorteile vom seriellen Monitor*

Der serielle Monitor ist eine gute und sehr einfache Möglichkeit eine serielle Verbindung mit Ihrem Arduino herzustellen. Wenn Sie sowieso in der Arduino IDE programmieren, liegt es nahe die integrierten Werkzeuge zu verwenden.

#### *Nachteile vom seriellen Monitor*

Durch die dürftigen Einstellungsmöglichkeiten eignet sich der serielle Monitor nicht für fortgeschrittene Anwendungen. Die Grenzen bereits sind schnell erreicht, wenn Ihre Anwendung eine nicht aufgeführte baud rate verlangt.

# 1. Grundlagen & Blink Beispiel



## Grundlagen der Inbetriebnahme: Es blinkt!

Herzlich willkommen zu unserem ersten Arduino Projekt bei dem wir durch ein beliebtes, kleines Beispiel die Grundlagen erläutern, um unser Arduino kompatibles UNO-Board über die offizielle Software anzusteuern.

Für dieses Projekt basierend auf unserem Arduino Starter Kit benötigen wir folgende Komponenten:

- yourDroid Board + USB-Kabel
- 5mm LED

Benötigte Software:

- Arduino Software ([Download auf Arduino.cc](http://arduino.cc))

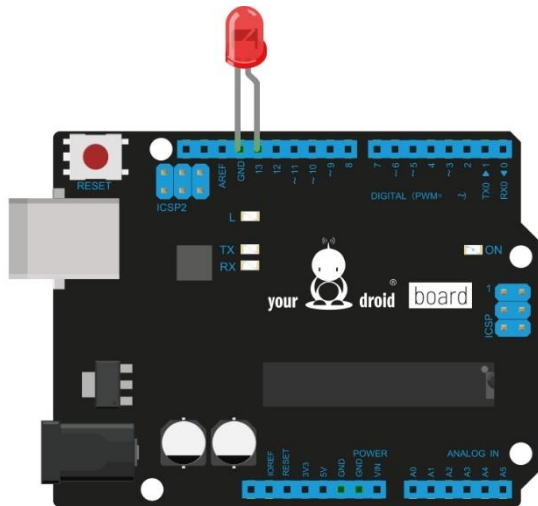
Normalerweise bräuchten wir noch Jumperkabel, Breadboard und einen Widerstand, für den Anfang reicht jedoch unser Arduino kompatibles UNO-Board und eine 5mm LED.

Auf dem Arduino UNO ist typischer Weise bereits eine LED am Pin 13 eingebaut, dieser Pin besitzt dadurch den nötigen Widerstand, um eine LED direkt zu betreiben.

## Anschließen

Wir stecken die LED also einfach mit dem langen Bein (Anode +) in Pin 13 und das kurze Bein (Kathode -) in den GND-Pin daneben.

Das Ganze sieht dann so aus:

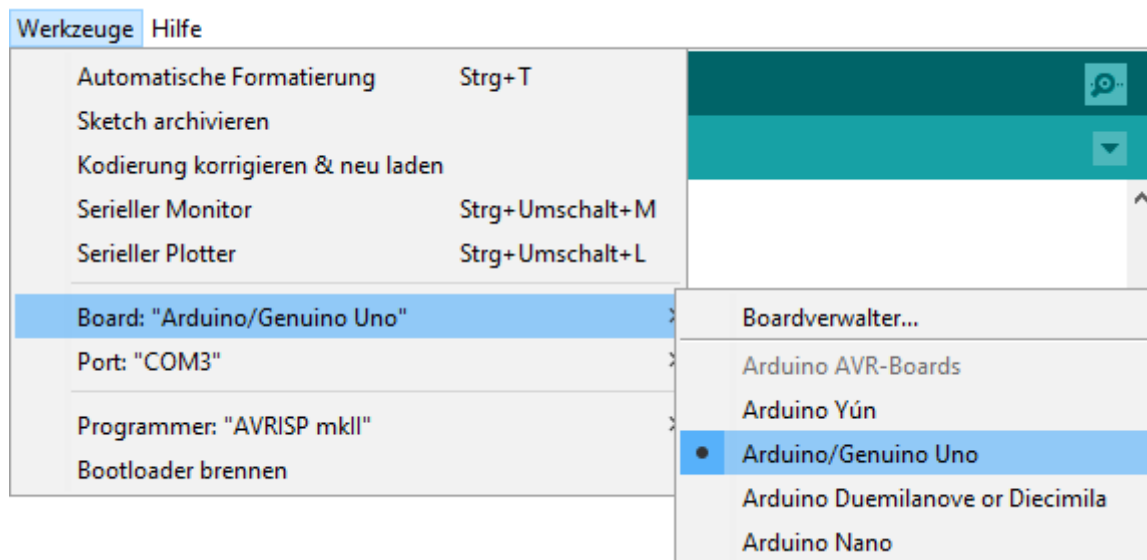


fritzing

## Programmieren

Nachdem wir die Arduino-Software installiert haben, schließen wir das yourDroid Board über USB an den Computer an.

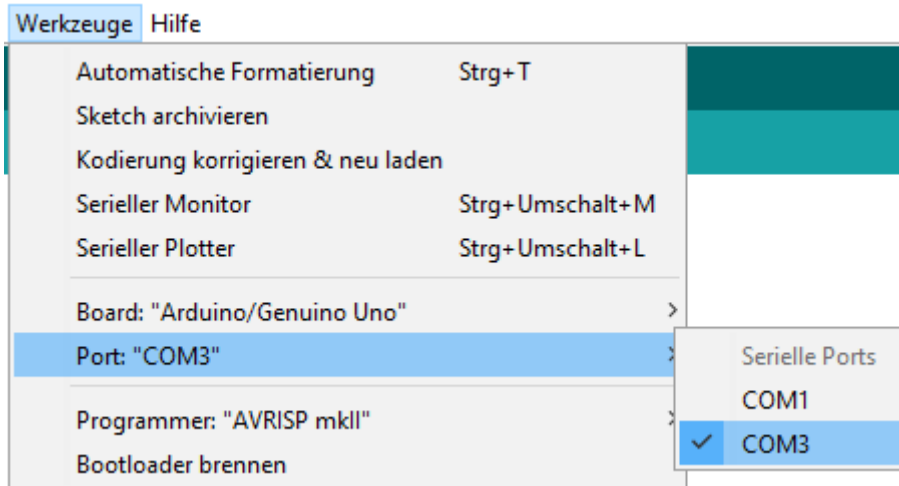
Nun öffnen wir die Arduino-Software und stellen sicher, dass bei "Werkzeuge - Board" der Arduino UNO ausgewählt ist. und der serielle Port richtig eingestellt ist. Unter Windows ist es meistens Port "COM 3" oder höher, denn Port 1-2 sind für das System reserviert.



Als nächstes stellen wir den richtigen Port ein. Unter Windows ist es meistens Port "COM 3" oder höher, denn Port 1-2 sind in der Regel für das System reserviert.

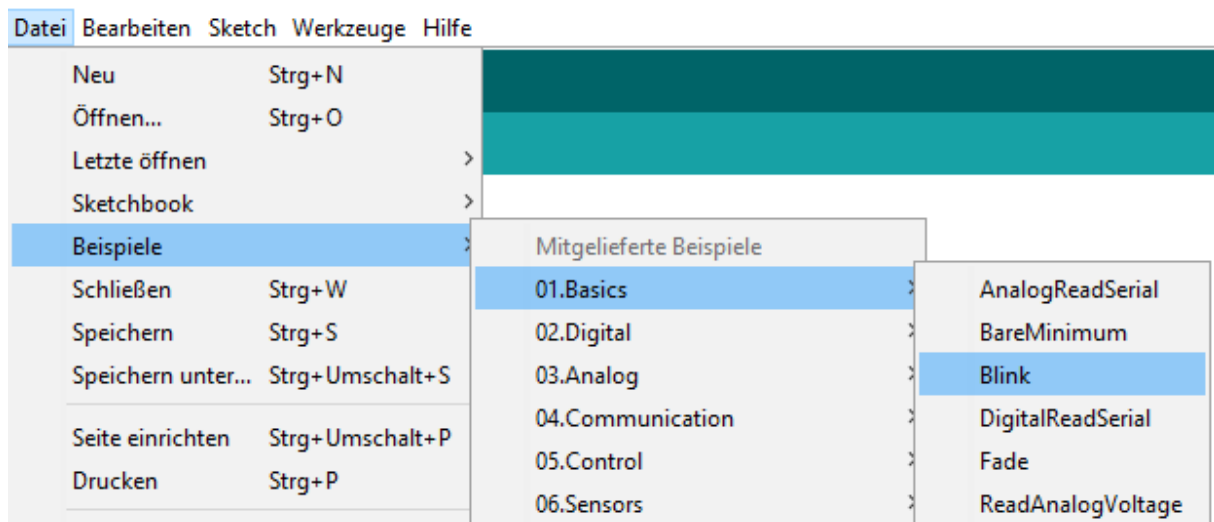
**Achtung: Der richtige Port auf Ihrem System wird vermutlich nicht derselbe wie auf den Screenshots sein!**





Die Arduino-Software liefert einige Programmbeispiele mit, für dieses Projekt benutzen wir einfach das "Blink"-Beispiel.

Für dieses Beispiel klicken wir oben links auf "Datei" und wählen unter dem Reiter "Beispiele - 01.Basics - Blink" aus.



Dann öffnet sich ein neuer Sketch, welcher hier nochmal mit deutschen Kommentaren versehen ist:

```

/*
  Blink
  Schaltet eine LED für 1 Sekunde an und für eine Sekunde aus.
*/

// Die Setup-Funktion wird einmalig nach jedem Start oder Reset
ausgeführt
void setup() {
  // digitaler pin 13 wird als Output deklariert
  pinMode(13, OUTPUT);
}

```

```
// Die Loop-funktion wird immer und immer wieder ausgeführt
void loop() {
  digitalWrite(13, HIGH); // Schaltet die LED an (HIGH ist die
  Spannungsversorgung)
  delay(1000);           // 1000 ms (= 1 Sekunde) Verzögerung
  digitalWrite(13, LOW); // Schaltet die LED aus indem die
  Spannungsversorgung auf LOW gesetzt wird
  delay(1000);           // 1 Sekunde Verzögerung
}
```

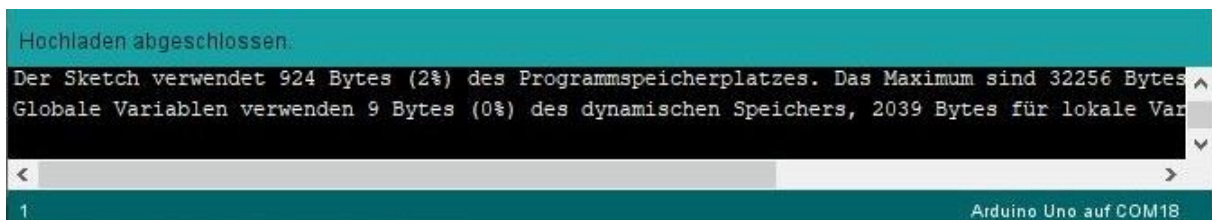
Dieses kleine Programm laden wir nun auf unser Board, indem wir auf die Schaltfläche klicken:



Im unteren Bereich können Sie den Fortschritt des Vorgangs verfolgen. Der Sketch wird zunächst kompiliert, damit der Code in ein geeignetes Format zur Übertragung konvertiert wird.



Im Anschluss wird der Sketch hochgeladen, die anderen LEDs auf dem Board sollten während dem Upload flackern.



## Fehlerbehebung

Die folgende Fehlermeldung bedeutet, dass das Board nicht angeschlossen ist.



Mögliche Fehlerursachen und Lösungen

- **Board nicht angeschlossen.** Manchmal hilft es auch einen anderen USB-Port auszuprobieren.

- **Treiber nicht richtig installiert.** In diesem Fall müssen Sie im Geräte-Manager Ihr Arduino finden und im Fenster „Eigenschaften“ den Treiber deinstallieren und anschließend neu installieren.
- **Falscher Port ausgewählt.** In der Arduino IDE unter „Werkzeuge“ den Port erneut überprüfen.
- **Falsches Board ausgewählt.** In der Arduino IDE unter „Werkzeuge“ das Board erneut überprüfen.



```

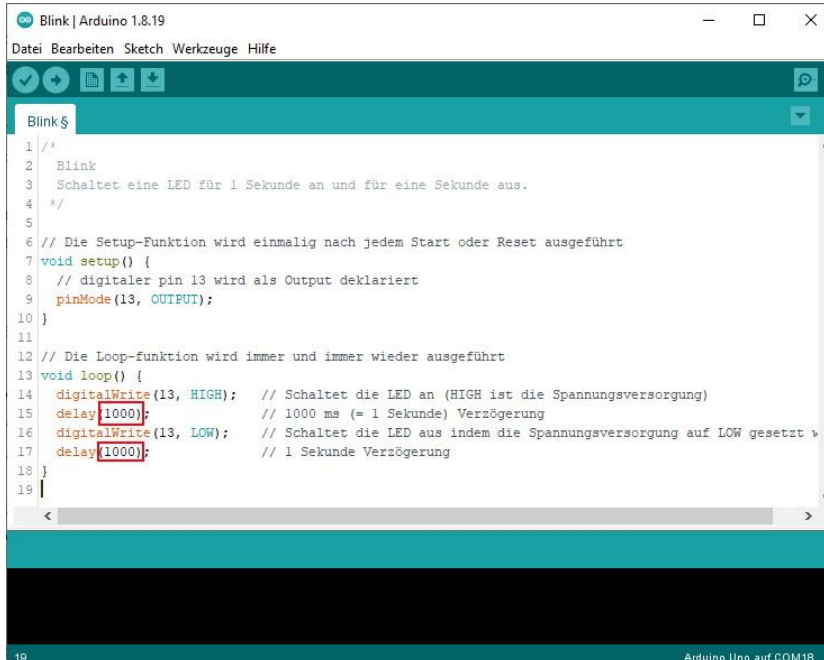
expected ';' before '}' token
^
exit status 1
expected ';' before '}' token
  
```

Diese Fehlermeldung wird jeder früher oder später erhalten. In der Programmiersprache von Arduino muss hinter jeder Zeile, also jedem Befehl ein Semikolon „;“ stehen. Andernfalls bricht die Kompilierung ab und erzeugt diese Fehlermeldung.

**Lösung:** Die entsprechende Zeile wird in der Arduino rot hinterlegt und kann korrigiert werden.

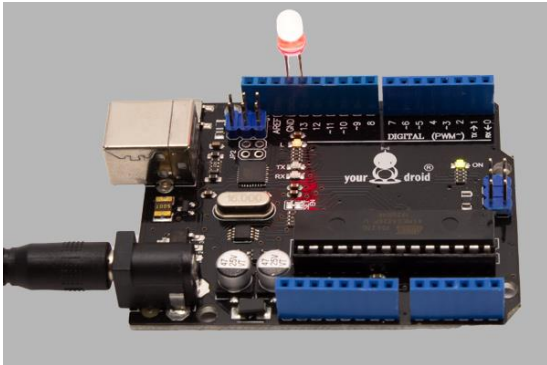
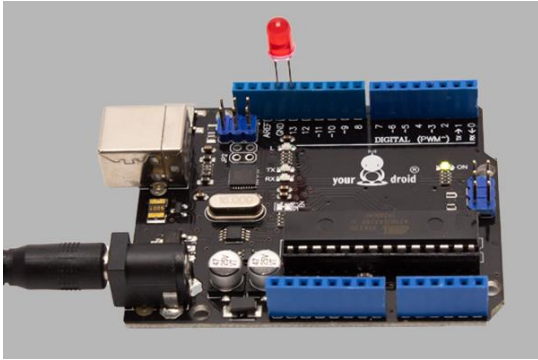
## Ergebnis

Wenn alles funktioniert hat, blinkt unsere LED im 1-Sekunden-Takt. Als nächstes können wir mal die Verzögerungen ändern und die LED schneller oder langsamer blinken lassen.



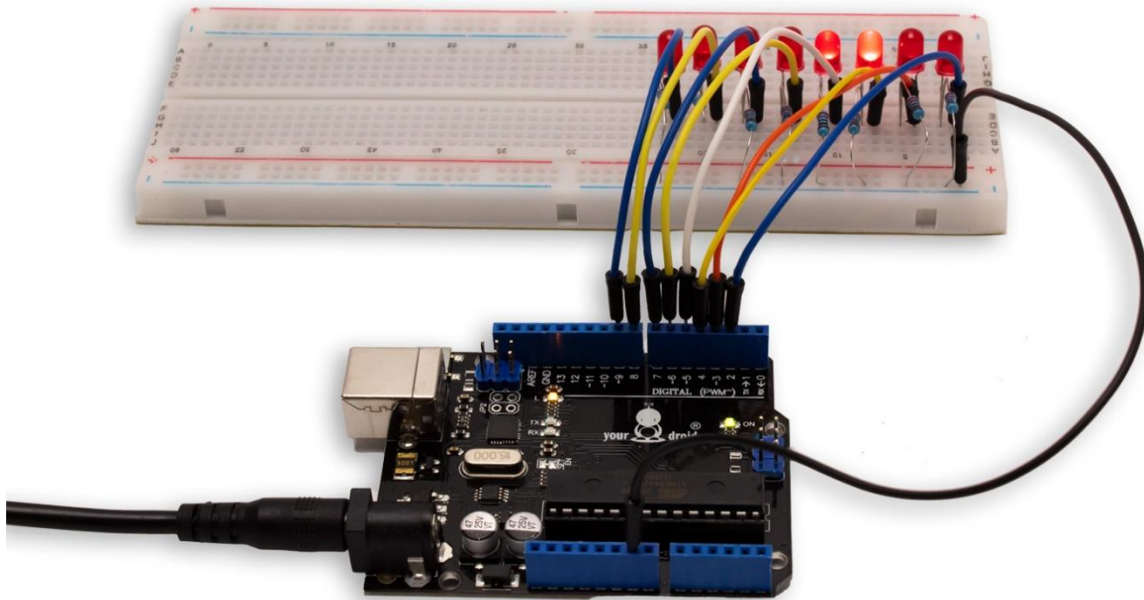
```

1 /*
2  Blink
3  Schaltet eine LED für 1 Sekunde an und für eine Sekunde aus.
4  */
5
6 // Die Setup-Funktion wird einmalig nach jedem Start oder Reset ausgeführt
7 void setup() {
8   // digitaler pin 13 wird als Output deklariert
9   pinMode(13, OUTPUT);
10 }
11
12 // Die Loop-funktion wird immer und immer wieder ausgeführt
13 void loop() {
14   digitalWrite(13, HIGH); // Schaltet die LED an (HIGH ist die Spannungsversorgung)
15   delay(1000); // 1000 ms (= 1 Sekunde) Verzögerung
16   digitalWrite(13, LOW); // Schaltet die LED aus indem die Spannungsversorgung auf LOW gesetzt wird
17   delay(1000); // 1 Sekunde Verzögerung
18 }
19
  
```



Viel Spaß beim Ausprobieren!

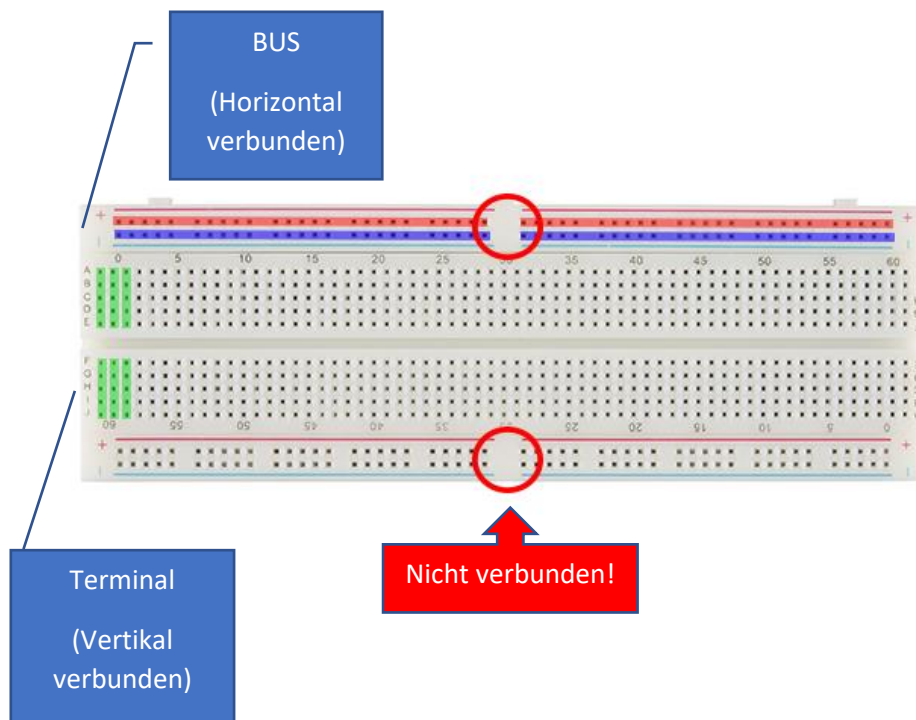
## 2. Lauflicht mit 8 LEDs



### Benötigte Komponenten

- 1x UNO R3 kompatibles Board
- 1x USB-Kabel
- 1x Breadboard
- 8x 5mm LED
- 8x Jumper Kabel
- 8x 220 Ohm Widerstand

## Hinweise zum Breadboard

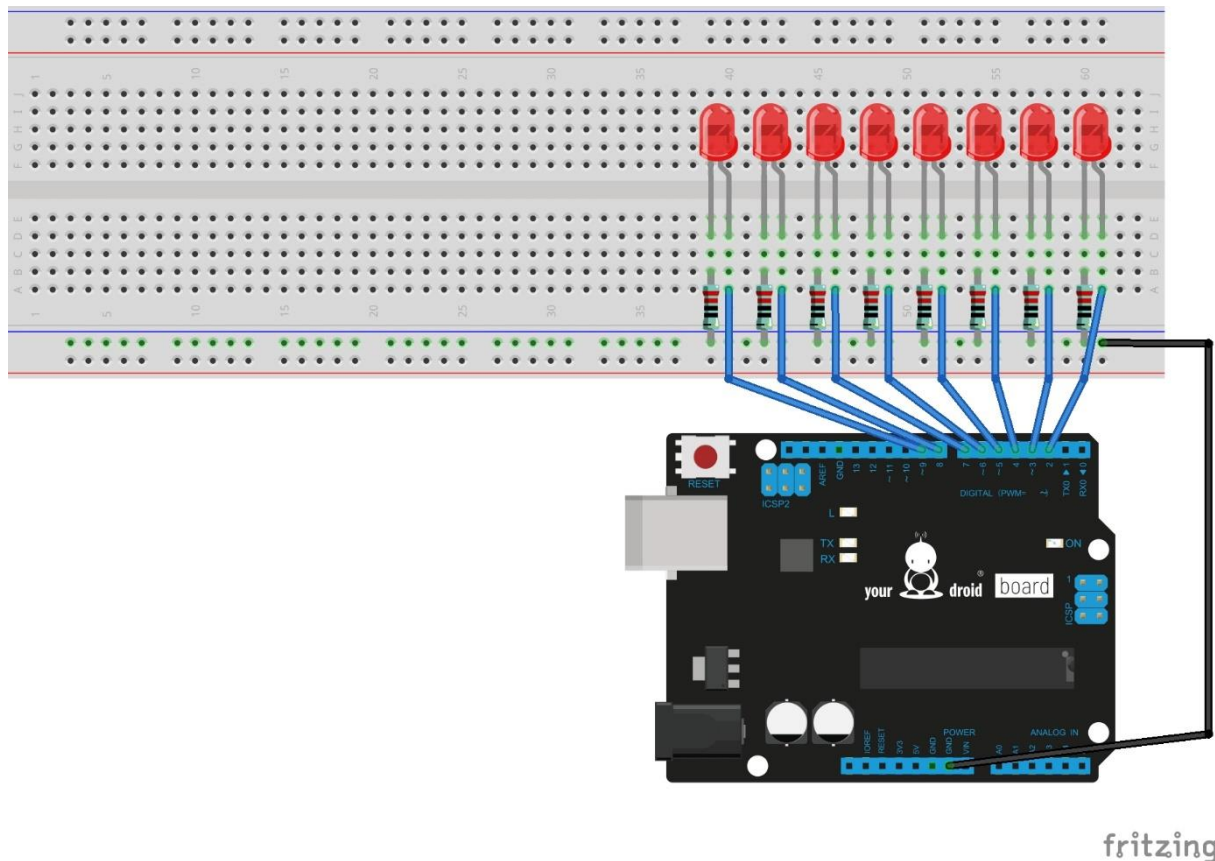


Die Kontakt-Terminals des Breadboards sind vertikal miteinander verbunden und werden zum Anschließen von Komponenten verwendet.

Die Verteiler-Terminals ( + / - ) sind waagerecht miteinander verbunden und bei den meisten Aufbauten für die Stromversorgung reserviert.

Eine beliebte Fehlerquelle bei Einsteigerprojekten sind Breadboards bei denen der Kontakt des Anschluss-Terminals in der Mitte unterbrochen ist, dieser ist meistens durch eine Lücke gekennzeichnet und sollte für den Anfang immer mit einem Kabel oder Draht überbrückt werden.

## Anschluss



## LEDs in elektronischen Schaltungen

Die Standard 5mm Leuchtdioden sind die am häufigsten verwendeten LEDs in elektronischen Schaltungen.

Man findet Sie in verschiedensten Farben auch in sämtlichen Hobbyprojekten und Tutorials wieder. Sie beginnen bei 8-12mA zu leuchten und erreichen bei ca. 20mA ihre volle Helligkeit. Diese LEDs müssen immer mit Vorwiderständen betrieben werden, in unserem Beispiel verwenden wir **220 Ohm Vorwiderstände**.

## Wieso werden Vorwiderstände benötigt?

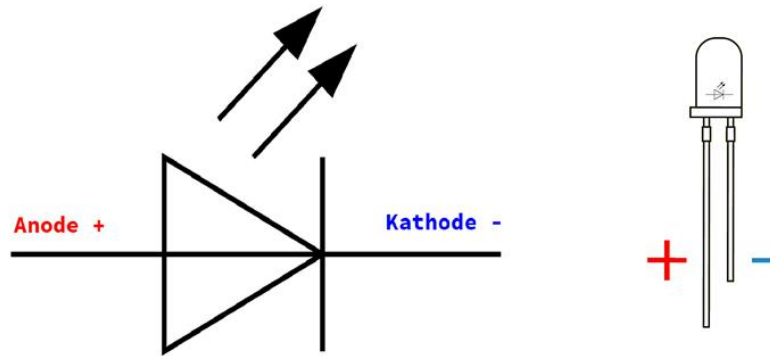
Da diese Leuchtdioden sehr empfindlich auf den Durchlassstrom reagieren, muss dieser mit Hilfe von Widerständen begrenzt werden - auch wenn die Betriebsspannung der Durchflussspannung entspricht. **Der Vorwiderstand dient zum Begrenzen der Spannung und des Stroms**, der durch die LED fließt.

Ohne den Vorwiderstand würde die LED zunächst warm werden und anschließend durchbrennen. Die LED muss sich nicht zwangsläufig sofort zerstören, aber da ein warmer Halbleiter besser leitet, ist es nur eine Frage der Zeit, bis der Stromanstieg zu viel für unsere kleine LED wird.

Wie man den Vorwiderstand berechnet, können Sie [auf der Webseite des Elektronik-Kompandiums im Detail nachlesen](#).



## LED- Polarität und Schaltzeichen



Bitte die Polung der LED beachten: Langes Beinchen Anode (+), kurzes Beinchen Kathode (-)

## Code

```
int timer = 100;          // Verzögerung

void setup() {
  // Schleife um pin 2 bis 9 als Output zu deklarieren:
  for (int LED_PIN = 2; LED_PIN < 10; LED_PIN++) {
    pinMode(LED_PIN, OUTPUT);
  }
}

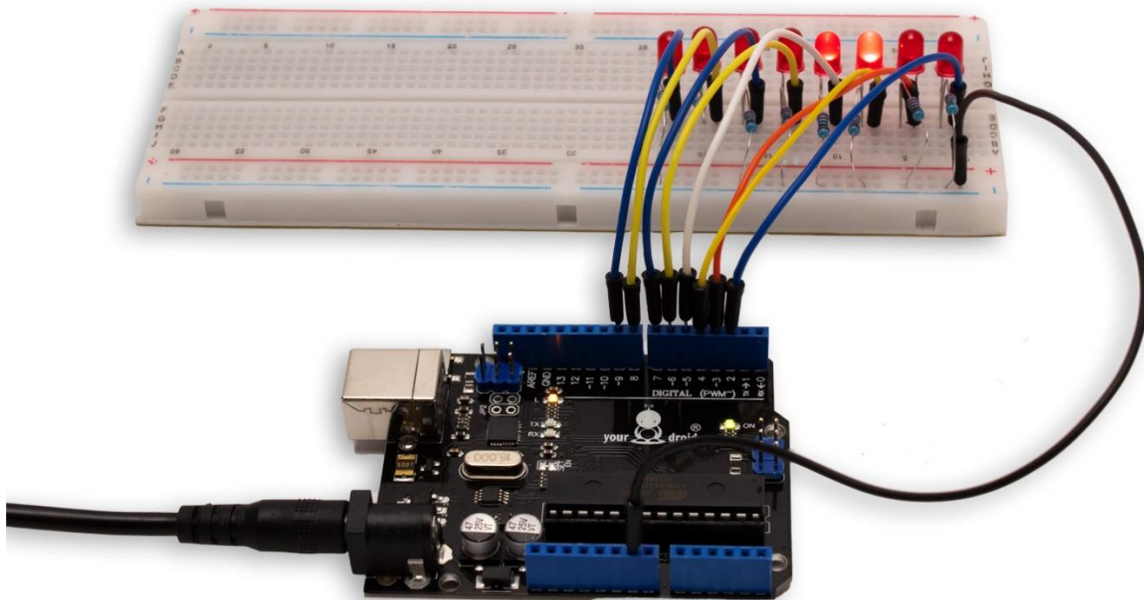
void loop() {
  // Schleife vom niedrigsten zum höchsten Pin:
  for (int LED_PIN = 2; LED_PIN < 10; LED_PIN++) {
    // LED einschalten:
    digitalWrite(LED_PIN, HIGH);
    delay(timer);
    // LED ausschalten:
    digitalWrite(LED_PIN, LOW);
  }

  // Schleife vom höchsten zum niedrigsten Pin:
  for (int LED_PIN = 8; LED_PIN >= 2; LED_PIN--) {
    // LED einschalten:
    digitalWrite(LED_PIN, HIGH);
    delay(timer);
    // LED ausschalten:
    digitalWrite(LED_PIN, LOW);
  }
}
```

Der Code basiert auf dem Arduino Beispiel für die for-Schleife. Wie das mit dem Hochzählen der Parameter funktioniert, ist [in der Dokumentation](#) auf der offiziellen Arduino Webseite erklärt.



## Ergebnis



Die LEDs sollten jetzt nacheinander aufleuchten. Dieser Aufbau eignet sich perfekt für Knightriderfans, nerdige Geschenke und Effekte im Modellbau!

## 3. Taster



### Benötigte Komponenten

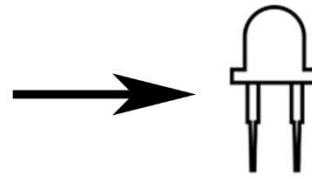
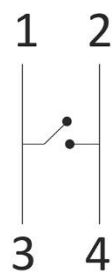
- Arduino UNO kompatibles Board + USB-Kabel
- Breadboard
- Taster
- 5mm LED
- 1k Widerstand (Band: braun-Schwarz-Rot-braun)
- 220 Ohm Widerstand (Band: Rot-Rot-Schwarz-Schwarz-braun)
- 4 Jumperkabel

### Was sind elektronische Taster?

Ein Taster, auch Knopf oder Button genannt, ist ein Bedienelement, das bei Betätigung auslöst (Ein) und beim Loslassen wieder in den Ursprungszustand zurückkehrt (Aus). **Ein Taster ist nur „Ein“, solange die Taste gedrückt wird.**

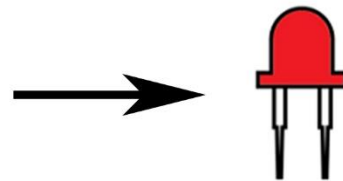
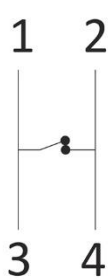
### Wie funktioniert ein Taster?

Die Pins sind paarweise verbunden 1-3 und 2-4. Bei Betätigung des Schalters wird der Kontakt in der Mitte geschlossen und der Strom kann fließen.



Taster offen

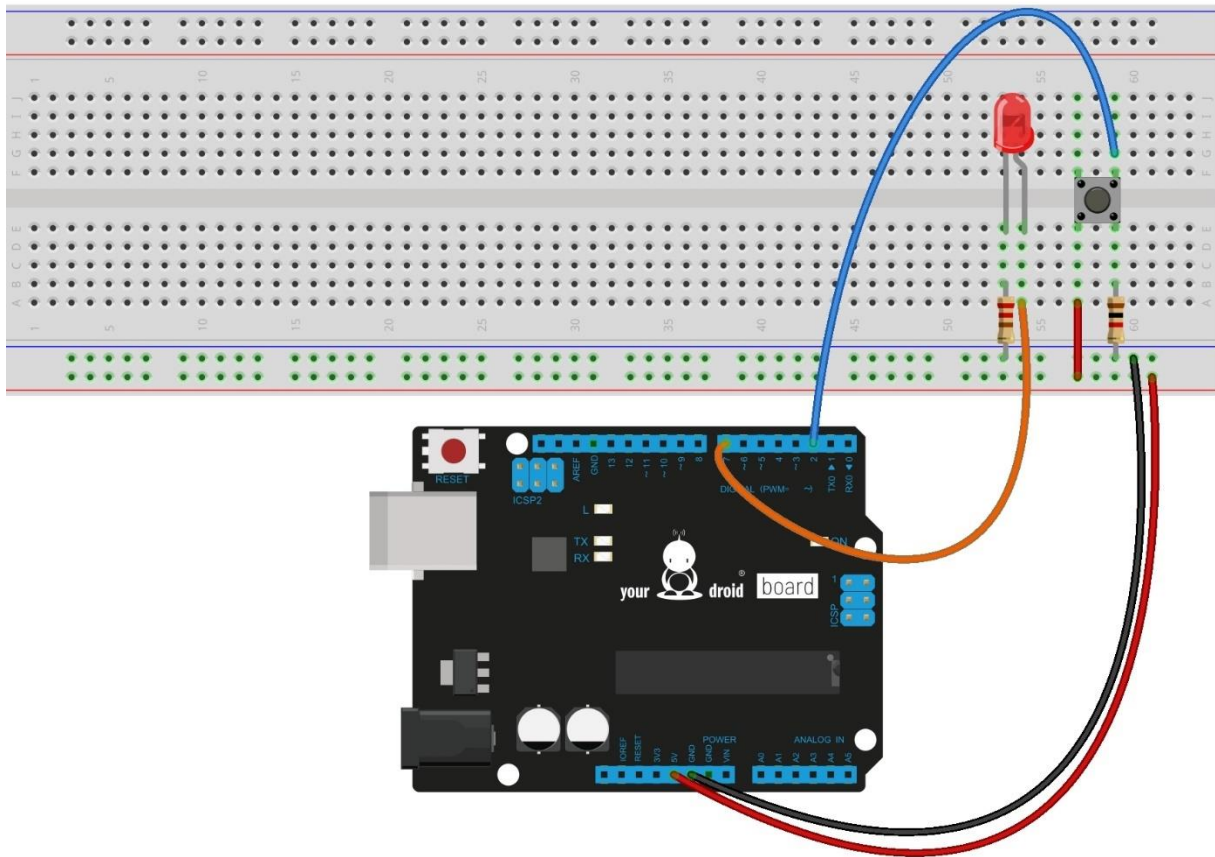
LED aus



Taster gedrückt

LED an

## Anschlussplan



fritzing

**Hinweis zur Schaltung:** Ohne den 10k „Pull-Down“-Widerstand kann es sein, dass die LED verrücktspielt. Das liegt daran, dass der Pin vom Taster in einem „float“-Zustand ist und eine Serie von Nullen und Einsen ausgibt. Durch den „Pull-Down“-Widerstand wird das Signal auf „low“ (Masse) gezogen, bis der Taster betätigt wird.

## Code für Arduino

```
const int tasterPin = 2;
const int ledPin = 7;

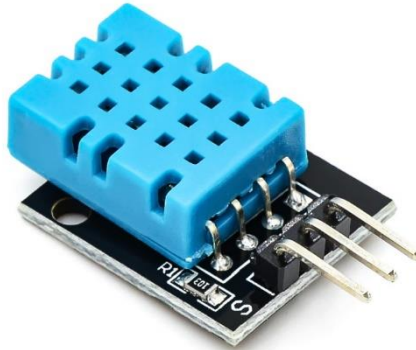
int buttonState = 0;          // Variable um den Taster-Status einzulesen

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(tasterPin, INPUT);
}

void loop() {
  buttonState = digitalRead(tasterPin); // Auslesen vom Taster-Status

  // Check ob der Taster gedrückt wird
  if (buttonState == HIGH) {
    // wenn ja, LED einschalten
    digitalWrite(ledPin, HIGH);
  } else {
    // ansonsten, LED ausschalten
    digitalWrite(ledPin, LOW);
  }
}
```

## 4. DHT11 Temperatursensor



### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- DHT11 Modul
- 3 Dupontkabel Buchse-Stecker

### Überblick DHT11 Temperatur- und Luftfeuchtigkeits-sensor

Dieses DHT11 Sensormodul eignet sich ideal zur Überwachung der Temperatur und relativen Luftfeuchtigkeit in Wohnräumen. Der Sensor misst Temperaturen von 0°C bis 50°C mit einer Genauigkeit von  $\pm 2^\circ\text{C}$  und die Luftfeuchtigkeit von 20% bis 95% mit einer Genauigkeit von  $\pm 5\%$ .

Das DHT11 Modul wird mit nur 3 Pins verkabelt und arbeitet mit 3V bis 5V. Es lässt sich besonders leicht an einen Arduino anschließen, um schnell und unkompliziert Messwerte ermitteln zu können. In der Bibliotheken-Verwaltung der Arduino-Software sind bereits standardmäßig mehrere Bibliotheken mit Beispielcode zur Installation verfügbar, so können Sie den DHT11 Sensor in kürzester Zeit in Betrieb nehmen.

### DHT11 vs DHT22

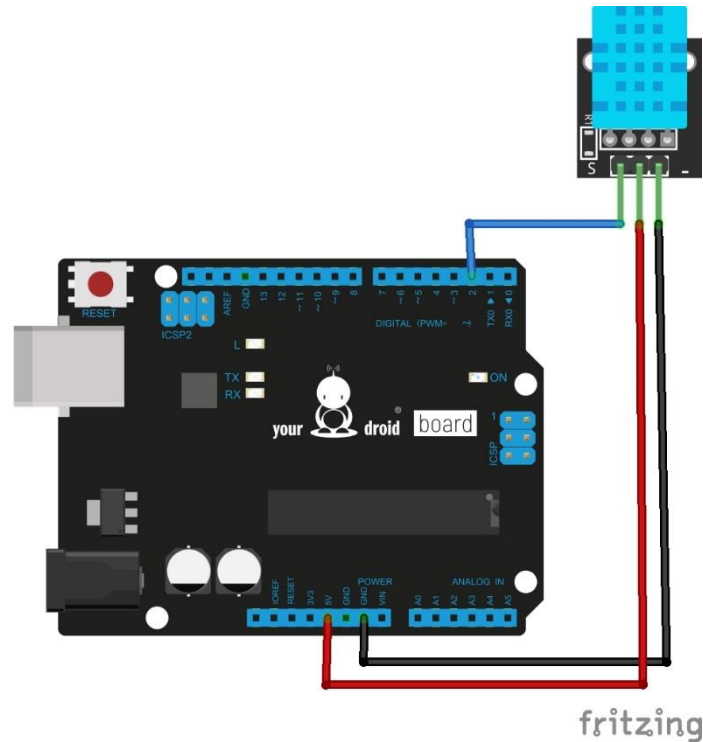
Als Alternative bietet sich der etwas teurere DHT22 Sensor an, welcher größere Messbereiche erkennen, genauer und zwei Mal pro Sekunde messen kann.

Falls Sie sich fragen, welcher Sensor der richtige für Ihr Projekt ist, haben wir hier eine kleine Übersicht zusammengestellt.

Vergleichswert	DHT11	DHT22
<b>Reichweite (Temperatur)</b>	0 - 50°C	-40 – 80°C
<b>Genauigkeit (Temperatur)</b>	$\pm 2^\circ\text{C}$	$\pm 0.5^\circ\text{C}$
<b>Reichweite (Luftfeuchtigkeit)</b>	20-95%	0-100%
<b>Genauigkeit (Luftfeuchtigkeit)</b>	$\pm 5\%$	$\pm 2\%$
<b>Messintervall</b>	1000 Millisekunden	500 Millisekunden

Als Fazit kann man sagen, dass der DHT22 auf ganzer Linie punktet, außer dem Preis. Trotzdem ist der DHT11 der beliebteste Temperatursensor für Microcontroller-Projekte, da er sehr günstig, genau und stromsparend ist.

## Anschlussplan

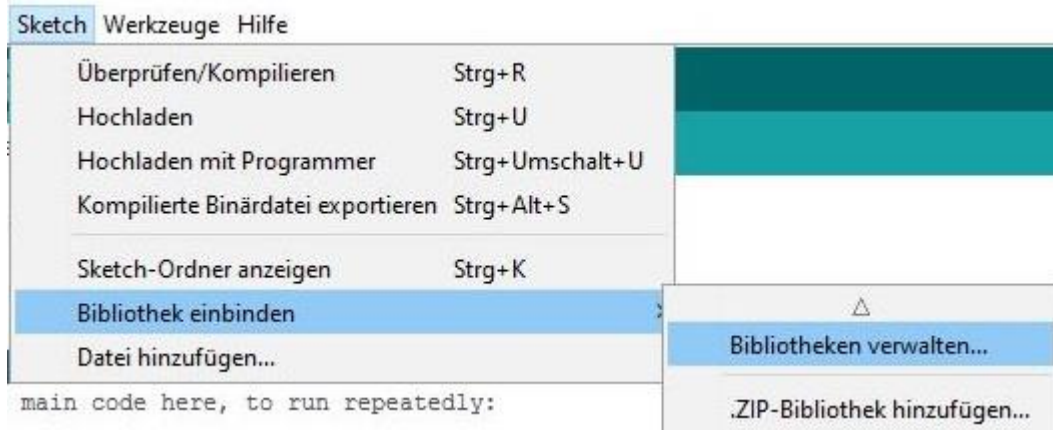


DHT11	Arduino
<b>S (Datenleitung)</b>	Pin 2
<b>VCC (Stromversorgung 3-5V)</b>	5V
<b>- (GND)</b>	GND

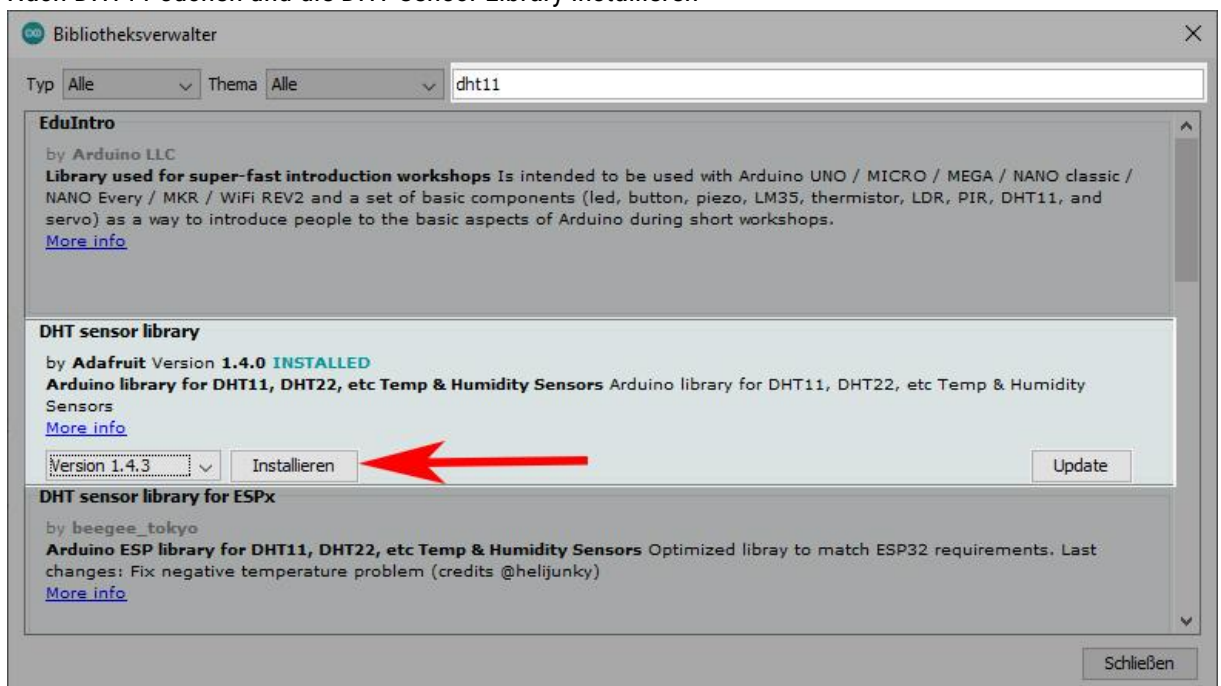
## DHT Sensor Library installieren

Um die ermittelten Messwerte ganz einfach in C° auszugeben, verwenden wir die DHT Bibliothek. Die Installation erfolgt unkompliziert über die Arduino IDE. Die Bibliothek kann auch für andere DHT-Sensoren verwendet werden.

## 1. Bibliotheken verwalten aufrufen



## 2. Nach DHT11 suchen und die DHT Sensor Library installieren



## Code für Arduino

Den Code finden Sie ebenfalls unter „Datei – Beispiele – DHT“. Sie müssen nur den DHT11 statt dem DHT22 auskommentieren.

```
// Example testing sketch for various DHT humidity/temperature sensors
// Written by ladyada, public domain

// REQUIRES the following Arduino libraries:
// - DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
// - Adafruit Unified Sensor Lib:
https://github.com/adafruit/Adafruit\_Sensor

#include "DHT.h"

#define DHTPIN 2 // Digital pin connected to the DHT sensor
// Feather HUZZAH ESP8266 note: use pins 3, 4, 5, 12, 13 or 14 --
// Pin 15 can work but DHT must be disconnected during program upload.
```



```
// Uncomment whatever type you're using!
#define DHTTYPE DHT11    // DHT 11
// #define DHTTYPE DHT22   // DHT 22   (AM2302), AM2321
// #define DHTTYPE DHT21   // DHT 21   (AM2301)

// Initialize DHT sensor.
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println(F("DHTxx test!"));

  dht.begin();
}

void loop() {
  // Wait a few seconds between measurements.
  delay(2000);


  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
  // sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Compute heat index in Fahrenheit (the default)
  float hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

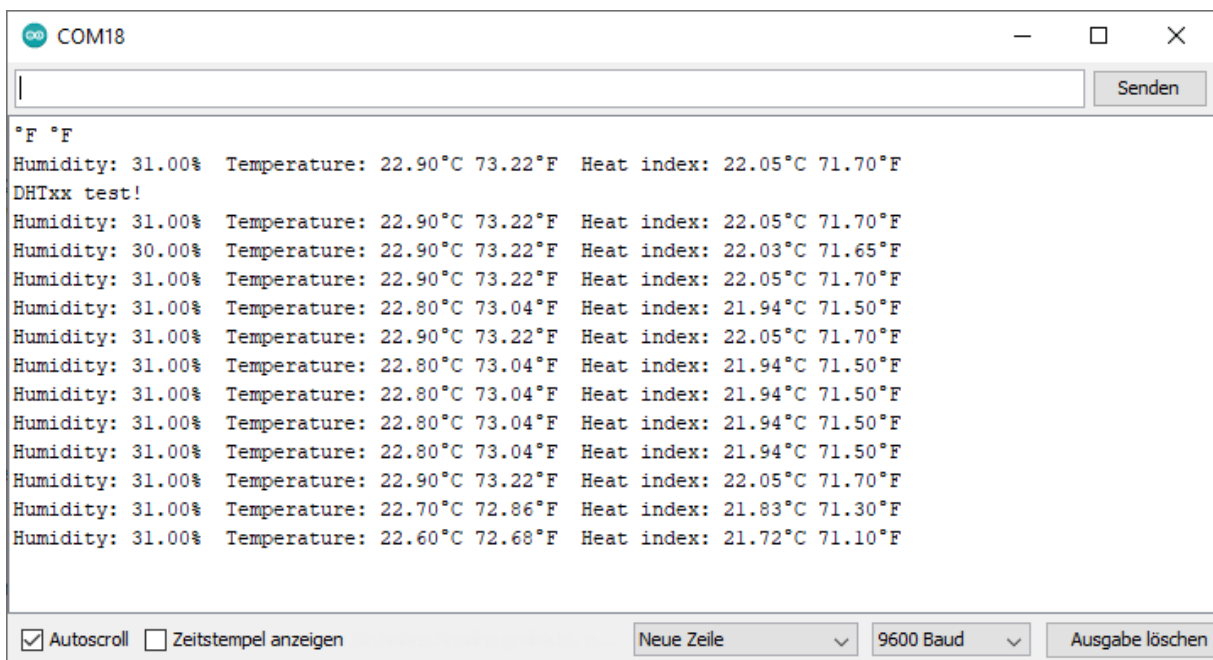
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%   Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F   Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
```

## Auslesen mit dem seriellen Monitor

Zum Aufrufen des seriellen Monitors klicken wir auf dieses Symbol  in der rechten, oberen Ecke der Arduino IDE, oder navigieren zu Werkzeuge -> Serieller Monitor



Hier bekommen wir dann die vom Sensor erfassten Messwerte angezeigt:



Wichtig ist, dass das angeschlossene Board korrekt eingestellt ist (wie beim Hochladen von Programmen) und die Baudrate auf 9600 eingestellt ist.

## 5.RGB LED-Modul



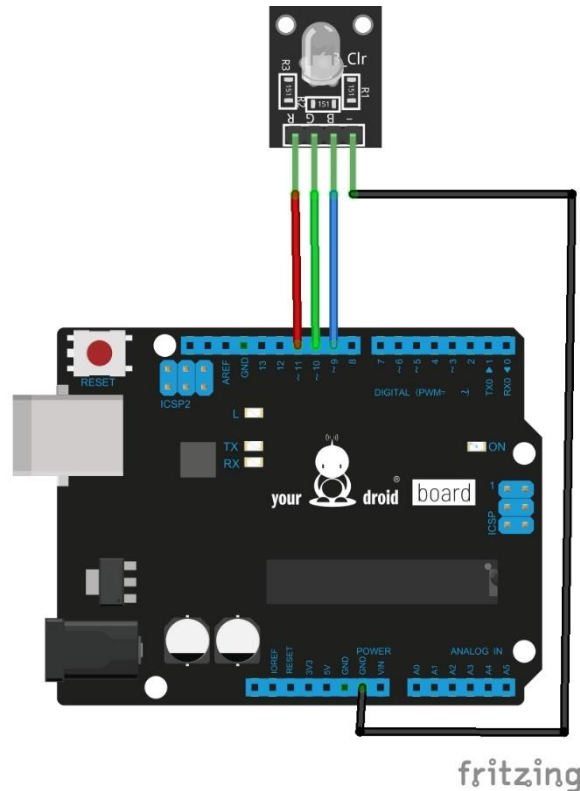
### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- RGB LED Modul
- 4 Dupontkabel Buchse-Stecker

### Funktionsweise

Dieses Modul besteht aus einer 5mm RGB LED und drei 150 Ohm Vorwiderständen. Die einzelnen Farbkanäle werden über drei Pins per PWM angesteuert. Weitere Farben werden durch Mischen der primären Farben erreicht.

## Anschlussplan



RGB Modul	Arduino
<b>R</b>	Pin 11
<b>B</b>	Pin 10
<b>G</b>	Pin 9
-	GND

## Code für Arduino

Mit diesem Code leuchtet die RGB LED pulsierend in Rot, Grün und Blau auf. Die PWM-Werte des roten, grünen und blauen Pins werden schrittweise erhöht und verringert, wodurch die RGB LED die verschiedenen Farben durchläuft.

```
#define RGB_ROT 11 // Pin fuer die rote LED
#define RGB_GRUEN 10 // Pin fuer die gruen LED
#define RGB_BLAU 9 // Pin fuer die blaue LED
void setup()
{
  pinMode(RGB_ROT, OUTPUT);
  pinMode(RGB_GRUEN, OUTPUT);
  pinMode(RGB_BLAU, OUTPUT);
}
void loop()
{
  int i;
  for (i=255; i>0 ;i--)
  {
    analogWrite(RGB_ROT, i);
    analogWrite(RGB_GRUEN, 0);
    analogWrite(RGB_BLAU, 0);
    delay(4);
  }
}
```

```
}  
delay(500); // RGB LED rot aufleuchten lassen  
for (i=255; i>0; i--)  
{  
  analogWrite(RGB_ROT, 0);  
  analogWrite(RGB_GRUEN, i);  
  analogWrite(RGB_BLAU, 0);  
  delay(4);  
}  
delay(500); //RGB LED gruen aufleuchten lassen  
for (i=255; i>0; i--)  
{  
  analogWrite(RGB_ROT, 0);  
  analogWrite(RGB_GRUEN, 0);  
  analogWrite(RGB_BLAU, i);  
  delay(4);  
}  
delay(500); // RGB LED blau aufleuchten lassen  
}
```

## 6. Buzzer



### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- Buzzer
- Breadboard
- 2 Jumperkabel
- Optional: 150 Ohm Widerstand

### Grundlagen Buzzer

Ein Buzzer oder auf Deutsch "Summer", ist ein elektrisches Bauteil für akustische Signale. Mit einem Buzzer können Sie einen Summ- oder Piepton erzeugen. Sie werden häufig in Computern, Druckern, Kopierern, Alarmanlagen, Spielzeugen und anderen elektronischen Geräten verwendet.

Es gibt zwei Arten von Buzzer, aktive und passive Buzzer. Der Unterschied liegt nicht in der Stromversorgung, sondern in der Oszillation des Tons.

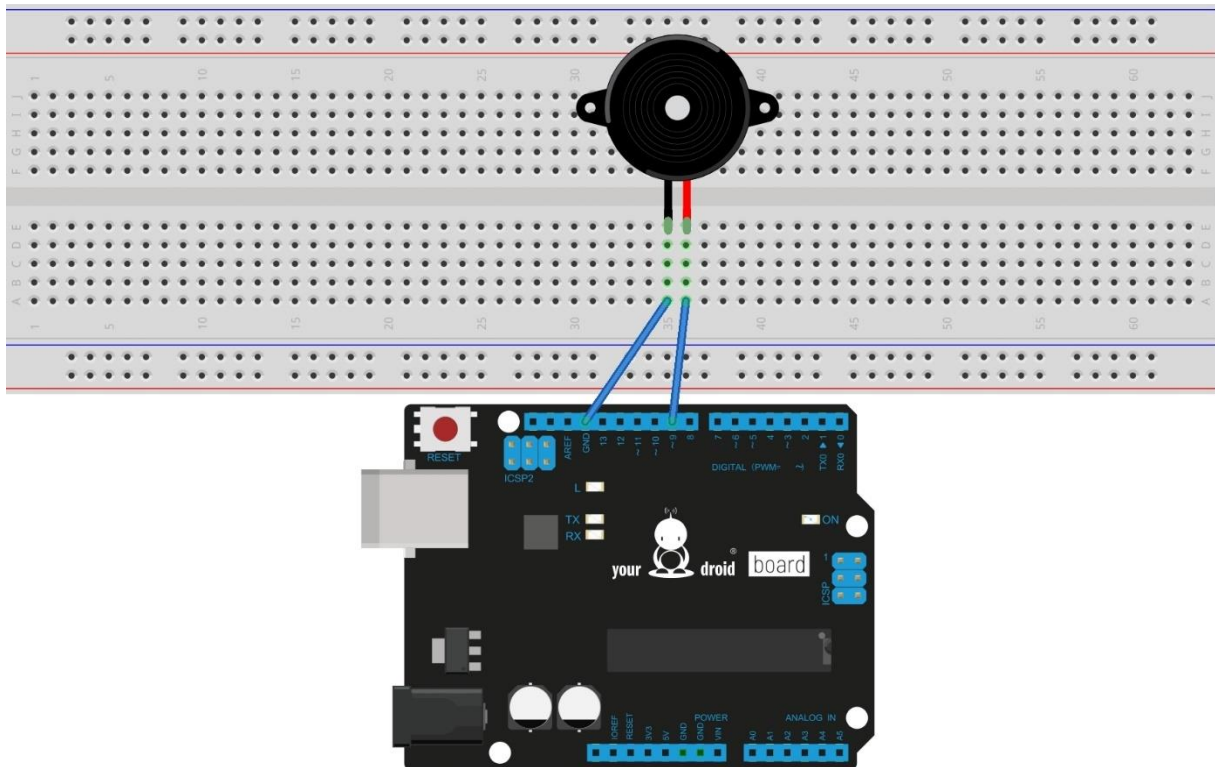


*Links aktiver Buzzer; Rechts passiver Buzzer*

Der aktive Buzzer besitzt einen Oszillator, der auf einer festgelegten Frequenz schwingt und somit einen **festgelegten Ton** ausgibt.

Der passive Buzzer besitzt keinen Oszillator und benötigt eine 2-5 KHz Sinuswelle, um durch die Schwingung einen **variablen Ton** zu erzeugen.

## Anschlussplan



fritzing

Buzzer	Arduino
+	Pin 9
-	GND

Optional kann noch ein 150 Ohm Widerstand zwischen Pin 9 und + gesetzt werden.

## Code für Arduino

### Aktiver Buzzer

```
int buzzer = 9; // Buzzer Pin
void setup()
{
  pinMode(buzzer, OUTPUT);
}
void loop()
{
  digitalWrite(buzzer, HIGH); // Ton an
  delay(1000);
  digitalWrite(buzzer, LOW); // Ton aus
  delay(1000);
}
```

### Passiver Buzzer

```
const int buzzer = 9;
```

```
void setup() {  
    pinMode(buzzer, OUTPUT);  
}  
  
void loop() {  
    tone(buzzer, 1000); // Sende 1KHz Tonsignal  
    delay(1000);        // 1 sec Pause  
    noTone(buzzer);      // Ton stoppen  
    delay(1000);        // 1 sec Pause  
}
```

In den beiden Codes sieht man nochmal deutlich die unterschiedliche Funktionsweise.

Beim aktiven Buzzer wird einfach eine Spannungsquelle angelegt (Pin auf High), um einen Ton zu erzeugen.

Beim passiven Buzzer kann mit der tone-Funktion die gewünschte Frequenz in KHz ausgegeben werden.

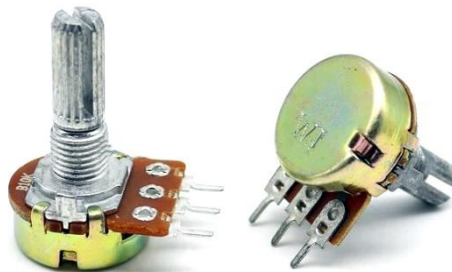
### Bonus – Super Mario Theme Song mit passivem Buzzer

*Hinweis: Hier wird der Buzzer nicht an Pin9, sondern an Pin 3 angeschlossen.*

Quelle: <https://create.arduino.cc/projecthub/jrance/super-mario-theme-song-w-piezo-buzzer-and-arduino-1cc2e4>



## 7.LED mit Potentiometer dimmen



### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- Potentiometer
- Breadboard
- 6 Jumperkabel
- 220 Ohm Widerstand
- 5mm LED

### Übersicht

Ein Potentiometer (kurz Poti) ist ein Widerstand, der mechanisch verändert werden kann. Das Drehpotentiometer besteht aus einer Bahn mit Widerstandsmaterial und einem beweglichen Abgreifkontakt (auch Schleifer oder Wischer genannt). Potentiometer werden oft für manuelle Regelungen eingesetzt, die stufenlos erfolgen sollen. Einsatzbeispiele wären Lautstärkeregler, Drehzahlregler, Dimmer.

In unserem Beispiel verwenden wir ein Drehpotentiometer (Drehregler) um die Helligkeit einer LED zu dimmen. Das Potentiometer besitzt 3 Pins, die äußeren sind für Anfang und Ende der Widerstandsbahn und der mittlere Pin ist der Mittelabgriff, von dem unser Arduino Board eine variable Spannung erhält.

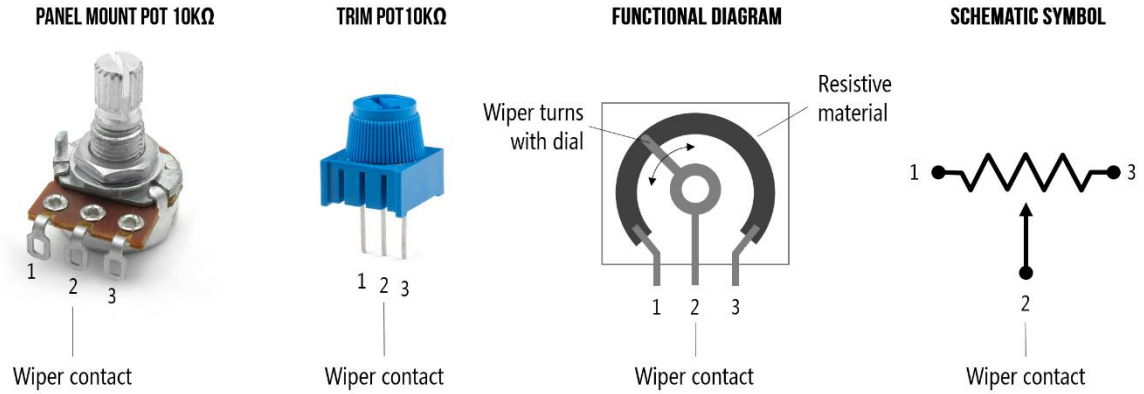
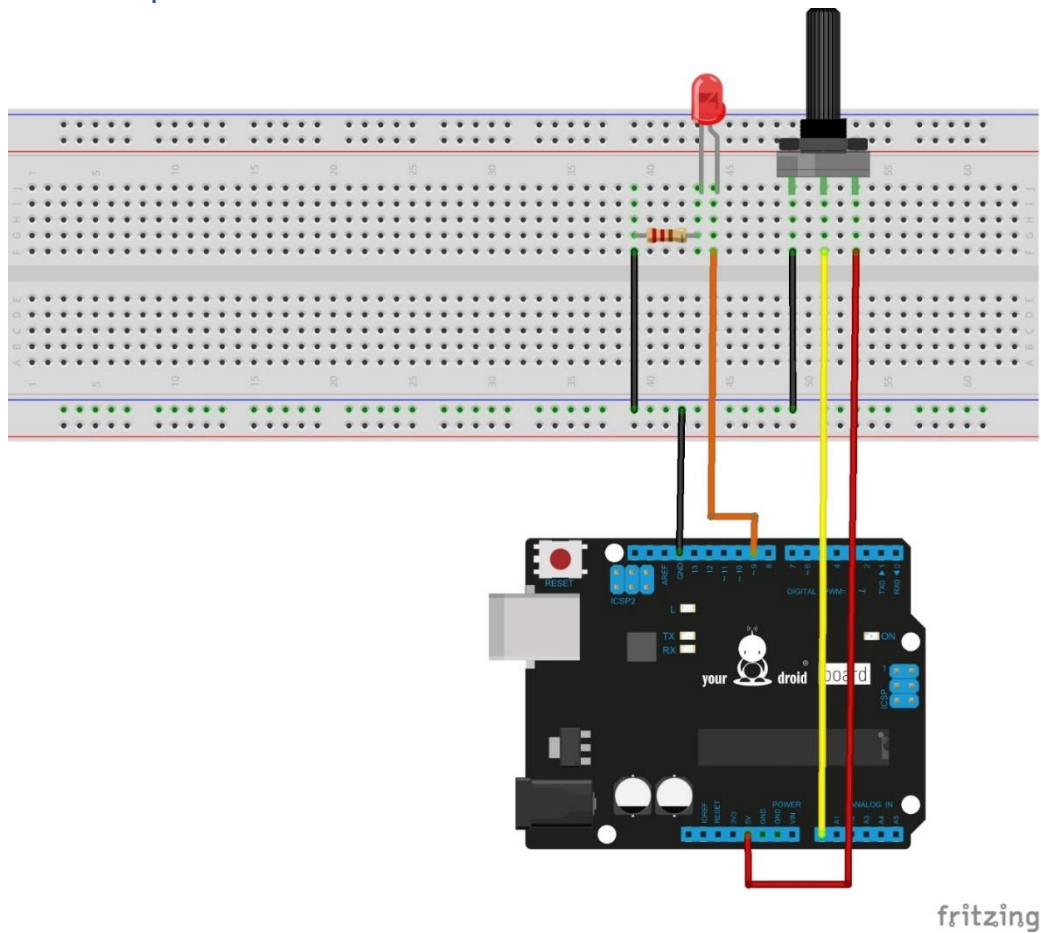


Abbildung 1 Beispiele und Funktionsweise verschiedener Potentiometer. Bildquelle <https://makeabilitylab.github.io/physcomp/arduino/potentiometers.html>

## Anschlussplan



## Code für Arduino

Mit diesem Code können Sie die Helligkeit der LED stufenlos über das Potentiometer regulieren.

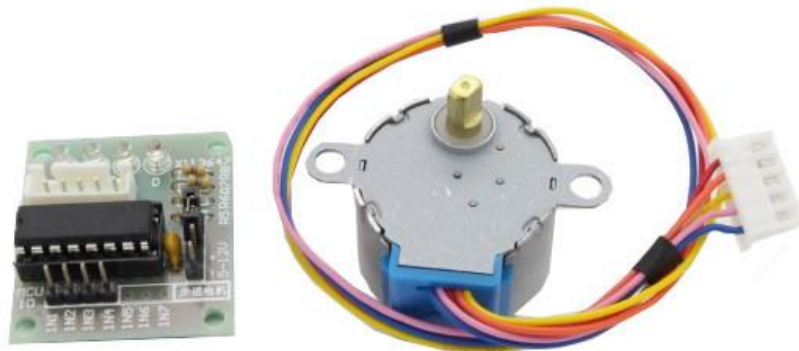
```
int eingang= A0;
int LED = 9;
int sensorwert = 0;

void setup() //Hier beginnt das Setup.
{
  pinMode (LED, OUTPUT);
}

void loop()
{
  sensorwert =analogRead(eingang);
  analogWrite (LED, map(sensorwert,0,1023,0,255));
}
```

Da das Potentiometer mit Werten zwischen 0 (0V) und 1023 (5V) arbeitet, benutzen wir die map-Funktion, um die LED mit einem PWM-Wert zwischen 0 und 255 anzusteuern.

## 8. Schrittmotor



### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- 28BYJ48 Schrittmotor
- ULN2003 Treiberplatine

### Was ist ein Schrittmotor?

Ein Schrittmotor ist ein elektromechanischer Motor, der elektrische Impulse in genaue Positionierungen umsetzt. Eine volle Rotation wird in Schritte unterteilt, bei unserem **28BYJ-48 Schrittmotor** sind **2038 Schritte** eine volle Umdrehung. Die Welle des Schrittmotors bewegt sich schrittweise durch ein rotierendes, elektromagnetisches Feld, welches durch Spulen erzeugt wird. Dadurch kann man ohne Feedback vom Schrittmotor jederzeit die genaue Position bestimmen.

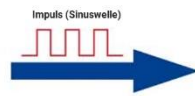
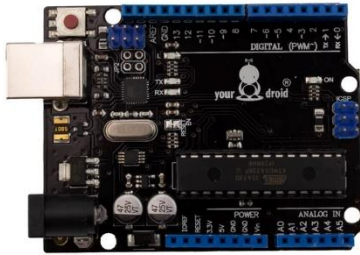
### Anwendungsbereiche des Schrittmotors

Schrittmotoren besitzen ein hohes Haltemoment bei niedrigen Geschwindigkeiten, was sie ideal für Anwendungen macht, bei denen es auf schnelle und präzise Bewegungen ankommt. Schrittmotoren werden in der Industrie vor Allem in automatisierten Prozessen, Robotik und Feinmechanik verwendet. Im privaten Sektor befinden sie sich in alltäglichen Elektronikgeräten wie Druckern, DVD- oder Blu-ray-Laufwerken.

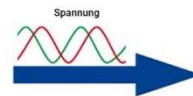
### Wie wird ein Schrittmotor angesteuert?

Die Steuerung von Schrittmotoren erfolgt über elektrische Impulse mit einer bestimmten Frequenz, Sequenz und Länge. So wird bestimmt wie schnell, wie viele Schritte und in welche Richtung der Schrittmotor sich drehen soll. Der Treiber übersetzt die Impulse vom Controller und versorgt den Schrittmotor mit Spannung.

## Controller



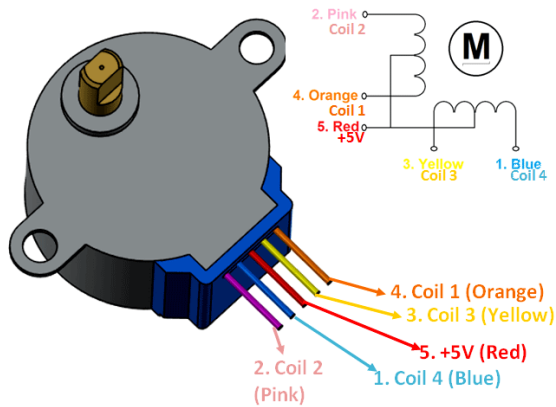
## Treiber



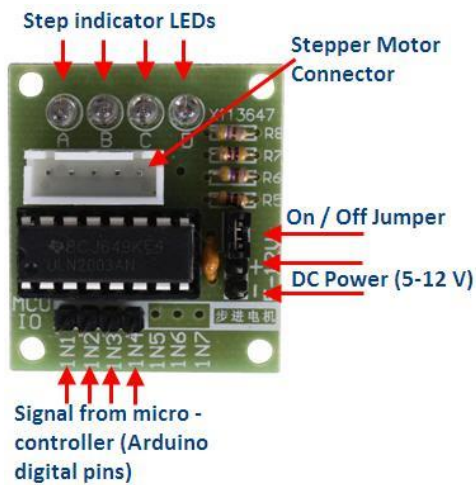
## Schrittmotor



## Anschlüsse

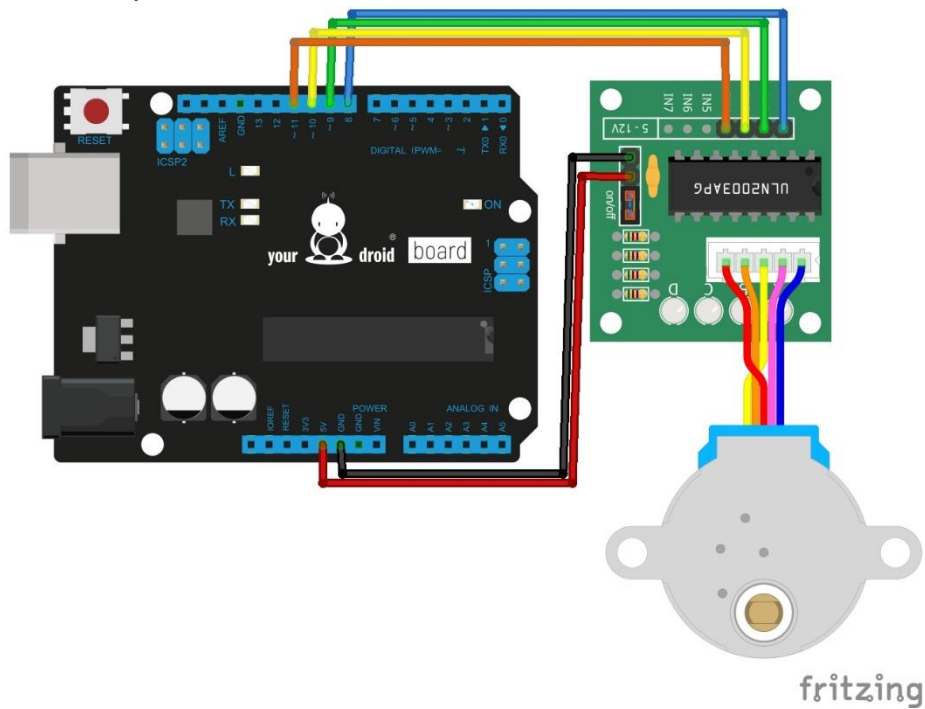


## 1 Anschlüsse vom Schrittmotor



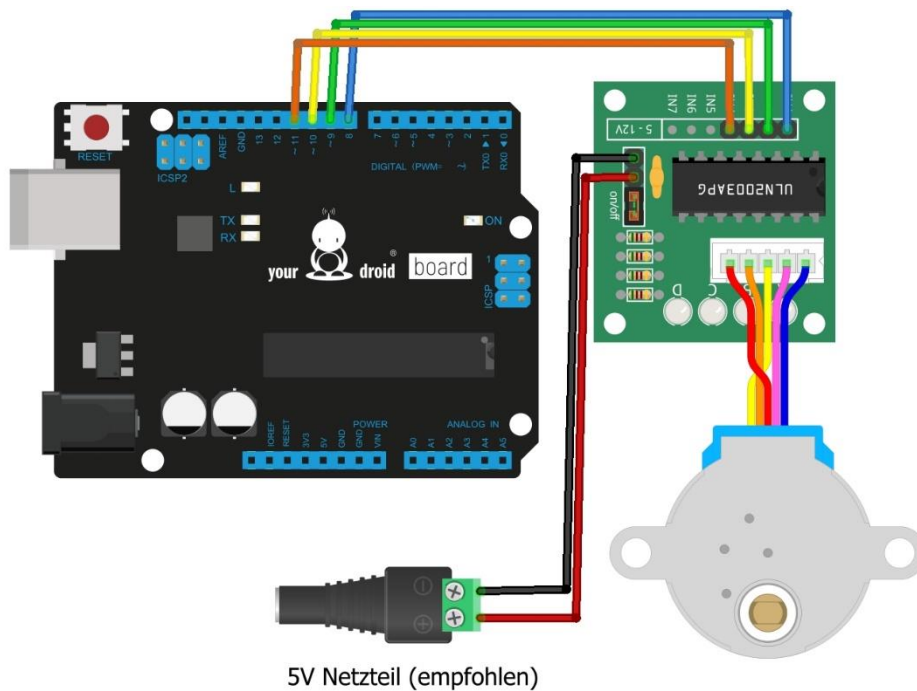
## 2 Anschlüsse der Treiberplatine ULN2003

## Anschlussplan



ULN2003 Platine	Arduino
IN1	Pin 8
IN2	Pin 9
IN3	Pin 10
IN4	Pin 11
+	5V
-	GND

**Hinweis:** Man kann die Stromversorgung für den Schrittmotor vom Arduino nehmen, verliert aber Leistung und Genauigkeit. Zum Ausprobieren in Ordnung, aber für richtige Projekte empfehlen wir ein [5V Netzteil](#) + [Adapter](#).

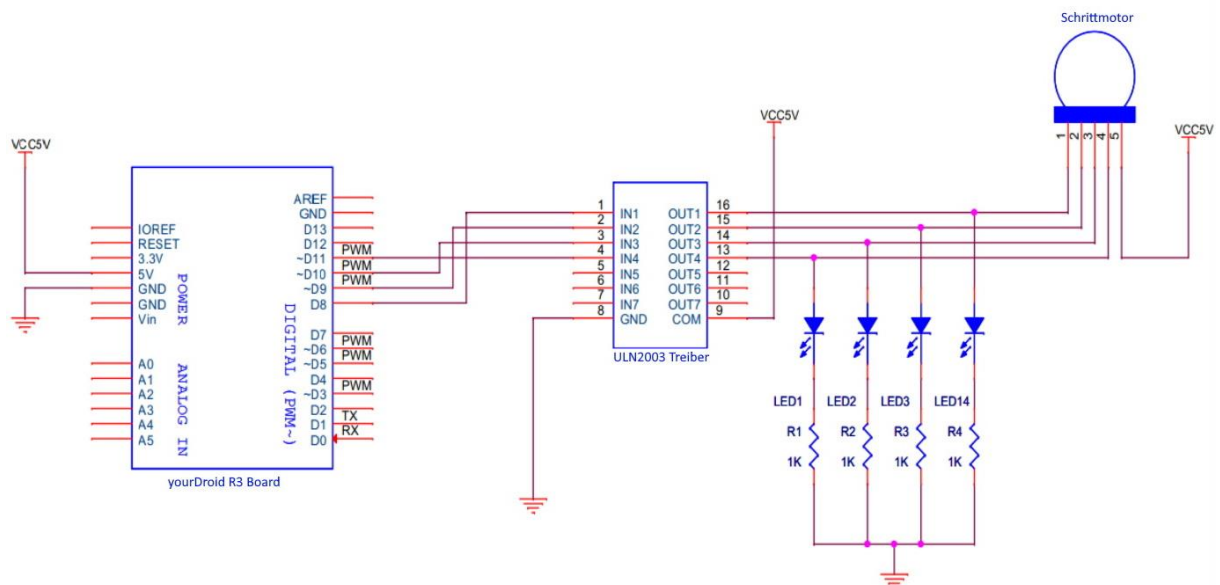


5V Netzteil (empfohlen)

fritzing

Wenn eine externe Stromversorgung (5-12V) benutzt wird, muss der Power-Jumper gesetzt werden!

## Schaltplan



## Code für Arduino

### Code mit Stepper.h Bibliothek

```
#include <Stepper.h>

// Anzahl der Schritte pro Umdrehung
#define STEPS 2038

// Motorgeschwindigkeit in RPM
#define MOTOR_RPM 6

Stepper stepper(STEPS, 8, 10, 9, 11); // Motor Pins

void setup()
{
}

void loop()
{
    stepper.setSpeed(MOTOR_RPM);
    stepper.step(STEPS);
    delay(1000);

    stepper.setSpeed(MOTOR_RPM);
    stepper.step(-STEPS);
}
```

### Code ohne Stepper.h Bibliothek

```
// Quelle:
https://github.com/NikodemBartnik/ArduinoTutorials/tree/master/28BYJ-48

#define STEPPER_PIN_1 8
#define STEPPER_PIN_2 9
#define STEPPER_PIN_3 10
#define STEPPER_PIN_4 11
int step_number = 0;
void setup() {
    pinMode(STEPPER_PIN_1, OUTPUT);
    pinMode(STEPPER_PIN_2, OUTPUT);
    pinMode(STEPPER_PIN_3, OUTPUT);
    pinMode(STEPPER_PIN_4, OUTPUT);
}

void loop() {

    OneStep(false);
    delay(2);

}

void OneStep(bool dir) {
```



```
    if(dir){
switch(step_number){
    case 0:
        digitalWrite(STEPPER_PIN_1, HIGH);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
    case 1:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, HIGH);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
    case 2:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, HIGH);
        digitalWrite(STEPPER_PIN_4, LOW);
        break;
    case 3:
        digitalWrite(STEPPER_PIN_1, LOW);
        digitalWrite(STEPPER_PIN_2, LOW);
        digitalWrite(STEPPER_PIN_3, LOW);
        digitalWrite(STEPPER_PIN_4, HIGH);
        break;
}
    }else{
        switch(step_number){
            case 0:
                digitalWrite(STEPPER_PIN_1, LOW);
                digitalWrite(STEPPER_PIN_2, LOW);
                digitalWrite(STEPPER_PIN_3, LOW);
                digitalWrite(STEPPER_PIN_4, HIGH);
                break;
            case 1:
                digitalWrite(STEPPER_PIN_1, LOW);
                digitalWrite(STEPPER_PIN_2, LOW);
                digitalWrite(STEPPER_PIN_3, HIGH);
                digitalWrite(STEPPER_PIN_4, LOW);
                break;
            case 2:
                digitalWrite(STEPPER_PIN_1, LOW);
                digitalWrite(STEPPER_PIN_2, HIGH);
                digitalWrite(STEPPER_PIN_3, LOW);
                digitalWrite(STEPPER_PIN_4, LOW);
                break;
            case 3:
                digitalWrite(STEPPER_PIN_1, HIGH);
                digitalWrite(STEPPER_PIN_2, LOW);
                digitalWrite(STEPPER_PIN_3, LOW);
                digitalWrite(STEPPER_PIN_4, LOW);
                break;
        }
    }
    step_number++;
    if(step_number > 3){
        step_number = 0;
    }
}
```

```
}  
}
```

## 9. DS1302 Echtzeituhrmodul (RTC)



### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- Echtzeituhr-Modul (RTC)
- Jumperkabel Buchse-Stecker

### Einleitung

Das DS1302 RTC-Modul ist ein günstiges Echtzeituhrmodul für verschiedenste DIY-Elektronik Projekte. Es zählt Sekunden, Minuten, Stunden, Tage, Wochentage, Monat und Jahr und Schaltjahre. Es ist sehr einfach in der Handhabung und kann durch Arduino Bibliotheken im Handumdrehen in vorhandene Projekte implementiert werden.

Es eignet sich ideal für Microcontroller Projekte mit Arduino und Raspberry Pi. Solche RTC-Module findet man in allen Projekten, die einen synchronen Zeitablauf benötigen. Sie werden besonders oft in digitalen Uhren, Kalendern oder Wetterstationen benutzt.

### Wozu wird das DS1302 RTC-Modul benötigt?

Das Modul wird benötigt, um unserem Arduino ein genaueres Zeitgefühl zu ermöglichen. Der Arduino nimmt Zeit nur mit der [Millis\(\) Funktion](#) wahr, es zählt also die Millisekunden seit dem letzten Neustart. Dies sorgt für zweierlei Probleme:

- Nach etwa 50 Tagen läuft die Zahl über und startet wieder bei Null.
- Durch Unregelmäßigkeiten im Frequenztakt vom Quarz entstehen schnell Abweichungen von mehreren Sekunden.

Diese Nachteile von der Zeiterfassung mit dem Arduino können mit dem DS1302 RTC-Modul umgangen werden.

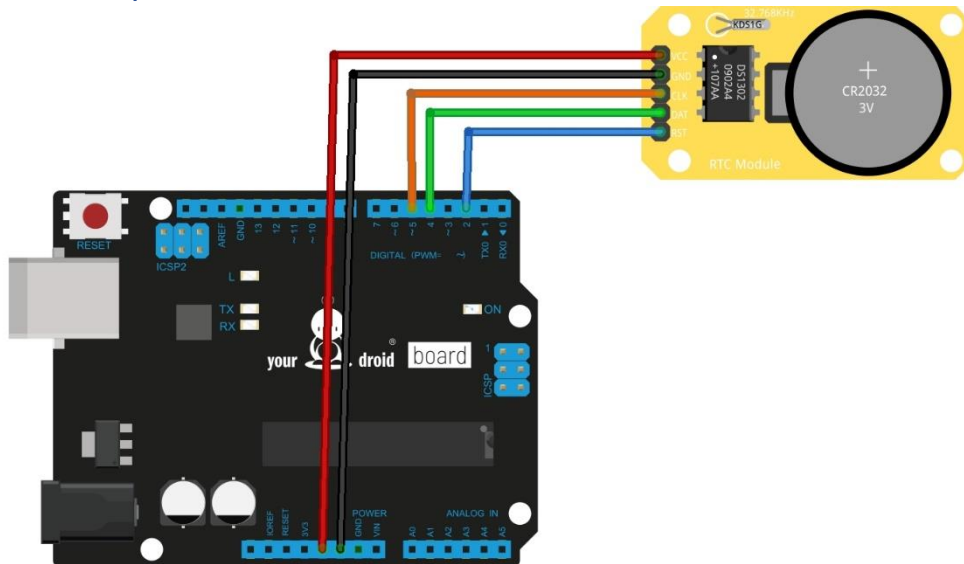
### Wozu dient die Batteriehalterung vom DS1302 RTC-Modul?

Die **CR2032 Knopfzelle** ist **optional** und dient dazu, dass der DS1302 RTC-Chip auch ohne Stromversorgung weiterläuft und die Uhrzeit beibehält.

### Nachteile gegenüber dem teureren DS3231 RTC-Modul

- Weniger präzise
- Ansteuerung über 3-Wire SPI statt I2C (benötigt mehr Pins)
- Kein Ladeschaltkreis für die Batterie

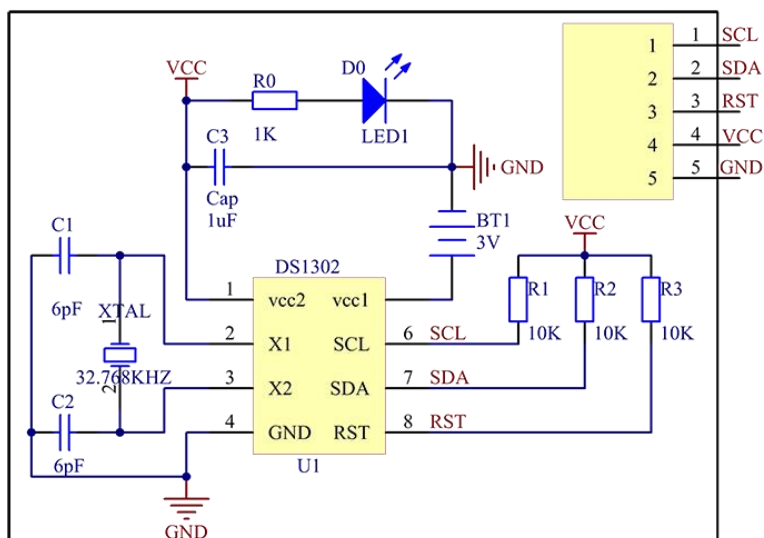
### Anschlussplan



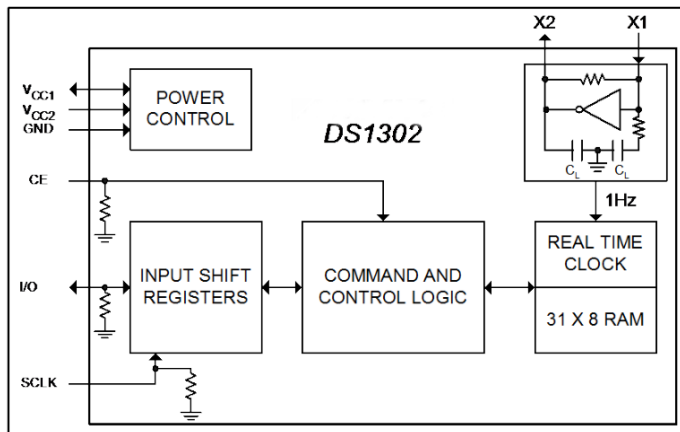
fritzing

RTC-Modul	Arduino
<b>VCC (Stromversorgung 2-5V)</b>	3.3V
<b>GND (Masse Pin)</b>	GND
<b>CLK (Clock Pin)</b>	D5
<b>DAT (Data Pin)</b>	D4
<b>RST (Reset Pin)</b>	D2

### Schaltplan



## Blockdiagramm



## Code für Arduino

Für diesen Beispielcode wird die Bibliothek „RTC by Makuna“ verwendet. Der **Code benötigt keine manuelle Eingabe** und nimmt als Startpunkt die **Uhrzeit des Kompilervorgangs**.

```
// CONNECTIONS:
// DS1302 CLK/SCLK --> 5
// DS1302 DAT/IO --> 4
// DS1302 RST/CE --> 2
// DS1302 VCC --> 3.3v - 5v
// DS1302 GND --> GND

#include <ThreeWire.h>
#include <RtcDS1302.h>

ThreeWire myWire(4,5,2); // IO, SCLK, CE
RtcDS1302<ThreeWire> Rtc(myWire);

void setup ()
{
    Serial.begin(57600);

    Serial.print("compiled: ");
    Serial.print(__DATE__);
    Serial.println(__TIME__);

    Rtc.Begin();

    RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
    printDateTime(compiled);
    Serial.println();

    if (!Rtc.IsDateTimeValid())
    {
        // Common Causes:
        // 1) first time you ran and the device wasn't running yet
        // 2) the battery on the device is low or even missing

        Serial.println("RTC lost confidence in the DateTime!");
        Rtc.SetDateTime(compiled);
    }
}
```

```

    if (Rtc.GetIsWriteProtected())
    {
        Serial.println("RTC was write protected, enabling writing now");
        Rtc.SetIsWriteProtected(false);
    }

    if (!Rtc.GetIsRunning())
    {
        Serial.println("RTC was not actively running, starting now");
        Rtc.SetIsRunning(true);
    }

    RtcDateTime now = Rtc.GetDateTime();
    if (now < compiled)
    {
        Serial.println("RTC is older than compile time! (Updating
DateTime)");
        Rtc.SetDateTime(compiled);
    }
    else if (now > compiled)
    {
        Serial.println("RTC is newer than compile time. (this is
expected)");
    }
    else if (now == compiled)
    {
        Serial.println("RTC is the same as compile time! (not expected
but all is fine)");
    }
}

void loop ()
{
    RtcDateTime now = Rtc.GetDateTime();

    printDateTime(now);
    Serial.println();

    if (!now.IsValid())
    {
        // Common Causes:
        // 1) the battery on the device is low or even missing and the
power line was disconnected
        Serial.println("RTC lost confidence in the DateTime!");
    }

    delay(10000); // ten seconds
}

#define countof(a) (sizeof(a) / sizeof(a[0]))

void printDateTime(const RtcDateTime& dt)
{
    char datestring[20];

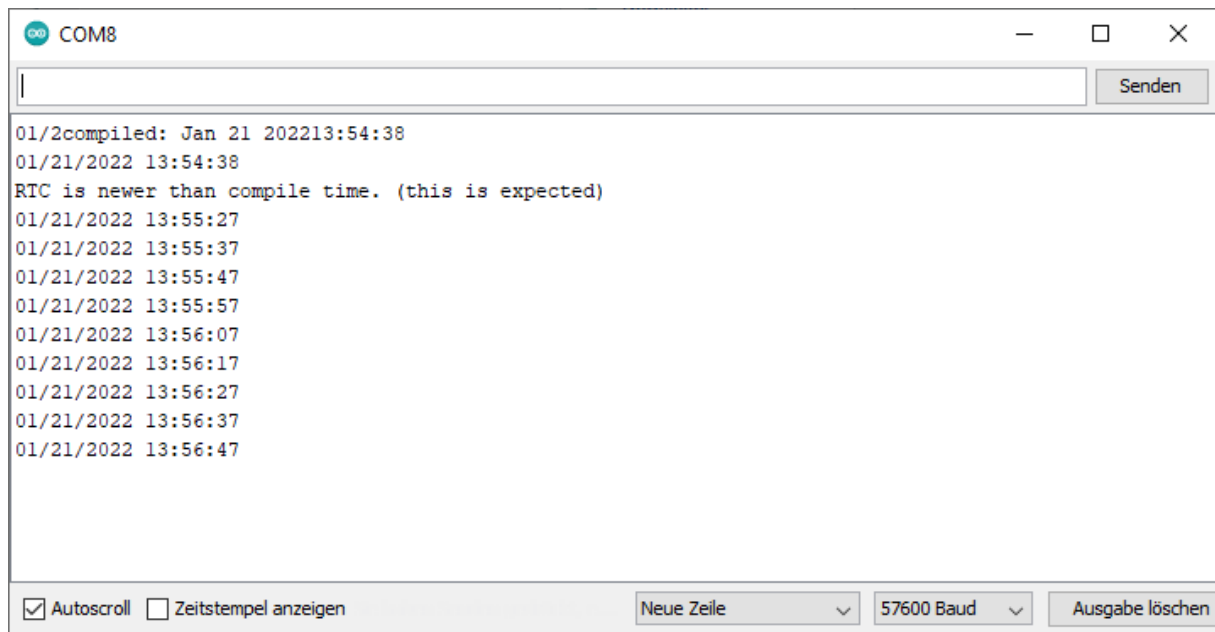
    snprintf_P(datestring,
        countof(datestring),
        PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
        dt.Month(),

```

```
dt.Day(),  
dt.Year(),  
dt.Hour(),  
dt.Minute(),  
dt.Second() );  
Serial.print(datestring);  
}
```

## Ergebnis

Nachdem der Code hochgeladen wurde, können Sie den **Seriellen Monitor** öffnen und die Uhrzeit ablesen.



## 10. Servomotor



### Benötigte Komponenten

- Arduino UNO kompatibles Board + USB-Kabel
- SG90 Micro Servomotor
- Breadboard
- Jumperkabel Stecker-Stecker
- Potentiometer oder Joystick

### Einleitung

In dieser Anleitung zeigen wir anhand der Servo.h Bibliothek wie man einen Servomotor mit einem Arduino kompatiblen Board ansteuert. Servomotoren sind sehr vielseitig einsetzbar, sie sind beliebt in RC-Modellen, Robotern, Automaten und bringen Bewegung in sämtlichen DIY-Elektronik Projekten.

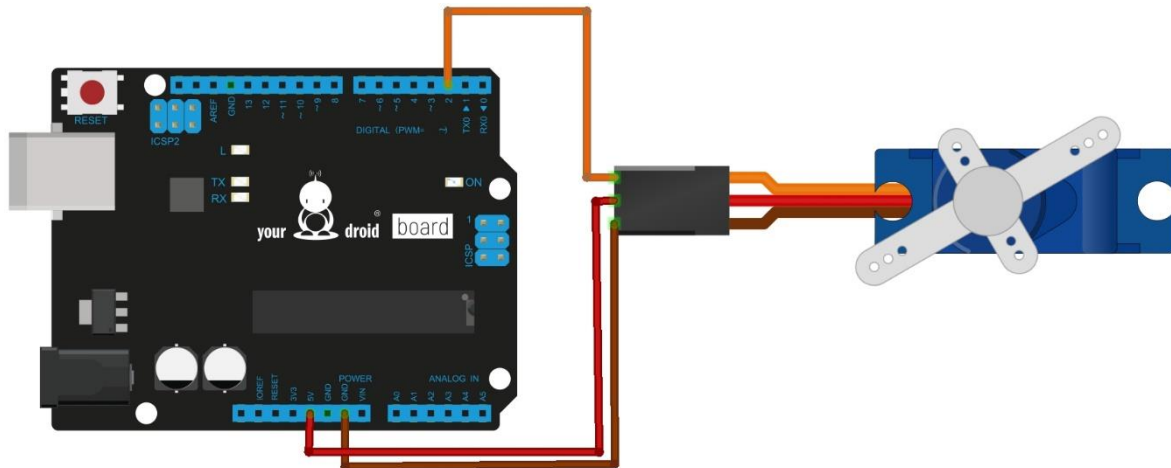
### Funktionsweise

Ein Servomotor (kurz Servo) ist ein Elektromotor, der sehr genau angesteuert werden kann. Neben der exakten Winkelposition kann auch die Drehgeschwindigkeit und Beschleunigung gesteuert werden. Der SG90 Micro Servo besteht aus einem kleinen DC-Motor, einem Getriebe und einer Steuerplatine mit Potentiometer.

Die Steuerplatine setzt die Signale in genaue Befehle um und über das Potentiometer wird die Position überprüft. Die üblichen Hobbyservos besitzen drei Pins zur Ansteuerung: GND, VCC und PWM. Gelegentlich findet man einen vierten Pin vor, dieser dient als Feedback-Pin, um die Position des Potentiometers über einen Microcontroller abzurufen.



## Anschlussplan



fritzing

Servomotor	Arduino
<b>Gelb (PWM-Signalleitung)</b>	Pin 2
<b>Rot (VCC)</b>	5V
<b>Braun (GND)</b>	GND

## Code

### Beispielcode zum Testen

Der Beispielcode stammt aus der Servo.h Bibliothek. Die Bibliothek ist bereits in der Arduino IDE integriert und muss nicht mehr heruntergeladen werden. Mit diesem Code lassen sich Servos schnell und einfach auf Funktion testen.

```

/* Sweep
  by BARRAGAN <http://barraganstudio.com>
  This example code is in the public domain.

  modified 8 Nov 2013
  by Scott Fitzgerald
  http://www.arduino.cc/en/Tutorial/Sweep
  */

#include <Servo.h>

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(2);  // attaches the servo on pin 2 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180
    degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in
    variable 'pos'
  }
}

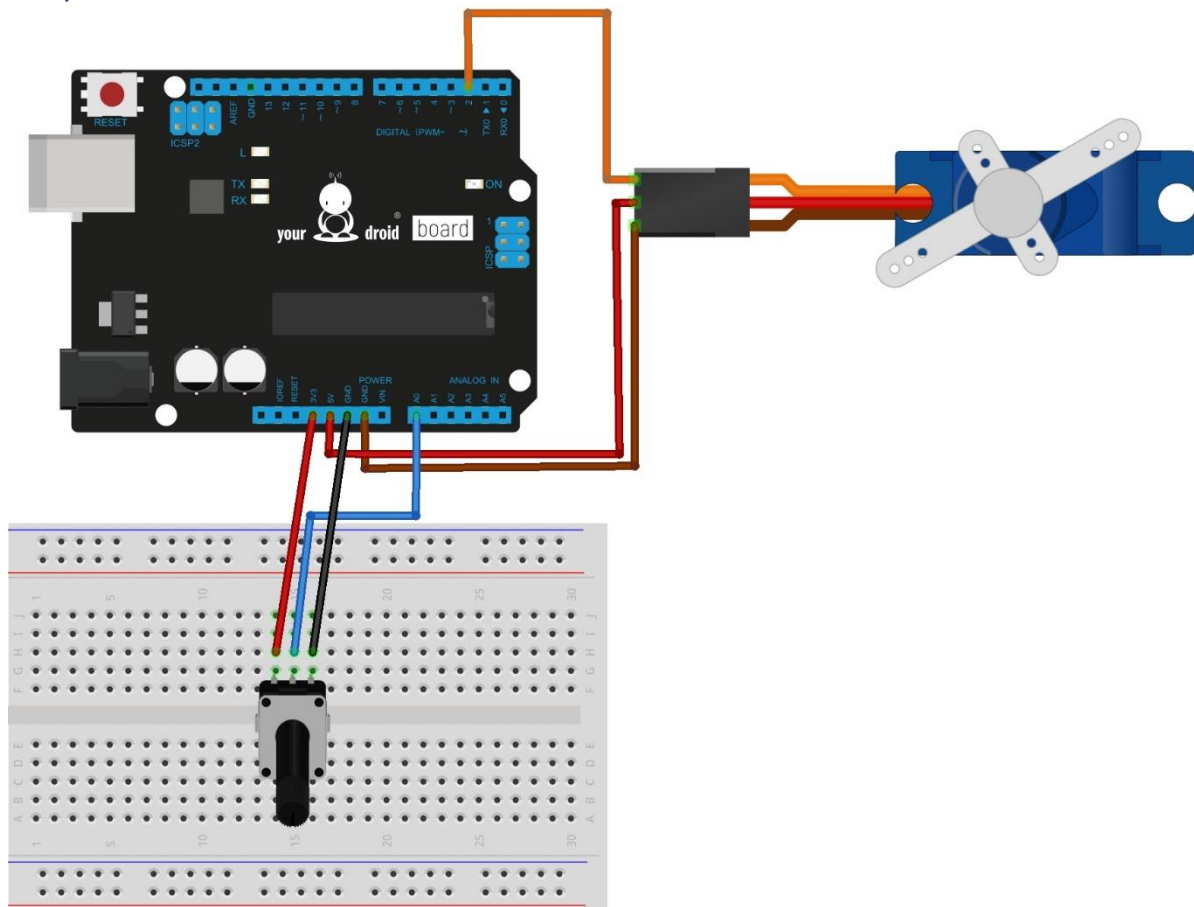
```

```

    delay(15); // waits 15ms for the servo to reach
the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0
degrees
    myservo.write(pos); // tell servo to go to position in
variable 'pos'
    delay(15); // waits 15ms for the servo to reach
the position
  }
}

```

### Beispiel mit Potentiometer



fritzing

Das Potentiometer kann auch durch einen Joystick ersetzt werden. A0 wird dann an VRx oder VRy angeschlossen.

```

/*
Controlling a servo position using a potentiometer (variable resistor)
by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>

modified on 8 Nov 2013
by Scott Fitzgerald
http://www.arduino.cc/en/Tutorial/Knob

```

```
*/  
  
#include <Servo.h>  
  
Servo myservo; // create servo object to control a servo  
  
int potpin = 0; // analog pin used to connect the potentiometer  
int val; // variable to read the value from the analog pin  
  
void setup() {  
  myservo.attach(2); // attaches the servo on pin 2 to the servo object  
}  
  
void loop() {  
  val = analogRead(potpin); // reads the value of the  
  potentiometer (value between 0 and 1023)  
  val = map(val, 0, 1023, 0, 180); // scale it to use it with the  
  servo (value between 0 and 180)  
  myservo.write(val); // sets the servo position  
  according to the scaled value  
  delay(15); // waits for the servo to get  
  there  
}
```

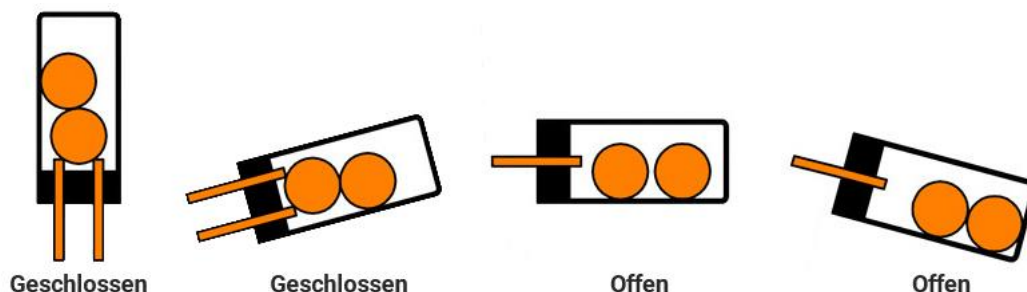
## 11. Neigungsschalter



### Übersicht

Neigungssensoren (englisch tilt switch) ermöglichen es Ihnen, die Ausrichtung oder Neigung zu erkennen. Sie sind klein, kostengünstig, stromsparend und einfach zu bedienen. Außerdem verschleßen sie bei richtiger Anwendung nicht. Die Einfachheit macht sie beliebt für Spielzeug, Gadgets und Haushaltsgeräte.

Manchmal werden sie auch „Quecksilberschalter“, „Neigungsschalter“ oder „Neigungswinkelschalter“ genannt. Früher waren die Neigungsschalter noch Quecksilberschalter, bei denen ein mit einem Tropfen Quecksilber gefülltes Glasrohr mit zwei elektrischen Kontakten versehen war. Mittlerweile wird statt Quecksilber rollende Kugeln verwendet. Am Ende vom Hohlraum sind zwei leitfähige Elemente (Pole), die ab einer gewissen Neigung von der Kugel verbunden werden, damit der Schaltzustand erreicht wird.



### Vorteile von Neigungsschaltern

Neigungsschalter sind zwar nicht so präzise oder flexibel wie ein vollständiger Beschleunigungsmesser, können jedoch Bewegungen und Ausrichtungen erkennen. Ein weiterer Vorteil ist, dass die größeren Neigungsschalter den Strom selbst schalten können. Beschleunigungsmesser geben hingegen eine digitale oder analoge Spannung aus, die dann mit einer zusätzlichen Schaltung analysiert werden muss. Ein gutes Beispiel für den Einsatz von Beschleunigungssensoren sind Controller mit einer Bewegungssteuerung (z.B. von der Switch oder Wii Konsole).

### Nachteile von Neigungsschaltern

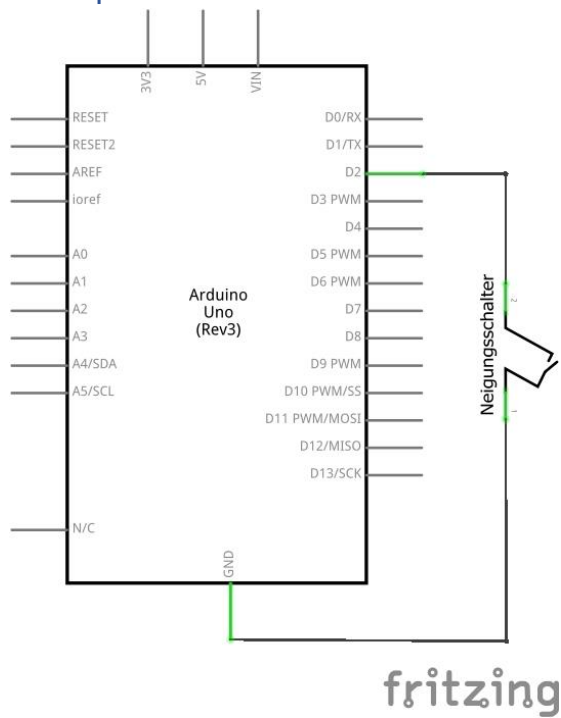
- Quecksilberschalter sind giftig und nicht mehr zeitgemäß
- Vibrationen können Fehleingaben erzeugen

- Ohne Schwerkraft funktionieren die Schalter nicht
- Funktion ist in beweglichen Projekten nicht zuverlässig

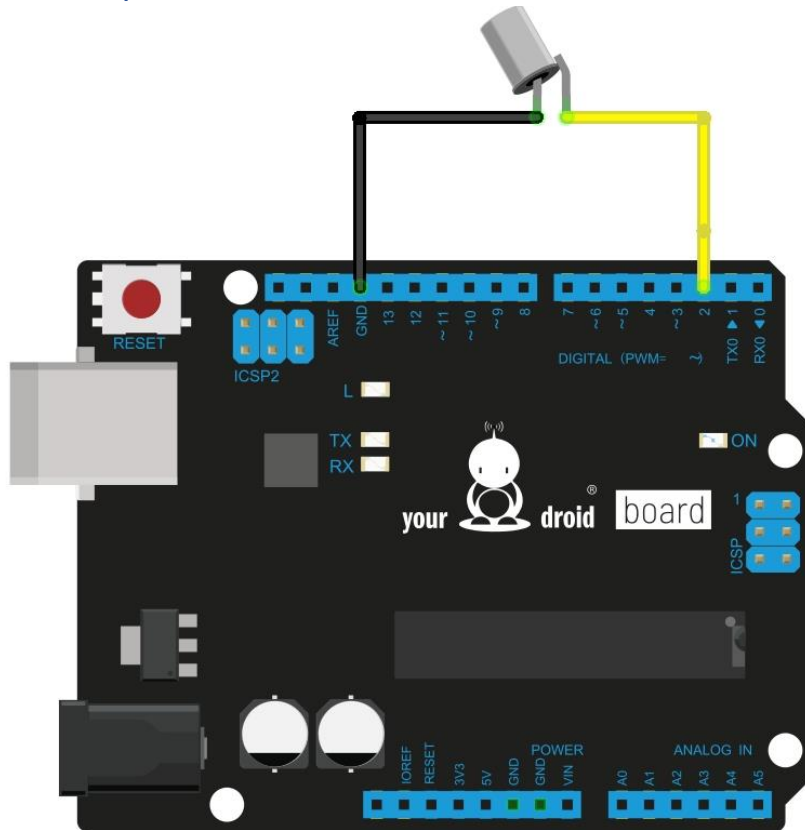
## Benötigte Komponenten

- Uno R3 kompatibles Board
- Neigungsschalter
- 2x Dupontkabel Buchse-Stecker

## Schaltplan



## Anschlussplan



fritzing

## Beispielcode

Mit diesem Beispielcode bringen wir die integrierte LED (Beschriftung L auf der Platine) zum Aufleuchten, wenn der Neigungsschalter schaltet. Dieses Beispiel ist etwas komplizierter als üblich, funktioniert durch die [Debounce-Funktion](#) aber wesentlich zuverlässiger.

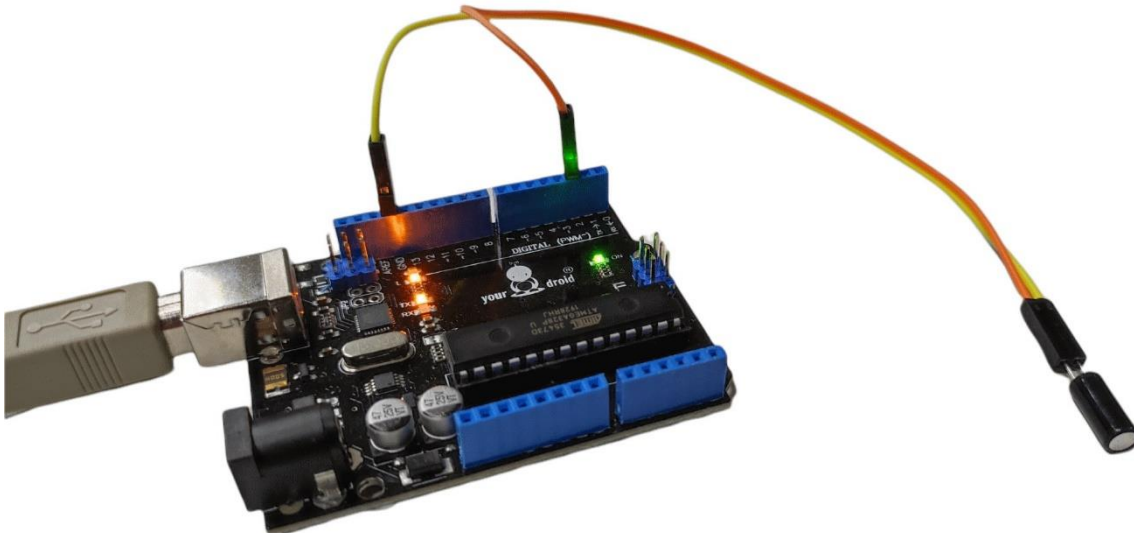
```
// Pins werden deklariert
const int ledPin = 13; // Pin für die LED
const int sensorPin = 2; // Pin für den Neigungsschalter

//Variablen werden deklariert
int sensorState; // Aktueller Status vom Sensor
int lastTiltState = HIGH; // Letzter Status vom Sensor
int sensorWert; // Variable um den Messwert einzulesen
unsigned long lastDebounceTime = 0; // Letzte Zeit als der Sensor
ausgelöst wurde
unsigned long debounceDelay = 50; // Entprellwert in Millisekunden um
Fehleingaben zu vermeiden

void setup()
{
  pinMode(ledPin, OUTPUT); // Initialisiere den LED Pin als Ausgang
  pinMode(sensorPin, INPUT); // Initialisiere den Neigungsschalter als
  Eingang
  digitalWrite(sensorPin, HIGH); // Setze den Neigungsschalter auf HIGH
}
```

```
digitalWrite(ledPin,LOW); // Setze die LED auf LOW
Serial.begin(9600); // Serielle Verbindung mit Baud 9600 öffnen
}

void loop() {
  sensorWert = digitalRead(sensorPin); // Sensorwert lesen
  // Debounce (Entprellen) vom Sensorwert, um Fehleingaben zu verhindern
  if (sensorWert == lastTiltState) {
    // Debounce timer zurücksetzen
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // Wenn der Sensorwert länger vorhanden ist als der Debounce Delay, wird
    // der Sensorwert übernommen
    lastTiltState = sensorWert;
  }
  //Wenn der Sensorwert 1 (HIGH) ist, dann wird die LED eingeschaltet.
  //Ansonsten ist die LED aus
  if (sensorWert == 1) {
    digitalWrite(ledPin,HIGH); // LED einschalten
  }
  else
  {
    digitalWrite(ledPin,LOW); // LED ausschalten
  }
  Serial.println(sensorWert); // Ausgabe vom Sensorwert
  delay(500);
}
```



## 12. Fotowiderstand



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Fotowiderstand
- 1k Widerstand
- 3 Jumperkabel
- Breadboard

### Übersicht

In dieser Anleitung zeigen wir Ihnen, wie Sie einen Fotowiderstand als **Lichtsens**or verwenden, um die Umgebungshelligkeit zu messen. Lichtsensoren sind besonders praktisch, um in Elektronikprojekten die Tageszeit zu berücksichtigen oder Aktoren auszulösen. Typische Anwendungen wären intelligente Lichtsteuerungen von Lampen, die Steuerung von Sonnenschutz wie Rollläden und Jalousien oder die automatische Helligkeit von digitalen Displays.

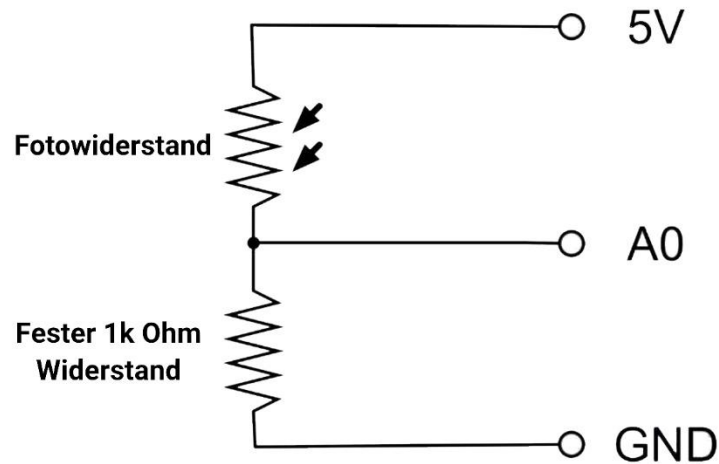
### Funktionsweise

Ein Fotowiderstand ist lichtempfindliches Bauteil, es wird auch oft mit LDR abgekürzt, Englisch für **Light Dependent Resistor**. Wie der Name bereits andeutet, verändert der LDR je nach Helligkeit seinen Widerstand. Dieser Fotowiderstand hat ca. **50k Ohm in Dunkelheit** und **500 Ohm** in hellem Licht.

Unser Arduino UNO R3 Board kann nur analoge Signale (Spannungen) erfassen, mit dem reinen Widerstand kann es nichts anfangen.

Um den Widerstand in analoge Messwerte umzuwandeln, verwenden wir einen normalen Widerstand in Reihe geschaltet. So kann der analoge Eingang am Arduino eine variierende Spannung auslesen, die wir dann in Messwerte umwandeln.



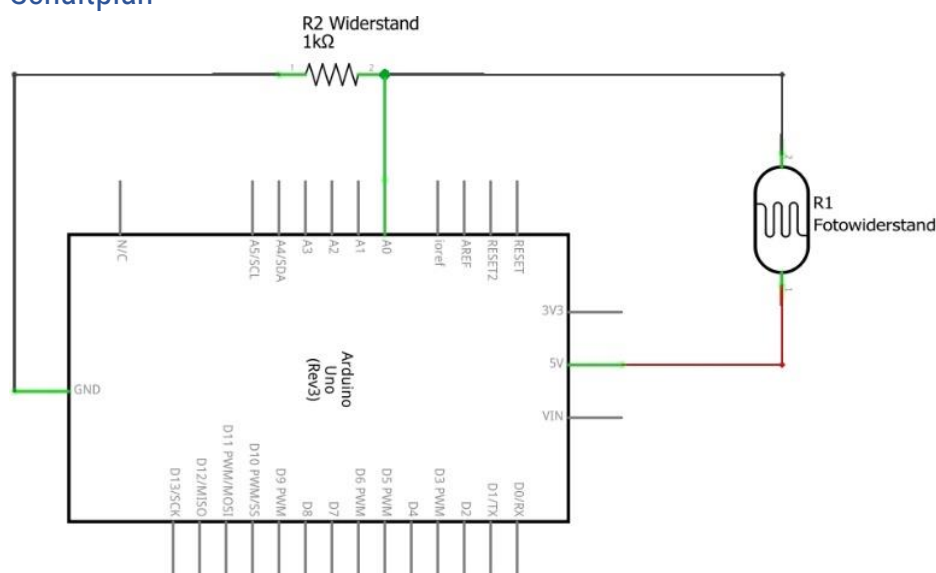


Wenn zwei Verbraucher zwischen 5V und GND in Reihe geschaltet werden, teilen Sie sich die Spannung. Dieses Prinzip machen wir uns zunutze, um zwischen den beiden Widerständen die Spannung zu ermitteln.

Unter hellem Licht wird der Widerstand vom Fotowiderstand gering, so dass der 1k Widerstand Richtung GND zieht und die vollen 5V durchfließen können. Wenn das Licht schwächer wird, erhöht sich der Widerstand vom Fotowiderstand und wird größer als der feste 1k Widerstand. Dadurch wird die gemessene Spannung kleiner, als würde man einen Regler herunterdrehen.

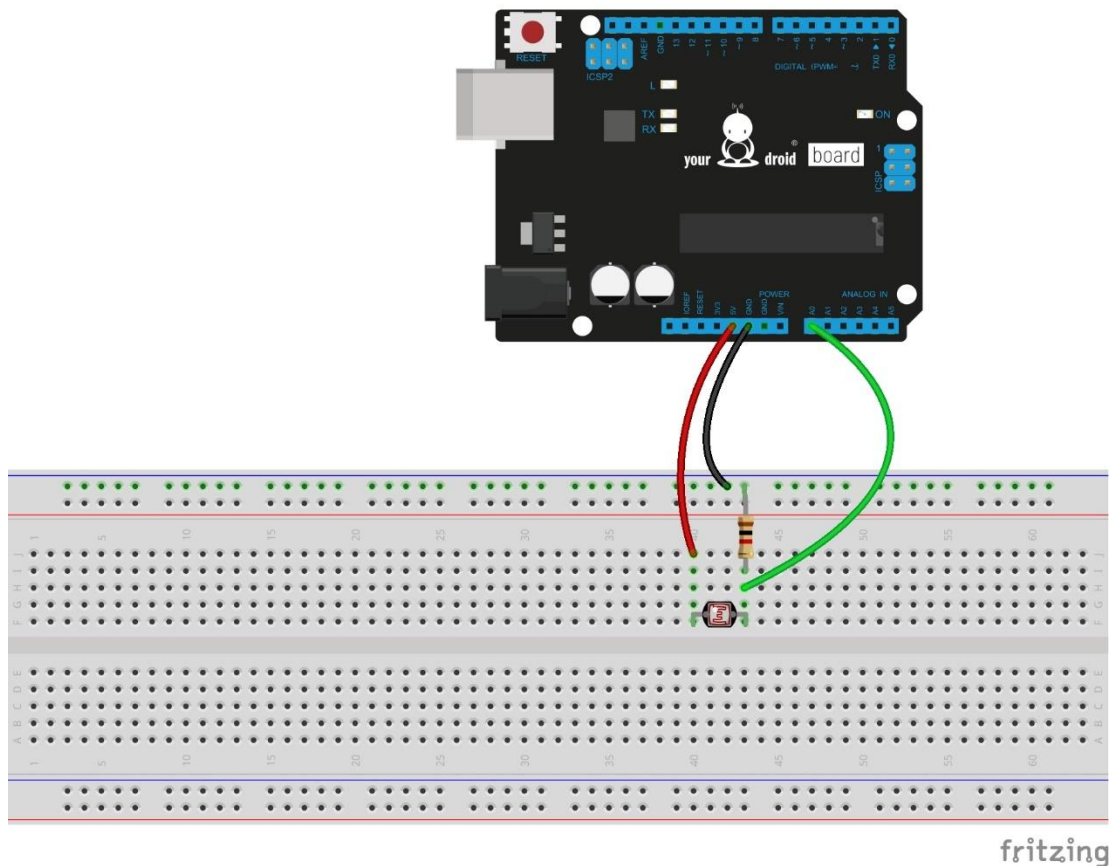
Die gemessene Spannung wird am analogen Eingang A0 des Arduinos in Zahlen von 0 bis 1023 umgewandelt. Hier entspricht der Messwert von 1023 unseren 5V - also eine hohe Helligkeit. Die Zahl 0 entspricht 0 Volt - also völlige Dunkelheit

## Schaltplan



fritzing

## Anschlussplan



Das erste Beinchen des Fotowiderstands wird an 5V verkabelt. Das zweite Beinchen mit einem Widerstand+ Kabel an GND und mit einem weiteren Kabel an A0.

## Code

Mit diesem Beispielcode können Sie die Messwerte im seriellen Monitor auslesen. Ab dem Schwellenwert 512, also ca. der Hälfte der maximalen Helligkeit geht die onboard LED auf dem Arduino UNO aus.

Probieren Sie es aus und halten sie die Hand über den Fotowiderstand. Beobachten Sie im seriellen Monitor der Arduino IDE, wie sich die Messwerte verändern.

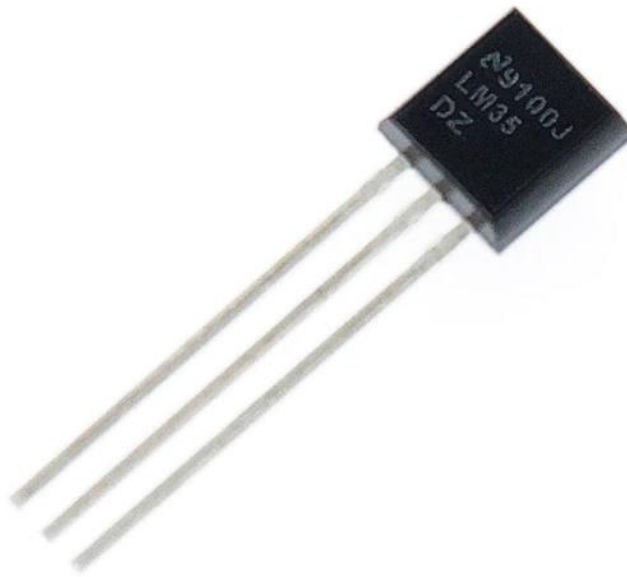
```
int LDR = A0; // Analoger Pin als LDR Eingang
int LED = 13; // LED-Pin
int sensorWert = 0; // Variable für den Sensorwert mit 0 als Startwert

void setup()
{
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop()
{
  sensorWert = analogRead(LDR); //
```

```
Serial.println(sensorWert); //Ausgabe am Serial-Monitor.  
  
if (sensorWert > 512 ) // Wenn der Schwellenwert 512 überschritten wird,  
soll die LED leuchten  
{  
  digitalWrite(LED, HIGH); // LED an  
}  
// ansonsten...  
else{  
  digitalWrite(LED, LOW); // LED aus  
  delay (50);  
}  
}
```

## 13. LM35 Temperatursensor



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- LM35 Temperatursensor
- 3 Jumperkabel
- Breadboard

### Übersicht

Der LM35 ist ein geläufiger und einfach zu nutzender Temperatursensor. Er benötigt keine weitere Hardware und kann mit einem Arduino direkt ausgelesen werden. Der Sensor besitzt eine Genauigkeit von  $\pm 0,5^{\circ}\text{C}$  bei Raumtemperatur und  $\pm 1^{\circ}\text{C}$  über die gesamte Messreichweite von  $0-150^{\circ}\text{C}$ . Der größte Nachteil von diesem Sensor ist, dass er eine negative Vorspannung zum messen negativer Temperatur en unter  $0^{\circ}\text{C}$  benötigt. Für Projekte, die auch negative Temperaturen messen sollen, empfehlen wir daher einen DS18B20B Temperatursensor.

### Funktionsweise

Der LM35 Temperatursensor liefert eine Ausgangsspannung zwischen 0 und 1,5V, die linear proportional zur Temperatur ist. Diese Spannung kann man am analogen Eingang vom Arduino auslesen. Die Schwierigkeit liegt im Schreiben des Codes, um die analogen Werte in Grad Celsius zu konvertieren.

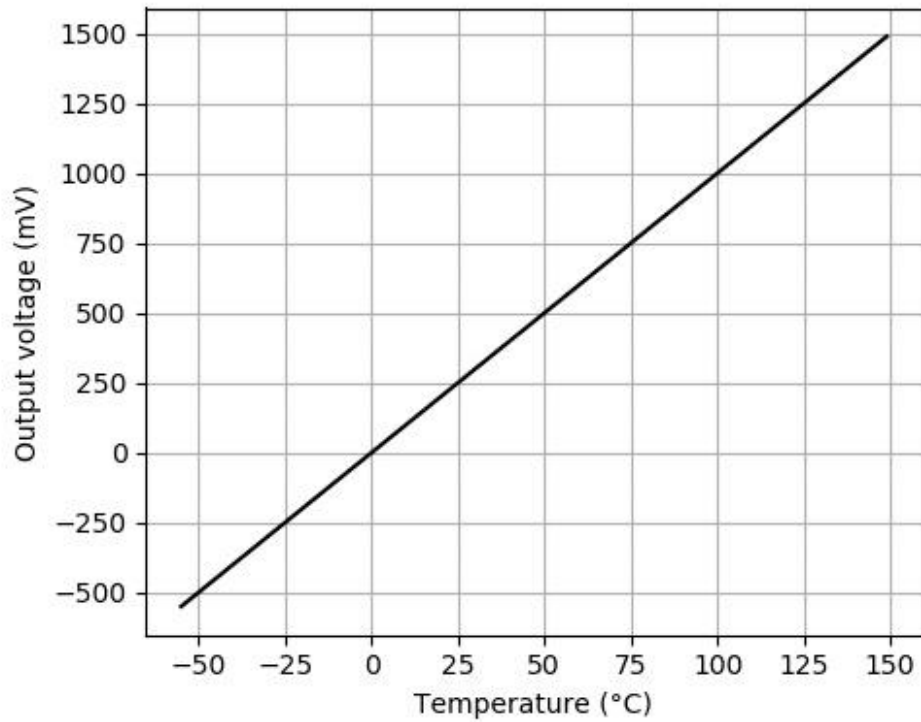
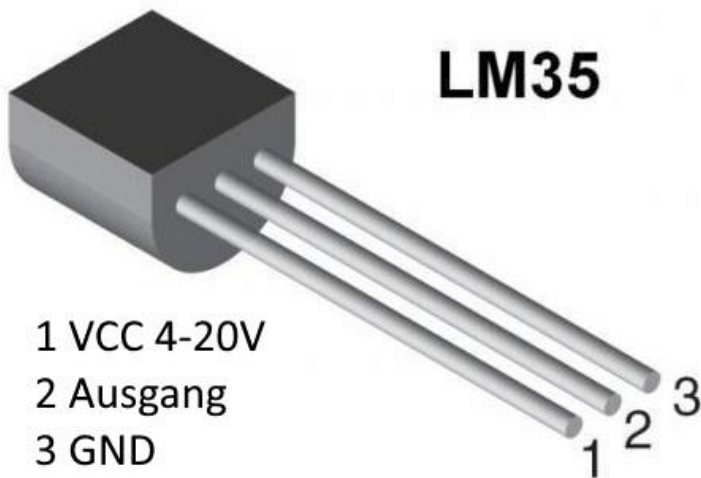
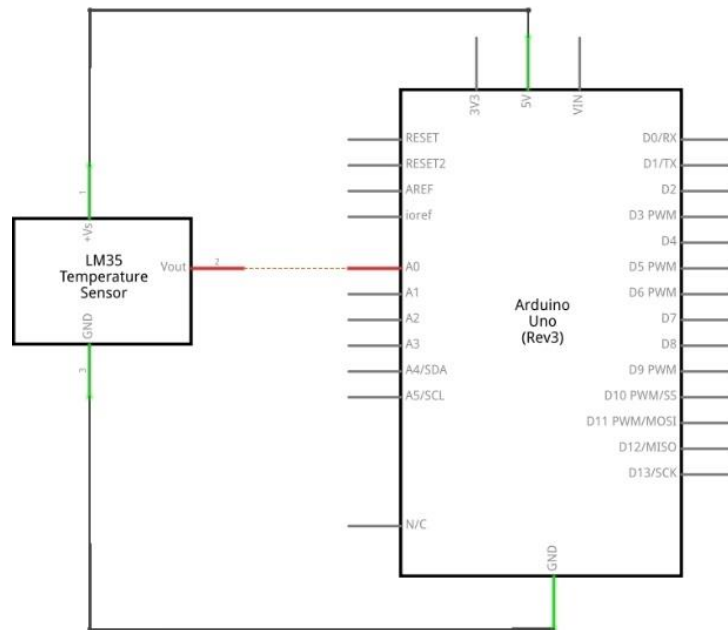


Abbildung 2 Output (mV) in Relation zur Temperatur in °C

### Pinbelegung

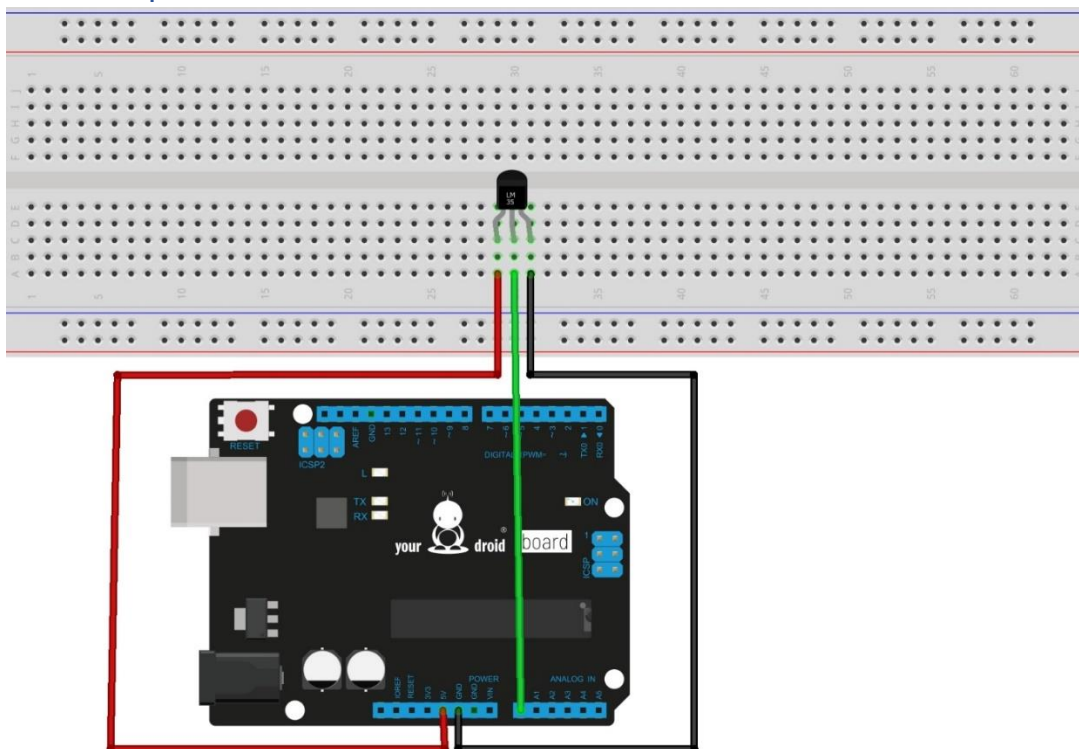


## Schaltplan



fritzing

## Anschlussplan



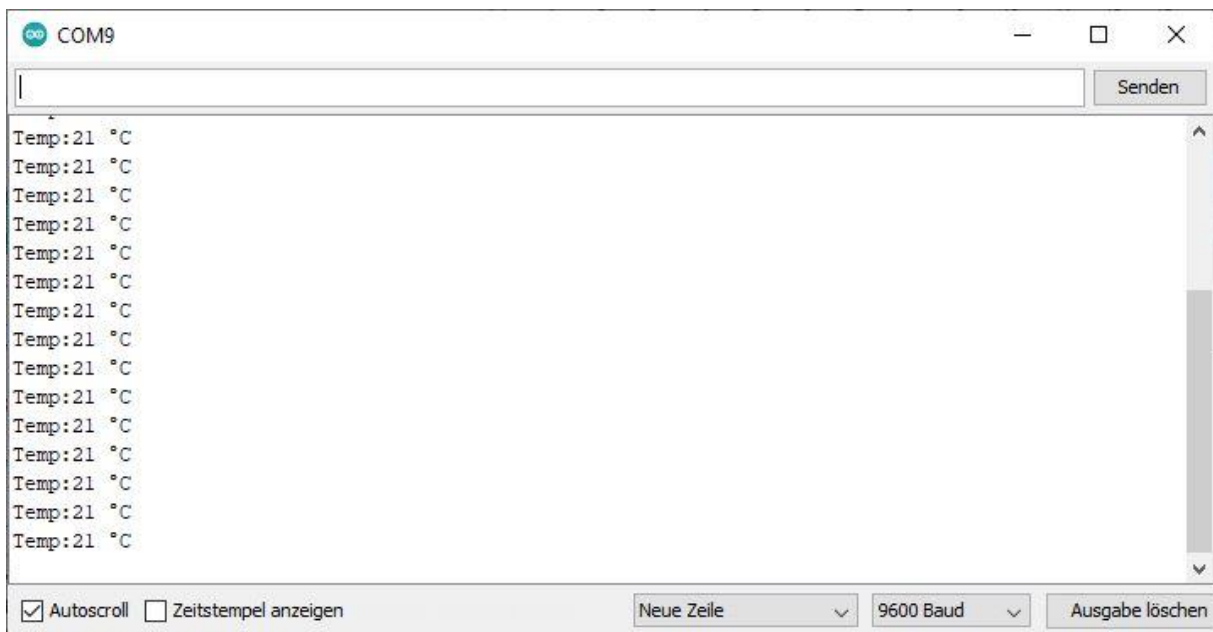
fritzing

LM35	Arduino
1 VCC	5V
2 OUT (Datenleitung)	A0
3 GND	GND

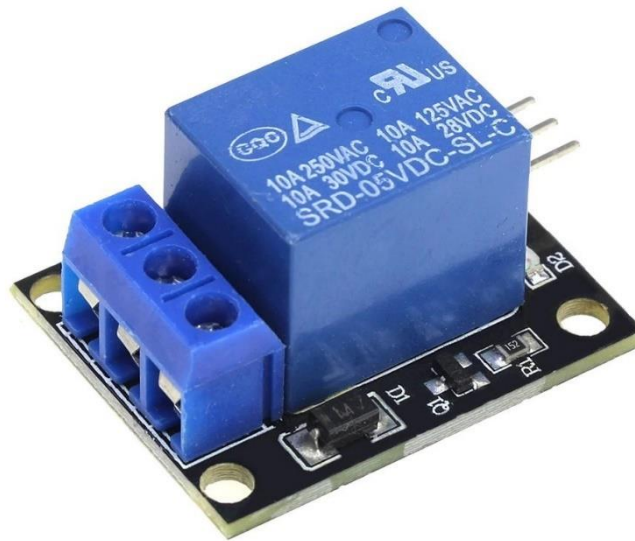
## Code

```
int LM35 = A0;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  int val;
  int dat;
  val=analogRead(LM35);
  dat=(125*val)>>8;
  Serial.print("Temp:");
  Serial.print(dat);
  Serial.println(" °C");
  delay(500);
}
```

Nachdem das Programm hochgeladen wurde, können Sie die Temperatur in Grad Celsius im seriellen Monitor der Arduino IDE auslesen.



## 14. Relais Modul



### Benötigte Komponenten

- Arduino UNO kompatibles Board
- Relais Modul
- 5mm LED
- Breadboard
- 220 Ohm Widerstand

### Übersicht

Mit diesem 1Kanal Relais 5V Modul können Sie eine große Last einfach über den 5V-Anschluss eines Mikrocontrollers, wie einem Raspberry Pi oder Arduino, ansteuern und schalten. Das Relais kann verwendet werden um leistungsstarke Elektronik wie zum Beispiel AC- oder DC-Motoren zu steuern. Dabei ist es durch den einen digitalen Eingang besonders einfach in der Handhabung. Über die Header Pins kann 1 Kanal 5V Relais einfach mit Dupont-Kabeln an den Arduino angeschlossen werden. Die zu schaltende Last wird über die Schraubklemmen angeschlossen. Die Ansteuerung besteht aus VCC (+), GND (-) und dem Signal Pin (S). Das Relais schaltet bei anliegenden 5V auf dem Eingang (High-Level).



## Pinbelegung

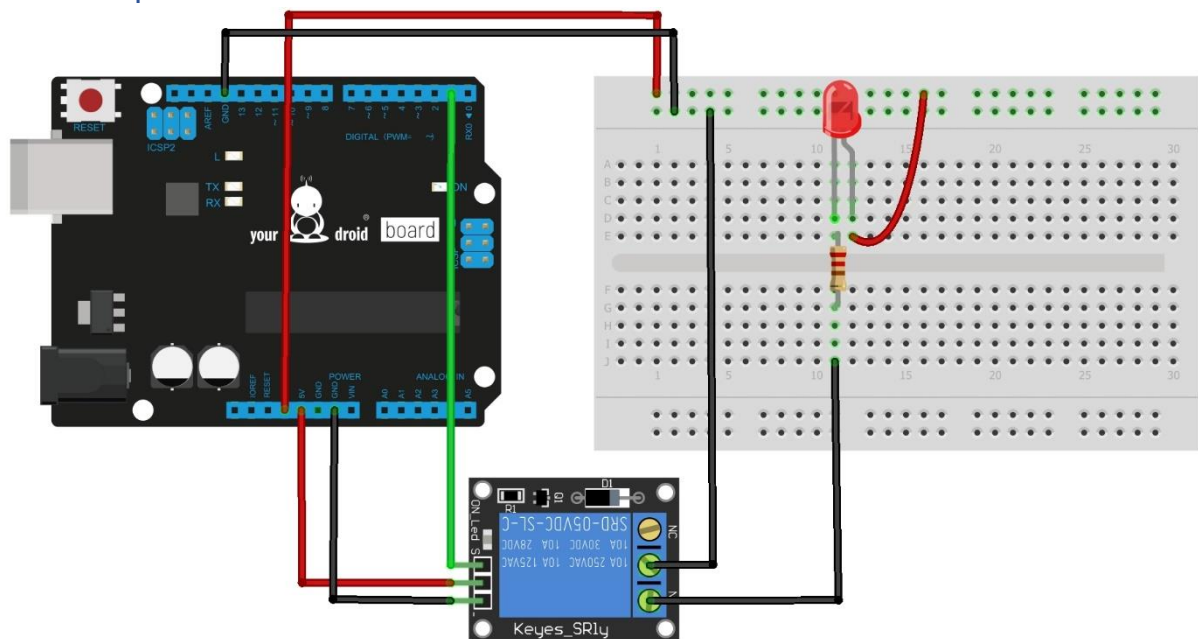


**COM:** Gemeinsamer Pin

**NC (Normally Closed):** Bei der NC-Konfiguration, ist das Relais standardmäßig geschlossen. Es fließt Strom, bis ein Signal den Stromkreis öffnet.

**NO (Normally Open):** Bei der NO-Konfiguration ist das Relais immer offen. Es fließt kein Strom, bis ein Signal den Stromkreis schließt.

## Anschlussplan



fritzing

Relais Modul	Arduino UNO
<b>S</b>	D2
<b>GND -</b>	GND
<b>VCC +</b>	5V

## Code

```
const int RELAIS = 2; // Relais-Pin am Arduino

void setup() {
  pinMode(RELAIS, OUTPUT); // Relais Pins als Ausgang deklarieren
}

void loop() {
  digitalWrite(RELAIS, HIGH); //RELAIS an

  delay(500); //500ms Pause

  digitalWrite(RELAIS, LOW); //RELAIS an
  delay(5000); // 500ms Pause
}
```

## 15. Flammensensor



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Flammensensor
- Buzzer
- 10k Widerstand
- 7 Jumperkabel
- Breadboard

### Übersicht

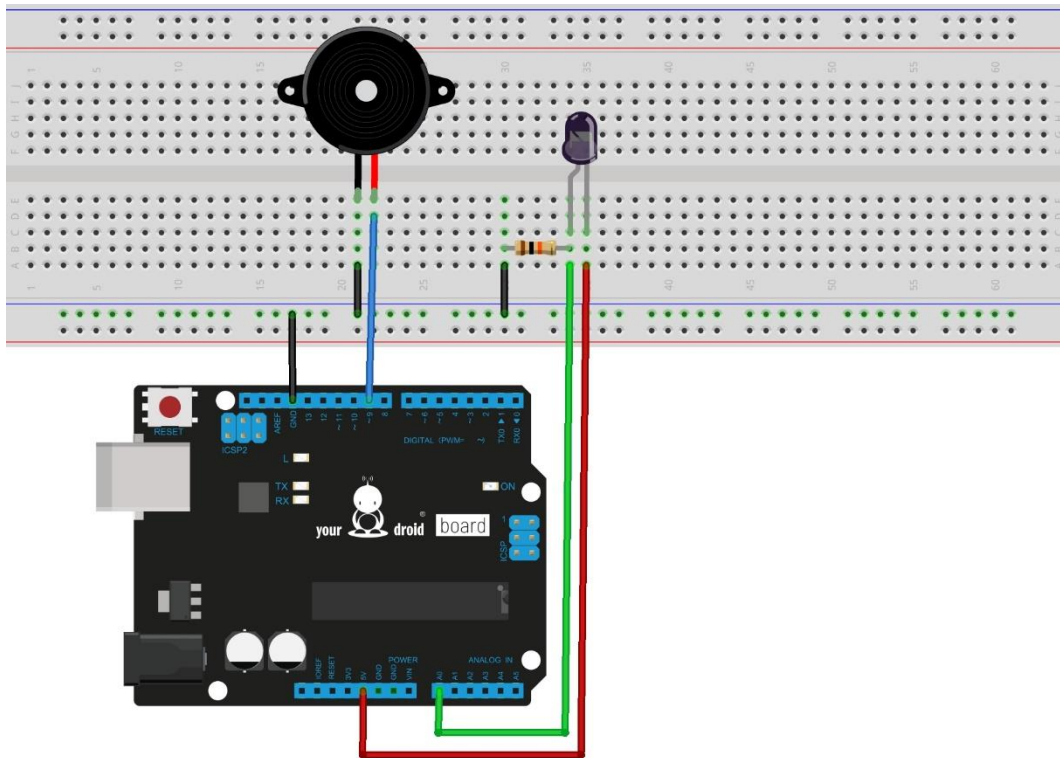
Dieser Flammensensor ist eigentlich eine Infrarot-Fotodiode in einem dunklen Gehäuse, die besonders empfindlich gegenüber dem Licht ist, welches von Feuer erzeugt wird.

### Funktionsweise

Dieser Sensor reagiert besonders empfindlich auf das Infrarot-Lichtspektrum, welches von Feuer erzeugt wird. Der Flammensensor erkennt Licht mit einer Wellenlänge von 760 nm bis 1100 nm, welches genau dem Spektrum von einer offenen Flamme entspricht.

### Anschlussplan

Das kürzere Beinchen der Fotodiode ist für den Minuspol, das längere Beinchen für den Pluspol. In diesem Szenario schließen wir den **Minuspol (kurzes Beinchen) an 5V**, den Pluspol mit einem Widerstand auf GND und dazwischen unser Jumperkabel auf den analogen Eingang A0.



fritzing

## Code

```
int Sensor=0; //Anschluss vom Flammensensor
int Buzzer=9; // Pluspol vom Buzzer
int val=0; // Variable fuer den Messwert
void setup()
{
  pinMode(Buzzer, OUTPUT);
  pinMode(Sensor, INPUT);
  Serial.begin(9600);
}
void loop()
{
  val=analogRead(Sensor);
  Serial.println(val);
  if(val>=600) // Wenn der Messwert größer als 600 ist, soll der Buzzer
angehen.
  {
    digitalWrite(Buzzer, HIGH);
  }else
  {
    digitalWrite(Buzzer, LOW);
  }
  delay(500);
}
```

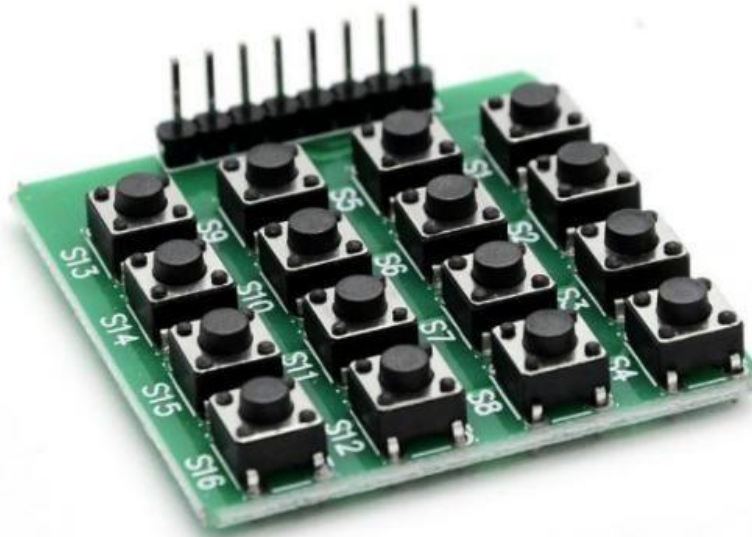
## Experiment

Versuchen Sie mit einem Feuerzeug den Sensor auszulösen. Sie können den aktuellen Messwert im seriellen Monitor in der Arduino IDE auslesen.

Wenn der Sensor eine Flamme erkennt, steigt die Spannung an unserem analogen Eingang. Ohne Feuer in der Nähe, erhalten wir eine Spannung von ca. 0,3V. Wenn der Sensor eine Flamme erkennt, steigt die Spannung auf ca. 1V, je näher sich das Feuer befindet, desto höher die anliegende Spannung an unserem analogen Eingang.

Ab dem Schwellenwert von 600 soll der Buzzer angehen und uns alarmieren.

## 16. Keypad Modul



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- 4x4 Keypad Modul
- Buzzer
- 220 Ohm Widerstand
- 11 Jumperkabel
- Breadboard

### Beschreibung

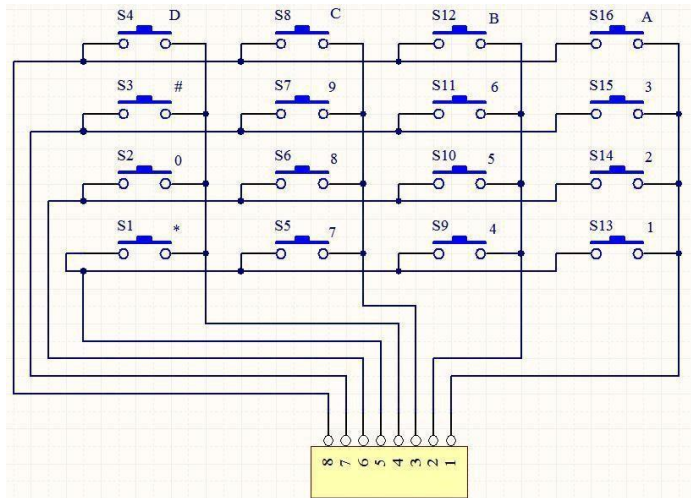
Tastenfelder oder Keypads eignen sich perfekt zur Dateneingabe oder Menü-Navigation in Projekten wie Sicherheitssysteme, Zugangskontrollen, Embedded Systems, Smart Home Steuerungen, Fernbedienungen und Robotern. Die Einsatzmöglichkeiten sind unbegrenzt, da die Tasten nicht unbedingt die aufgedruckten Zeichen ausgeben müssen, sondern mit Hilfe eines Arduinos oder Raspberry Pis frei programmiert werden können. Am beliebtesten sind Tastenfelder mit 4x3 oder 4x4 Tasten, die einer klassischen Telefontastatur ähneln.

### Funktionsweise

Das numerische Tastenfeld besteht aus 16 Tasten, die in einer Matrix angeordnet sind, d. h. alle Tasten einer Spalte sind mit einem Eingang und alle Tasten einer Zeile mit einem anderen verbunden.

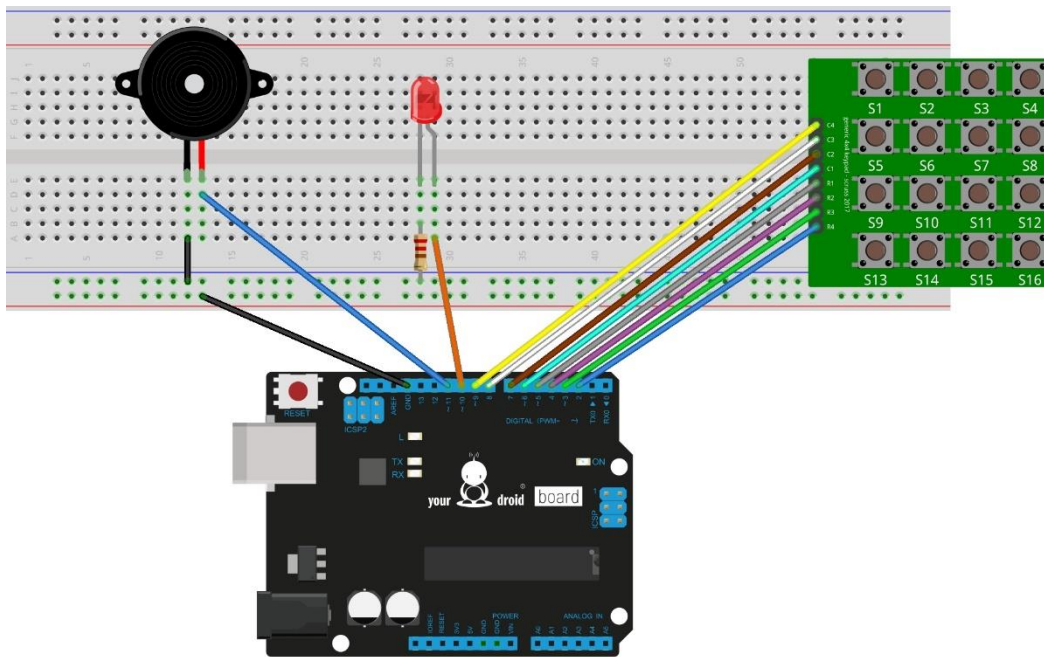
Wenn eine Taste gedrückt wird, wird der entsprechende Eingang der Zeile mit dem der Spalte verbunden, wodurch der Stromkreis geschlossen wird. Der Vorteil dieses Aufbaus ist, dass 16 Tasten mit nur 8 Eingängen des Mikrocontrollers angesteuert werden können.

## Schaltplan



## Anschlussplan

Keypad	Arduino
<b>C1</b>	Pin 9
<b>C2</b>	Pin 8
<b>C3</b>	Pin 7
<b>C4</b>	Pin 6
<b>L1</b>	Pin 5
<b>L2</b>	Pin 4
<b>L3</b>	Pin 3
<b>L4</b>	Pin 2



fritzing

## Code für Arduino

Zur Verwendung von diesem Beispielcode wird die Keypad.h Bibliothek benötigt.

Mit diesem Beispiel können Sie eine Taste drücken und die gedrückte Taste im seriellen Monitor der Arduino IDE auslesen. Die Taste 1 lässt die LED aufleuchten und die Taste 2 betätigt den Buzzer.

```
#include <Keypad.h>
int led = 10;
int buzzer = 11;
const byte ROWS = 4; // Definiere vier Reihen
const byte COLS = 4; // Definiere vier Spalten
char keys[ROWS][COLS] = {
  {'4', '8', 'C', 'G'},
  {'3', '7', 'B', 'F'},
  {'2', '6', 'A', 'E'},
  {'1', '5', '9', 'D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; // Pins fuer die Reihen
byte colPins[COLS] = {5, 4, 3, 2}; // Pins fuer die Spalten
// Initialisiere Keypad Bibliothek
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
void setup() {
  Serial.begin(9600); pinMode(led, OUTPUT); pinMode(buzzer, OUTPUT);
}

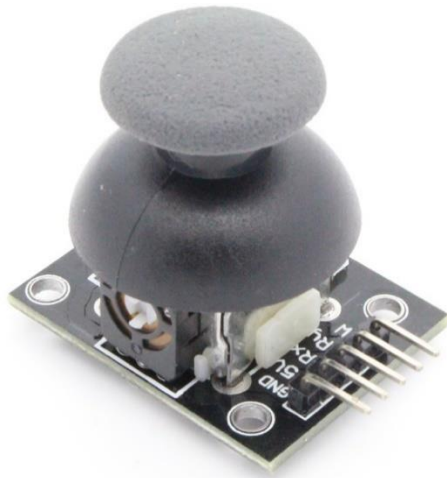
void loop() { digitalWrite(led, LOW); digitalWrite(buzzer, LOW); char key
= keypad.getKey(); if (key != NO_KEY) { Serial.println(key);
}
if (key == '1') // Wenn die Taste 1 gedrückt wird, soll die LED leuchten
{
  digitalWrite(led, HIGH); delay(1000);
}

if (key == '2') // Wenn die Taste 2 gedrückt wird, soll der Buzzer piepsen
```



```
{  
digitalWrite(buzzer, HIGH); delay(1000);  
  
}  
}
```

## 17. Joystick Modul



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Joystick Modul
- 5 Buchse-Stecker Kabel

### Beschreibung

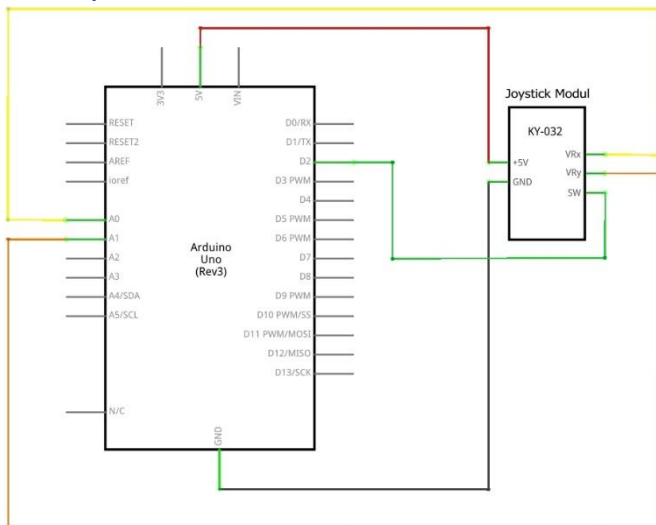
Mit Joysticks bringen Sie Ihre Projekte unter analoge Kontrolle! Dieses Joystick Modul eignet sich perfekt für eine 2-Achsen-Steuerung wie Sie häufig in Roboterarmen verwendet wird. Das Modul macht sich ebenfalls super als Eingabemethode für Spiele und Menüs.

Dieses Joystick Modul besitzt 5 Pins: GND, 5V, VRx, VRy und SW. Die Beschriftung kann je nach Charge variieren, die Belegung und Funktion bleibt aber die gleiche. Der Joystick arbeitet analog und liefert genauere Werte als digitale Joysticks. Er verfügt über einen integrierten Taster, der durch Drücken ausgelöst werden kann. Dadurch lassen sich z.B. Umschaltfunktionen (Toggle) oder Auswahlfunktionen (Select) programmieren.

### Funktionsweise

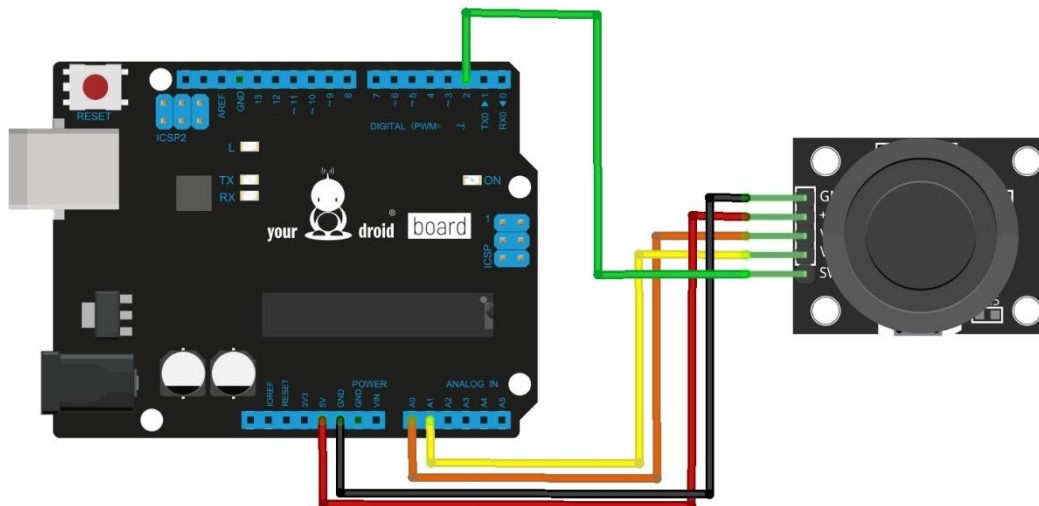
Am Arduino werden die Daten der X- und Y-Pins über analoge Eingänge gelesen. Das Joystick Modul besitzt jeweils ein Potentiometer für die X- und Y-Achse. Je nach Position des Joysticks wird eine andere Ausgangsspannung an unseren Arduino gegeben. Das Arduino Board wandelt die analoge Spannung zwischen 0 bis 5V in Werte von 0 bis 1023 um. Der Taster liefert permanent 2V, bei Betätigung wird er mit GND verbunden. Im seriellen Monitor lesen wir 1 im Ruhezustand und 0 beim Auslösen.

## Schaltplan



fritzing

## Anschlussplan



fritzing

Joystick	Arduino
<b>GND</b>	Ground
<b>+5V</b>	5V
<b>VRx</b>	Analoger Pin A0
<b>VRy</b>	Analoger Pin A1
<b>SW</b>	Digitaler Pin 2

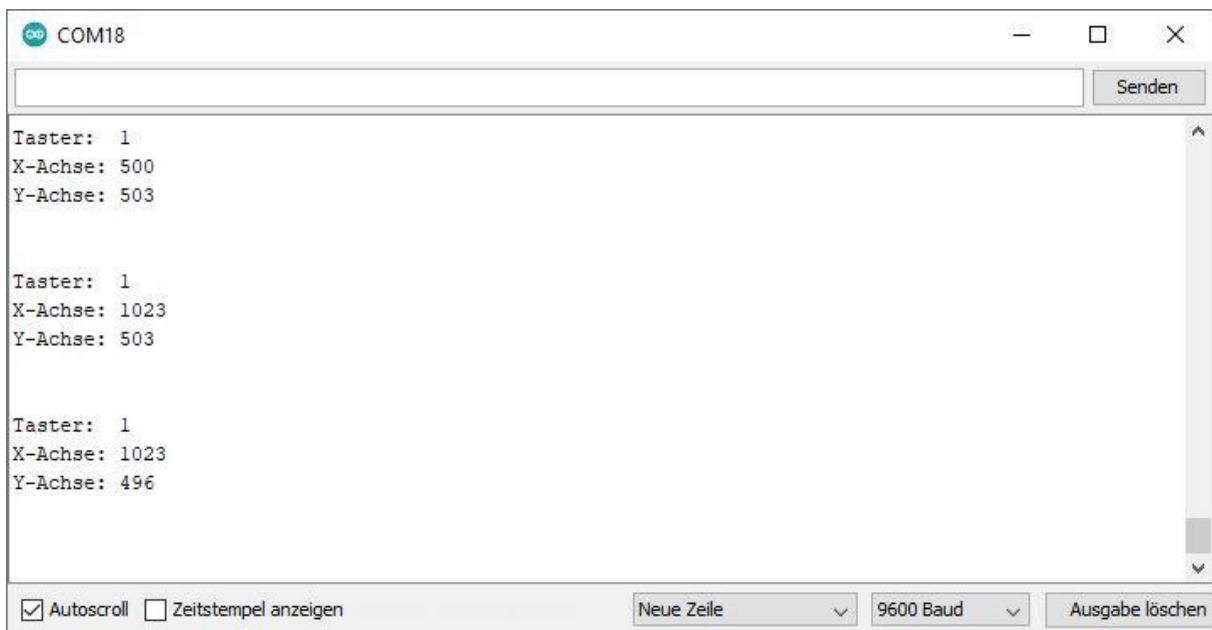
## Code für Arduino

```
const int SW_pin = 2; // Taster-Pin
const int X_pin = A0; // X-Ausgang
const int Y_pin = A1; // Y-Ausgang

void setup() {
  pinMode(SW_pin, INPUT);
  digitalWrite(SW_pin, HIGH);
}
```

```
Serial.begin(9600);  
}  
  
void loop() {  
  Serial.print("Taster: ");  
  Serial.print(digitalRead(SW_pin));  
  Serial.print("\n");  
  Serial.print("X-Achse: ");  
  Serial.print(analogRead(X_pin));  
  Serial.print("\n");  
  Serial.print("Y-Achse: ");  
  Serial.println(analogRead(Y_pin));  
  Serial.print("\n\n");  
  delay(500);  
}
```

Mit dem seriellen Monitor können Sie die aktuellen Werte auslesen. Wenn der Joystick im Mittelpunkt (Ruheposition) steht, sollte er einen Wert von ca. 512 ausgeben. Wenn Sie eine Achse bewegen, ändert sich der Wert von 0 bis 1023.



## 18. Wasserstandsensor



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Wasserstandsensor
- 3 Buchse-Stecker Kabel

### Beschreibung

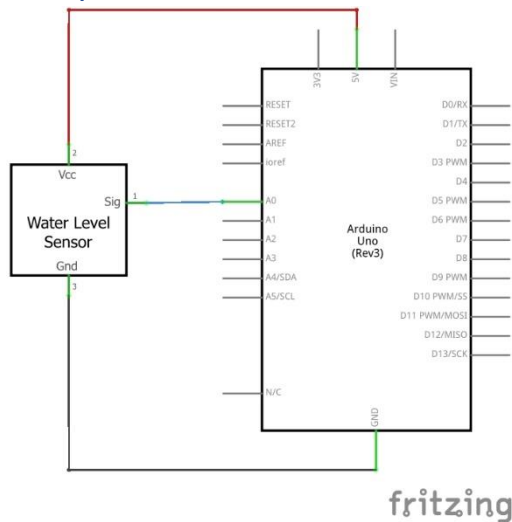
In dieser Anleitung zeigen wir Ihnen, wie Sie den Wasserstandsensor auslesen können. Dieser Sensor erkennt die Tiefe von Wasser mit Hilfe einer Verstärkerschaltung und mehreren Leiterbahnen auf der Platine. Beim Eintauchen des Moduls verändert sich der Widerstand der Leiterbahnen je nach Tiefe des Wassers. Das Signal der Wassertiefe lässt sich dann am analogen Eingang des Arduinos umwandeln und auslesen.

Mit dem Wasserstandsensor können Sie Regen und Wassertief messen oder erkennen, ob ein Leck vorhanden ist. Der Sensor besteht hauptsächlich aus einem Transistor, einem 1M Ohm Widerstand und mehreren offenen Leiterbahnen.

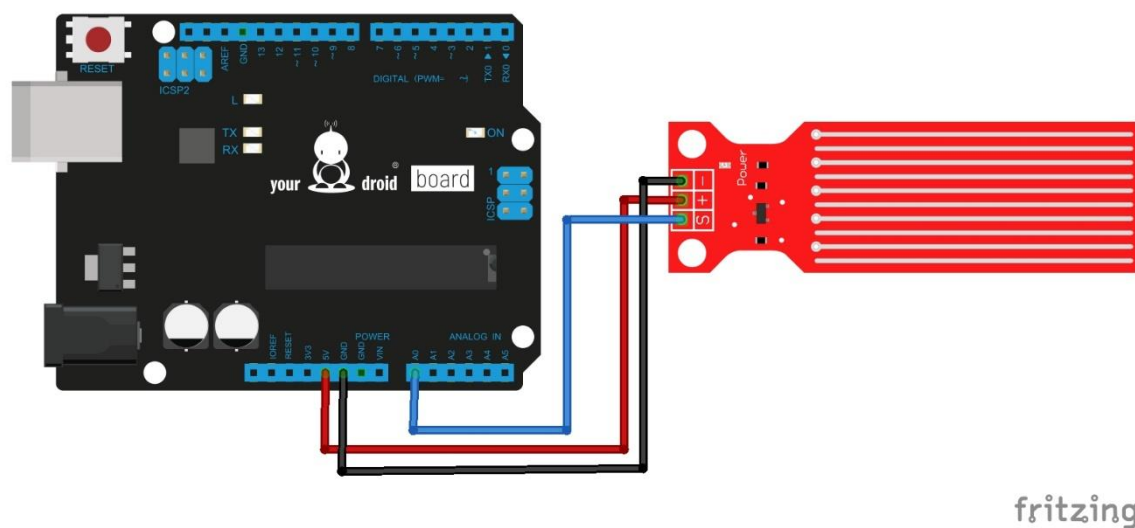
Der Sensor besitzt einen 1M Ohm Pullup-Widerstand, um den Sensor auf „High“ zu ziehen, bis Wassertropfen die Leiterbahnen des Sensors kurzschließen. Somit funktioniert der Sensor sogar am digitalen Eingang des Arduinos. Am analogen Eingang kann allerdings die Menge an Wasser zwischen den Kontakten der Leiterbahnen gemessen werden.

Der Wasserstand wird durch eine Reihe von freiliegenden parallelen Leiterbahnen ermittelt. Je nach Wassermenge verändert sich der analoge Wert. Dieser analoge Wert kann ausgelesen und in einem Programm verwendet werden, um die Funktion des Wasserstands-Sensors zu erreichen.

## Schaltplan



## Anschlussplan



Wasserstandsensor	Arduino
-	GND
+	5V
S	A0

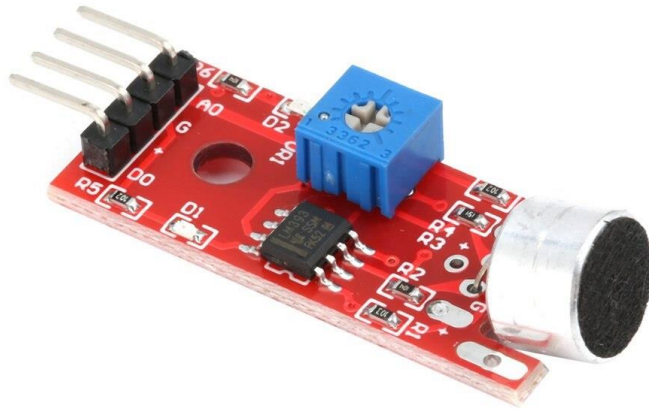
## Code für Arduino

```
int Sensor = A0;
int alterMesswert = 0; // Variable, um den vorherigen Messwert zu
vergleichen
char printBuffer[128];

void setup()
{
  Serial.begin(9600);
```

```
}  
  
void loop()  
{  
    int messwert = analogRead(Sensor); // Sensor auslesen und in der  
    Variable Messwert speichern  
  
    if(((alterMesswert>=messwert) && ((alterMesswert - messwert) > 10))  
|| ((alterMesswert<messwert) && ((messwert - alterMesswert) > 10)))  
    {  
        sprintf(printBuffer,"Messwert ist %d\n",Sensor, messwert);  
        Serial.print(printBuffer);  
        alterMesswert = messwert;  
    }  
}
```

## 19. Soundmodul



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Soundmodul
- 4 Buchse-Stecker Kabel

### Beschreibung

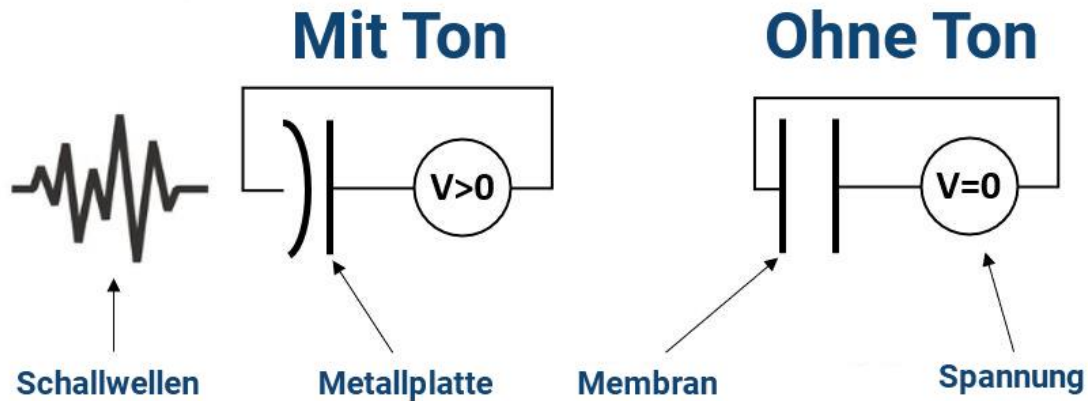
In dieser Anleitung zeigen wir Ihnen, wie Sie mit dem Soundmodul Töne erkennen können. Das Modul reagiert super auf Händeklatschen, Schnipsen oder Klopfen oder andere akustische Signale. Es besteht aus einer Schaltung mit einem Mikrofon, dessen Empfindlichkeit über das Potentiometer eingestellt werden kann. Einfach erklärt ist ein Mikrofon ein Schallwandler, der akustische Signale in elektrische Signale umwandelt. Mikrofone funktionieren im genauen Gegensatz zu Lautsprechern, die aus elektrischen Signalen Töne erzeugen.

### Funktionsweise

Mikrofone gibt es mit verschiedenen Formen, Größen und Technologien. Auf unserem Modul ist ein Kondensator-Mikrofon verbaut, mit dem wir uns etwas genauer beschäftigen.

Kondensatormikrofone werden oft in Mobiltelefonen, Laptops ect. Verbaut. Sie arbeiten wie ein Plattenkondensator, der seine Kapazität ändern kann. Im Inneren ist eine hauchdünne, elektrisch leitende Membran dicht an einer elektrisch geladenen Metallplatte angebracht.



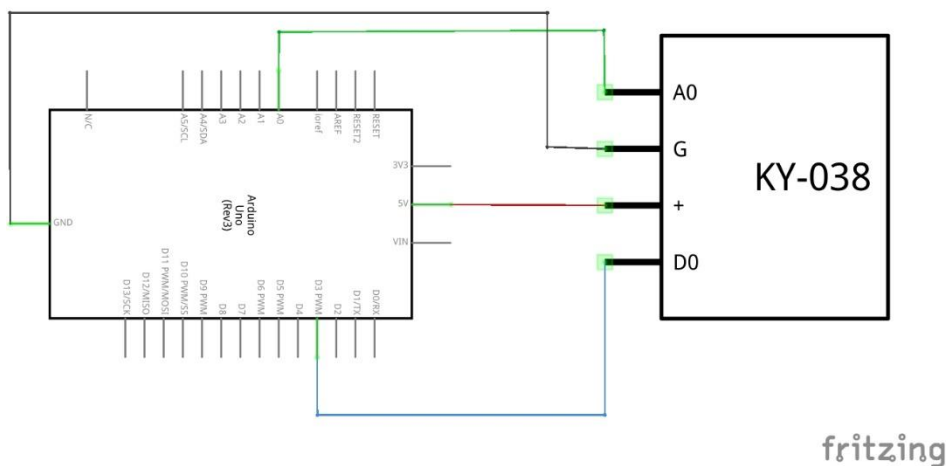


Wenn Schall auf die Membran trifft, fängt sie an zu schwingen, wodurch sich die Kapazität des Kondensators verändert. Diese Veränderung der Kapazität resultiert in einer variablen Spannung.

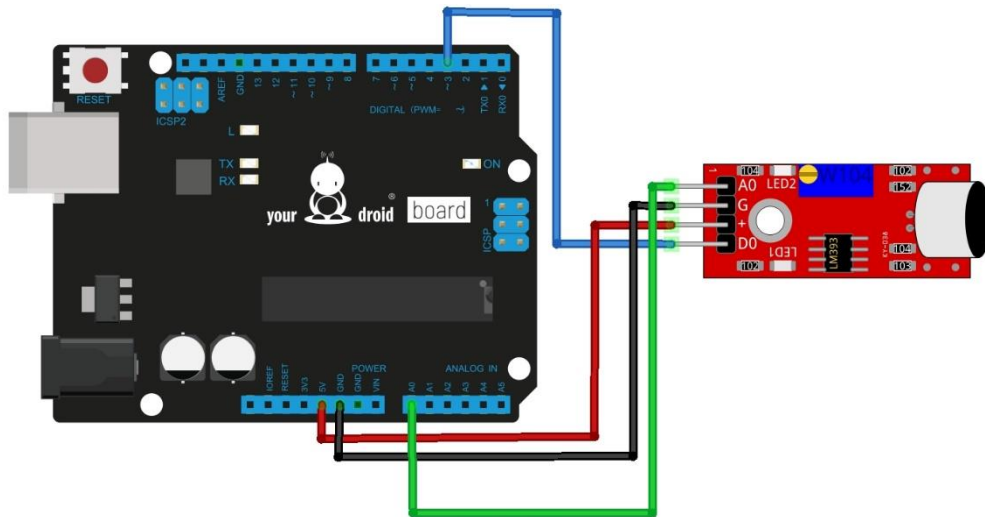
Der Pin DO ist ein Digitaler Ausgang, der „High“ ausgibt, wenn der Schwellenwert überschritten wird. Den Schwellenwert können Sie mit dem Potentiometer einstellen. Mit diesem Anschluss können Sie nur erkennen, ob eine bestimmte Lautstärke überschritten wird.

Während der digitale Ausgang nur 1 oder 0 ausgibt, können Sie mit dem analogen Ausgang AO nicht nur Geräusche erkennen, sondern auch die Lautstärke als analoges Signal verarbeiten.

## Schaltplan



## Anschlussplan



fritzing

Soundmodul	Arduino
-	GND
+	5V
AO (analoger Ausgang)	A0
DO (digitaler Ausgang)	3

## Code für Arduino

### Analoges Signal

Mit dem Code für den analogen Ausgang können Sie die Lautstärke direkt im seriellen Monitor der Arduino IDE auslesen.

```
int sensorPin = A0;    // Pin vom Soundmodul
int ledPin = 13;      // Pin der LED, 13 ist die integrierte LED des
                        // Arduino UNO
int sensorValue = 0;  // Variable um den Messwert zwischenspeichern

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  sensorValue = analogRead(sensorPin);
  digitalWrite(ledPin, HIGH);
  delay(sensorValue);
  digitalWrite(ledPin, LOW);
  delay(sensorValue);
  Serial.println(sensorValue, DEC);
}
```

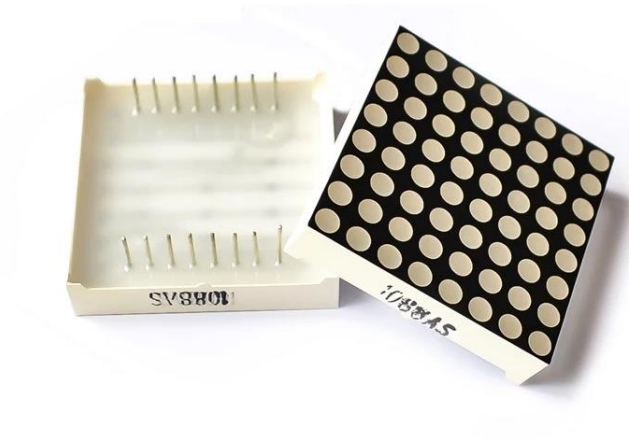
### Digitales Signal

Mit dem digitalen Ausgang können Sie erkennen, ob eine gewisse Lautstärke überschritten wird. Den Schwellenwert können Sie am Potentiometer vom Modul einstellen. Wenn ein Geräusch den Schwellenwert überschreitet, liegt am Pin 3 vom Arduino ein „High“-Signal an und die LED „L“ auf dem Arduino leuchtet auf.

```
int ledPin=13;// Pin der LED, 13 ist die integrierte ledPin des Arduino
UNO
int sensorPin=3; //define switch port
;int val;//define digital variable val
void setup()
{
  pinMode(ledPin,OUTPUT);
  pinMode(sensorPin,INPUT);
}
void loop()

{ val=digitalRead(sensorPin);
if(val==HIGH)// Wenn an Pin 3 ein Signal anliegt, soll die LED
aufleuchten
{
  digitalWrite(ledPin,HIGH);
}
else
{
  digitalWrite(ledPin,LOW); // ansonsten soll die LED aus sein
}
}
```

## 20. LED-matrix



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- 8x8 LED Matrix
- 16 Jumperkabel

### Übersicht

Die 8x8 Punkt LED-Matrix besitzt 64 einzeln ansteuerbare LEDs und eignet sich perfekt als kleines Display. Die Vorteile von der LED-Matrix sind eine sehr niedrige Stromaufnahme, hohe Helligkeit, weiter Betrachtungswinkel. Im Prinzip gibt es zwei verschiedene Arten von Punktmatrix-LED-Anzeigen – **gemeinsame Kathode** und **gemeinsame Anode**. Sie sehen fast gleich aus und unterscheiden sich von außen nur durch die Beschriftung. Der merkliche Unterschied liegt in der Ansteuerung der LEDs, ob sich alle LEDs GND oder VCC teilen. Eine Seriennummer mit **AX** besitzt eine gemeinsame Kathode, während **BX** eine gemeinsame Anode bezeichnet. In unserem Starter Kit ist eine LED-Matrix mit gemeinsamer Kathode enthalten.

Zum Anschließen der LED-Matrix werden zwei Breadboards oder Lötarbeiten benötigt.

## Funktionsweise

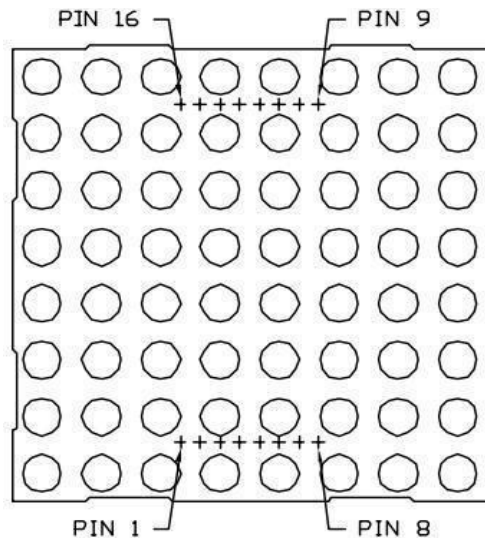


Abbildung 3 Außenansicht des Moduls

Das Display besteht aus einem 8x8 Raster mit 64 LEDs. Jede LED kreuzt sich mit einer Reihe und einer Spalte. Wenn an einer bestimmten Reihe Spannung anliegt (1) und an einer bestimmten Spalte GND (0), leuchtet die entsprechende LED auf.

Wenn sie den erste LED aufleuchten lassen wollen, müssen Sie Pin 9 „high“ und Pin 13 „low“ setzen. Um die komplette erste Reihe aufleuchten zu lassen, müssen Sie Pin 9 „high“ setzen und Pin 13,3,4,10,6,11,15 und 16 auf „low“.

Wenn Sie komplette erste Spalte aufleuchten lassen wollen, müssen Sie Pin 13 „low“ setzen und Pins 9, 14, 8, 12, 1, 7, 2 und 5 auf „high“.

Innere Ansicht des Moduls

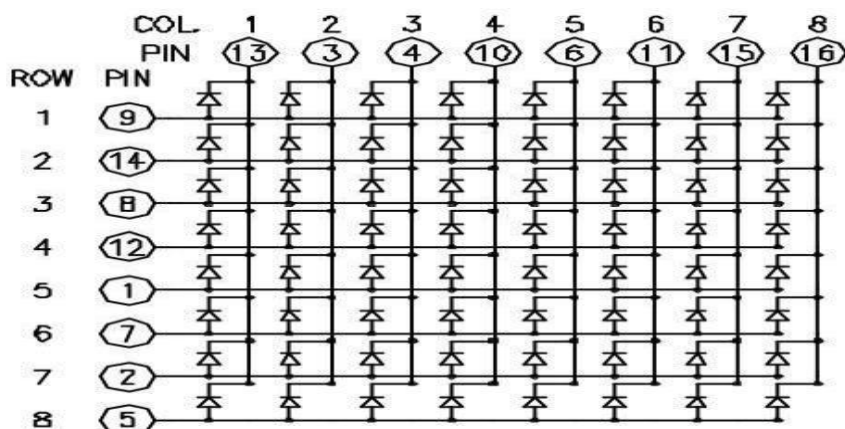
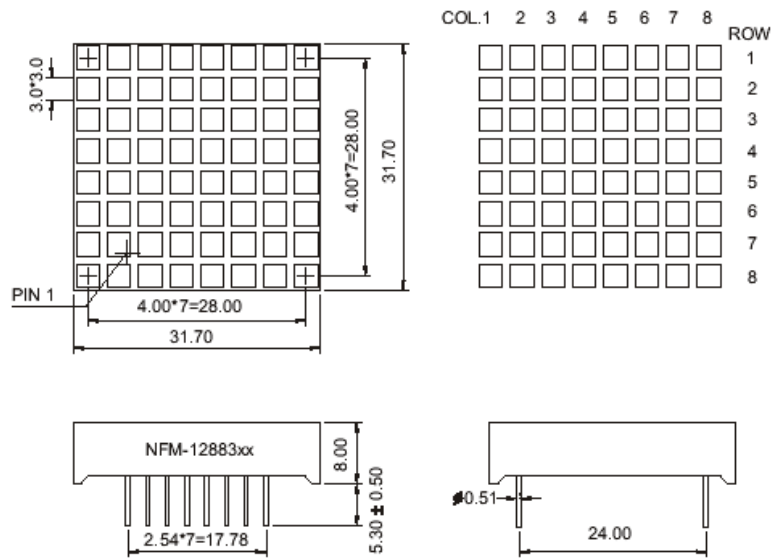
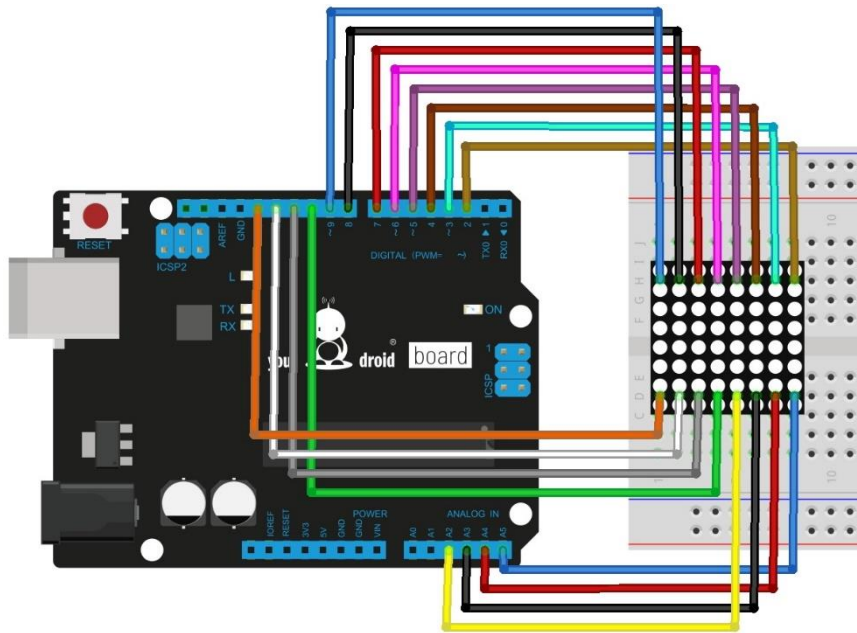


Abbildung 4 Innenansicht des Moduls

## Anschlussplan



Matrix Pin	Reihe (Row)	Spalte (Column)	Arduino Pin
1	5	-	13
2	7	-	12
3	-	2	11
4	-	3	10
5	8	-	A2
6	-	5	A3
7	6	-	A4
8	3	-	A5
9	1	-	2
10	-	4	3
11	-	6	4
12	4	-	5
13	-	1	6
14	2	-	7
15	-	7	8
16	-	8	9



## Code für Arduino

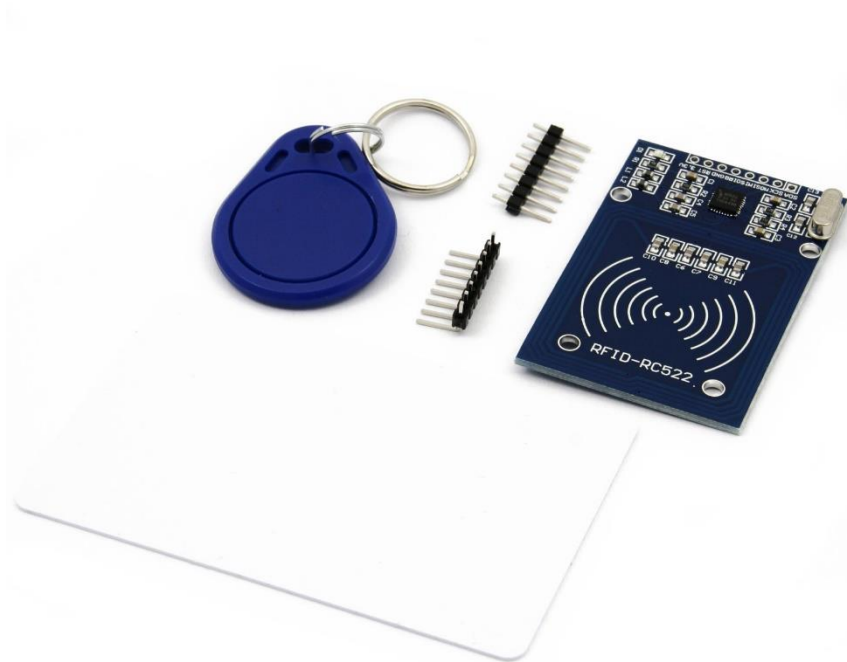
### Test-code

Um ein Gefühl für die Ansteuerung der Matrix zu bekommen, versuchen wir zunächst die erste LED zum Blinken zu bringen. Hierfür müssen wir die Pins 9 und 13 der LED-Matrix ansprechen (siehe Tabelle weiter oben). Bei unserer Verkabelung entspricht das Pin 3 und Pin 11 an unserem Arduino.

### Animiertes LED-herz

Beim zweiten Code werden alle Anschlüsse benötigt, wenn alles geklappt hat blinken die LEDs abwechselnd als kleines und großes Herz auf der Anzeige.

## 21. MFRC522 RFID Modul



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- MFRC522 RFID Modul + Transponder
- Buchste-Stecker Jumperkabel
- 5mm LED
- 220 Ohm Widerstand
- Breadboard

### Übersicht

RIFD (engl. radio-frequency identification) ist eine Technologie die elektromagnetische Wellen nutzt um Daten über kurze Distanzen zu übertragen. RFID ist besonders nützlich zur Identifikation von Personen und Tieren, bargeldlosen Bezahlung oder als Zutrittskontrolle.

Die Übertragung der Daten erfolgt über ein Lesegerät (Reader) welches ein elektromagnetisches Wechselfeld erzeugt auf das der RFID-Transponder (RFID-Tag/Karte) reagiert. Die vom RFID-Tag aufgenommene Hochfrequenz dient während der Kommunikation als Stromversorgung, wodurch die RFID-Transponder keine Batterien benötigen.

Die RFID-Transponder bestehen aus einem kleinen Chip und einem aufgewickelten Kupferdraht der als Antenne dient. Sie sind in verschiedenen Varianten erhältlich, zb. als Schlüsselanhänger oder im Kreditkartenformat.

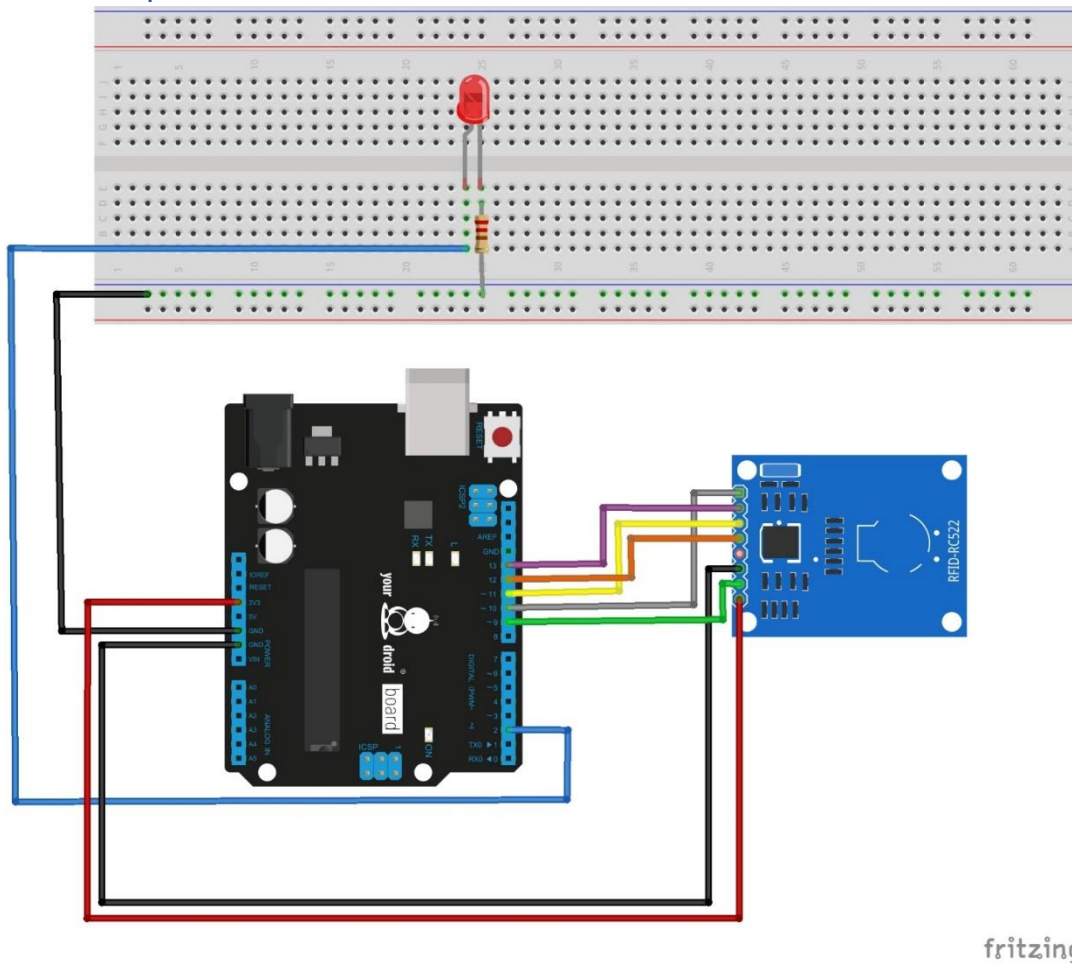


## MFRC522 RFID-Modul

Das MFRC522 Modul ist ein RFID Reader/Writer für kontaktlose Kommunikation mit **13.56Mhz**. Die interne Antenne ist ausgelegt um ISO/IEC 14443A/MIFARE Transponder zu lesen und beschreiben. Die Schaltung auf dem Modul demoduliert und entschlüsselt die empfangenen Daten und ist in der Lage die Daten auf Parität und Fehler zu prüfen.

Beim MFRC522 Modul sind 2 verschiedene Stiftleisten enthalten, zur Verwendung auf einem Breadboard sollte man die **gewinkelte Stiftleiste** anlöten.

## Anschlussplan



fritzing

RFID Modul	Arduino
SDA	10
SCK	13
MOSI	11
MISO	12
GND	GND
RST	9
3.3V	3.3V
IRQ	Nicht angeschlossen

Achtung, das Modul muss mit 3,3V betrieben werden! Bei Verwendung mit einem 5V Board wird ein [Pegelwandler](#) benötigt!

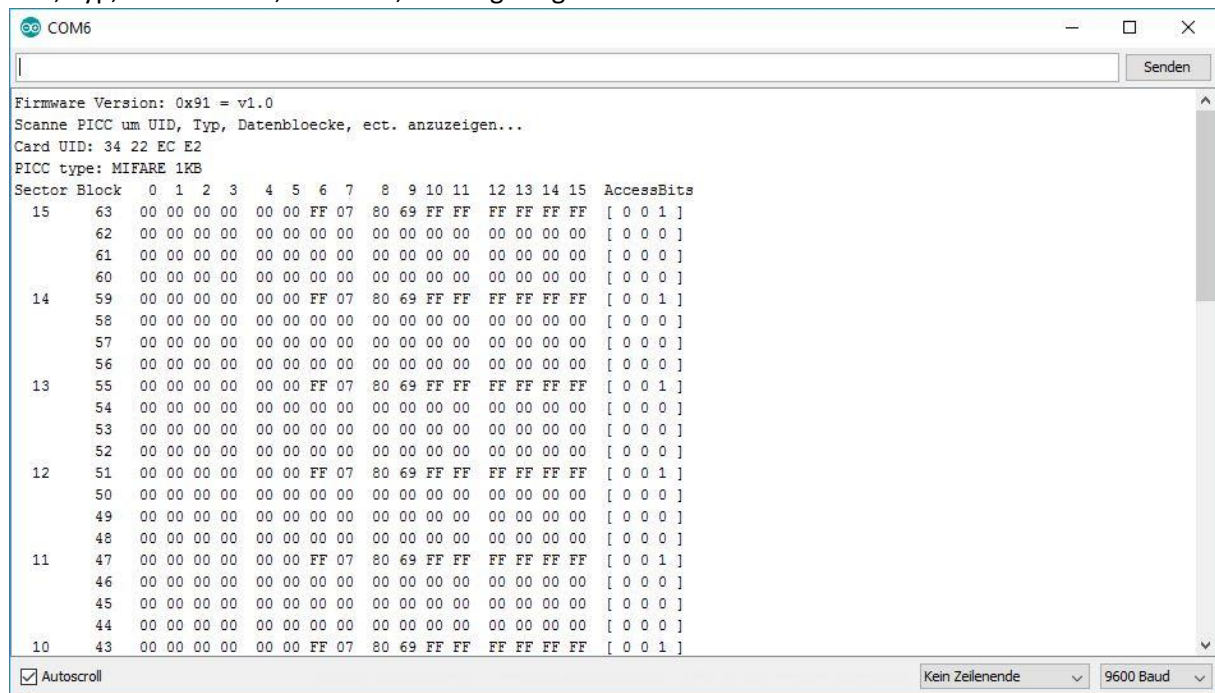
## Beispiel 1 RFID Tags auslesen

Zur Programmierung des Arduinos empfiehlt sich diese RFID-Bibliothek, welche auch in der offiziellen Arduino IDE zu finden ist.

Die Bibliothek enthält gute Beispiel-Sketches für grundlegende RFID-Funktionen.

Daher laden wir den „**DumpInfo**“-Beispielsketch auf unser Arduino um die einzigartige Identifikationsnummer (UID) des RFID-Tags herauszufinden. Den Sketch gibt es nochmal im „Code“-Ordner mit deutschen Kommentaren versehen, um die einzelnen Zeilen zu verstehen.

Anschließend öffnen wir den seriellen Monitor und halten ein RFID-Transponder an den RFID-Reader bis die Daten übertragen wurden. Wurde der Transponder erfolgreich ausgelesen, bekommen wir UID, Typ, Datenblöcke, Sektoren, ect. angezeigt.



```
COM6
Firmware Version: 0x91 = v1.0
Scanne PICC um UID, Typ, Datenblöcke, ect. anzuzeigen...
Card UID: 34 22 EC E2
PICC type: MIFARE 1KB
Sector Block 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 AccessBits
15 63 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
62 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
14 59 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
13 55 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
54 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
53 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
52 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
12 51 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
11 47 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
45 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
44 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]
10 43 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]
Autoscroll Kein Zeilenende 9600 Baud
```

## Beispiel 2: Eine LED aufleuchten lassen!

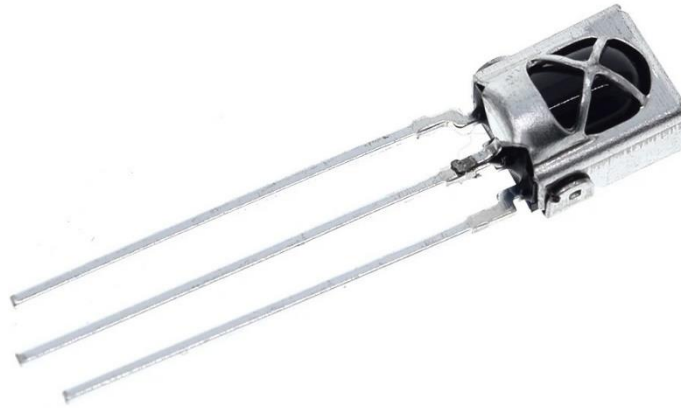
Nachdem wir nun die UID unseres RFID-Transponders kennen, können wir diese nutzen, um verschiedenste Funktionen zu programmieren. So zeigen wir im **RFID\_LED Beispielcode** wie man eine LED aufleuchten lässt sobald der richtige RFID-Tag erkannt wird. Diesen Sketch kann man auch einfach verändern, um z. B. ein Relais anstelle einer LED zu schalten.

Um den Sketch mit Eurem RFID-Tag verwenden zu können muss lediglich eine Zeile angepasst werden:

```
if (rfid == "34 22 EC E2") // Wenn die richtige Karte erkannt wurde...
```

Hier wird einfach die vorher ausgelesene Card UID eingetragen.

## 22. Infrarot-Fernbedienung



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Breadboard
- 11 Jumperkabel
- 1x IR Fernbedienung
- 1x IR Empfänger
- 6x 220 Ohm Widerstand
- 6x LED

### Übersicht

In Dieser Anleitung zeigen wir Ihnen, wie sie den Infrarotempfänger und die Infrarotfernbedienung mit Ihrem Arduino UNO kompatiblen Board verwenden.

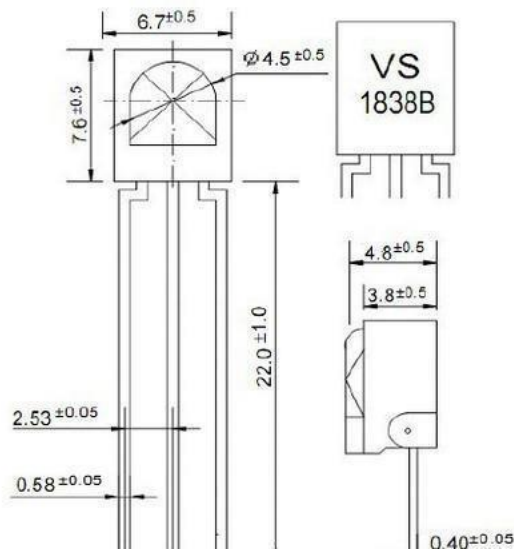
IR-Fernbedienungen senden binäre Impulse als Signal im unsichtbaren Infrarotbereich aus. Um Störsignale von anderen Quellen zu vermeiden, werden die Signale vormoduliert und erst dann mit Hilfe einer IR-LED in einer bestimmten Frequenz gesendet.

Der IR-Empfänger besitzt in der Regel eine IR-Fotodiode, die Signale auf einer bestimmten Frequenz empfängt und Störungen von anderen Lichtquellen filtert. Hinter der IR-Fotodiode befindet sich dann meistens ein Signalverstärker und Demodulator, um das Signal weiterverarbeiten zu können.

### Funktionsweise

Der IR-Empfänger wandelt das Lichtsignal, welches von der IR-LED gesendet wird, in ein schwaches, elektronisches Signal um. Dieses Signal wird dann durch einen IC verstärkt, automatisch gefiltert,

geglättet und demoduliert, bis das ursprüngliche Signal der Fernbedienung wiederhergestellt ist. Dieses Signal wird dann an den Output-Pin des IR-Empfängers gesendet.



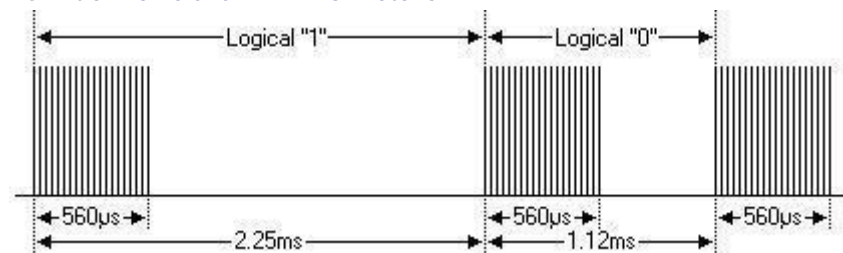
## Verschlüsselung mit NEC-Protokoll

Wenn Sie die Signale einer IR-Fernbedienung auslesen wollen, müssen Sie zuerst wissen, wie sie verschlüsselt sind. Die Verschlüsselung in unserem Beispiel ist das NEC-Protokoll.

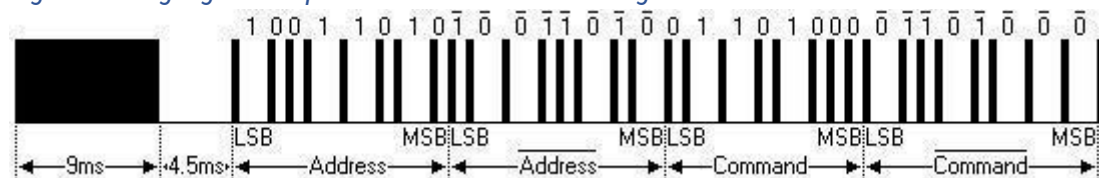
### Technische Daten des NEC-Protokolls

- 8-bit Adressen und 8-bit Befehlslängen.
- Adresse und Befehl werden zur Verlässlichkeit immer wiederholt.
- Puls Distance Modulation (PDM)
- Frequenz von 38 Khz
- Bit time von 1.125 ms oder 2.25 ms

### Definition von 0 und 1 im NEC-Protokoll



### Signalübertragung bei Knopfdruck auf der Fernbedienung

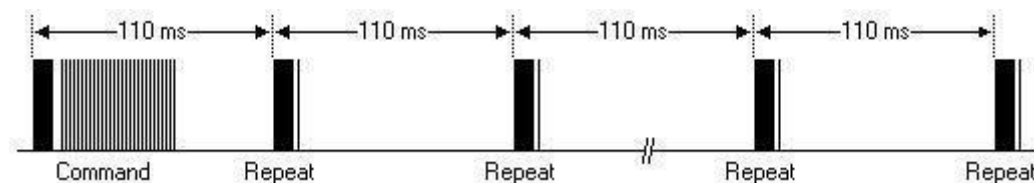


Diese Abbildung zeigt eine typische Impulsfolge des NEC-Protokolls. Bei diesem Protokoll wird das LSB (Least Significant Bit) zuerst übertragen. In diesem Fall wird die Adresse \$59 und der Befehl \$16 übertragen. Eine Nachricht wird durch einen 9 ms AGC-Burst eingeleitet, der bei den früheren IR-

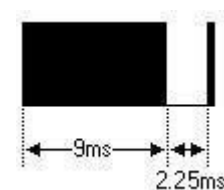
Empfängern zur Einstellung der Verstärkung verwendet wurde. AGC steht kurz für engl. „automatic gain control“ und dient dazu, den Ausgangspegel eines Verstärkers konstant zu halten.

Auf diesen AGC-Burst folgt ein 4,5 ms langes Leerzeichen, auf das die Adresse und der Befehl folgen. Adresse und Befehl werden zwei Mal übertragen, um die Fehlerquote zu verringern. Beim zweiten Übertragen sind alle Bits invertiert und können zur Überprüfung der empfangenen Nachricht verwendet werden. Die Gesamtübertragungszeit ist konstant, da jedes Bit mit seiner invertierten Länge wiederholt wird. Wenn Sie an diese Zuverlässigkeit nicht benötigen, können Sie die invertierten Werte auch ignorieren, oder die Adresse und den Befehl auf jeweils 16 Bit erweitern!

Ein Impuls wird gesendet, wenn eine Taste gedrückt und nach einer bestimmten Zeit losgelassen wird.

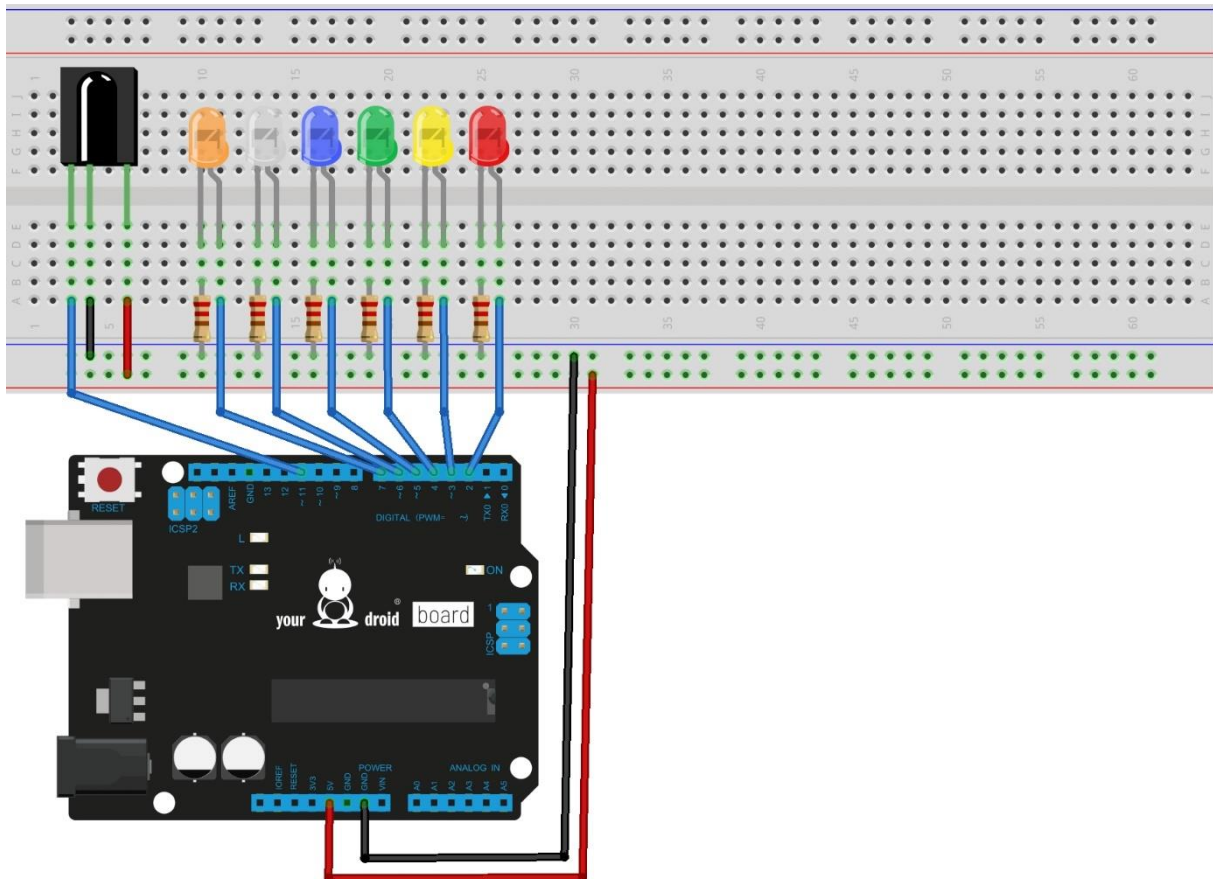


Ein Befehl wird nur einmal übertragen, auch wenn die Taste der Fernbedienung gedrückt bleibt. Alle 110ms wird ein Wiederholungscode gesendet, solange die Taste gedrückt bleibt. Dieser Wiederholungscode ist einfach ein 9ms AGC-Burst, gefolgt von einer 2,25ms Pause und einem 560µs Burst.



Wenn der Impuls im Empfänger eintrifft, findet eine Dekodierung, Signalverstärkung und Wellenglättung statt. Sie müssen sicherstellen, dass der Ausgangspegel genau das Gegenteil von dem des sendenden Signals ist. Das heißt, wenn kein Infrarotsignal vorhanden ist, ist der Ausgangspegel hoch; Wenn ein Infrarotsignal vorhanden ist, ist der Ausgangspegel low. Sie können das Signal des Empfängers mit einem Oszilloskop untersuchen, um das Signal besser zu verstehen.

## Anschlussplan



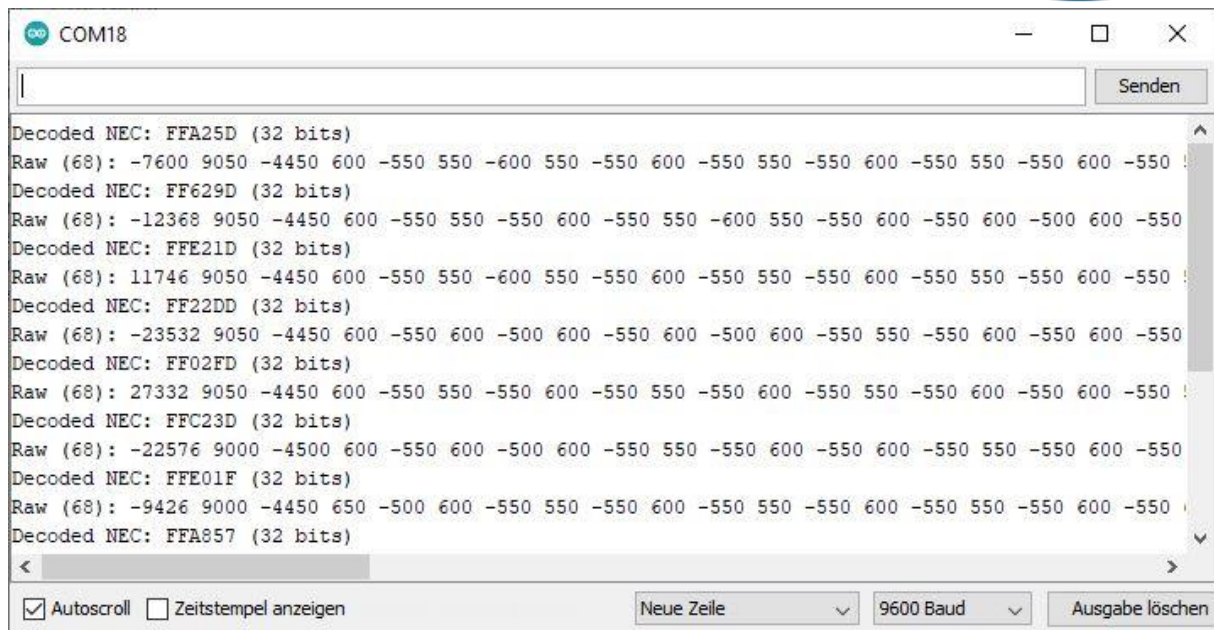
fritzing

Den VOUT-Pin vom IR-Empfänger schließen wir auf Pin 11 an. Die LEDs mit den Widerständen kommen an die Pins 2, 3, 4, 5, 6, 7. Die Spannungsversorgung erfolgt über den BUS des Breadboards, verbunden mit GND und 5V vom Arduino.

## Code

Für den Code wird die **IRremote Bibliothek** verwendet. Das Programm dekodiert das verschlüsselte Signal der IR-Fernbedienung. So können Sie im seriellen Monitor der Arduino IDE die Signale der Fernbedienung auslesen. Mit diesen Signalen können wir dann unsere Anwendung steuern, in diesem Beispiel die 6 LEDs. Laden Sie den Sketch hoch und versuchen Sie die LEDs an- und auszuschalten.





Die Codes können je nach Fernbedienung und Charge variieren. Wenn Sie z. B. die Tastenbelegung zum Einschalten der ersten LED ändern möchten, müssen Sie folgende Zeile anpassen:

```
long on1 = 0x00FFA25D;
```

Diese Zeile passen wir an, um die erste LED einzuschalten. Wie wir im seriellen Monitor erkennen können, entspricht **FFA25D** unserer ersten Taste der Fernbedienung.

Nun können Sie den angepassten Sketch erneut hochladen und die LEDs mit Hilfe der Taste anschalten. So können Sie nach der Reihe alle LEDs den Tasten zuweisen.

## 23. Acht LEDs mit 74HC595 ansteuern



### Benötigte Komponenten

- Arduino UNO R3 kompatibles Board
- Breadboard
- 14 Jumperkabel
- 74HC595 IC
- 8x 220 Ohm Widerstand
- 8x LED

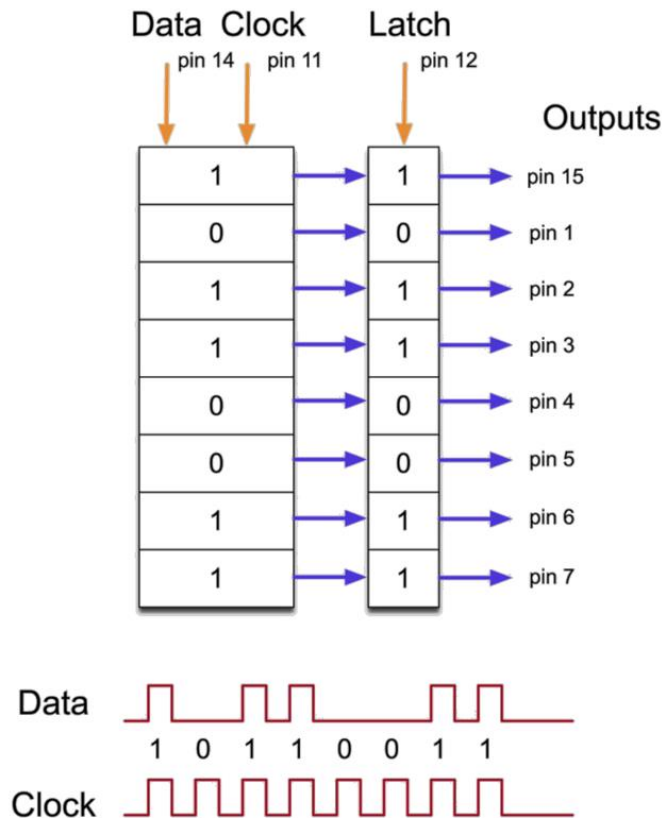
### Übersicht

In dieser Anleitung zeigen wir Ihnen, wie Sie mit Hilfe des 74HC595 acht LEDs mit weniger Pins steuern können. Während Sie die LEDs auch einzeln an den Arduino UNO anschließen könnten, würden Ihnen schnell die Pins für andere Komponenten wie Sensoren und Motoren ausgehen. Um dieses Problem zu umgehen, nutzen wir den 74HC595 Chip, einen Seriell-zu-Parallel-Logik-Wandler. Dieser Chip besitzt 8 Ausgänge, die Sie mit nur 3 Eingängen Bit für Bit ansteuern können. Das Ansteuern der LEDs mit dem 74HC595 IC geschieht etwas langsamer, ist aber mit über 500.000 Befehlen pro Sekunde statt 8.000.000 immer noch sehr schnell – schneller als das menschliche Auge eine Veränderung erkennen kann.

### Funktionsweise

Der 74HC595 Chip arbeitet mit einem Schieberegister (shift register), welches 8 Speicherplätze mit entweder 0 oder 1 füllen kann. Um jeden dieser Werte ein- oder auszuschalten, werden die Daten über die Pins "Data" und "Clock" des Chips eingespeist.

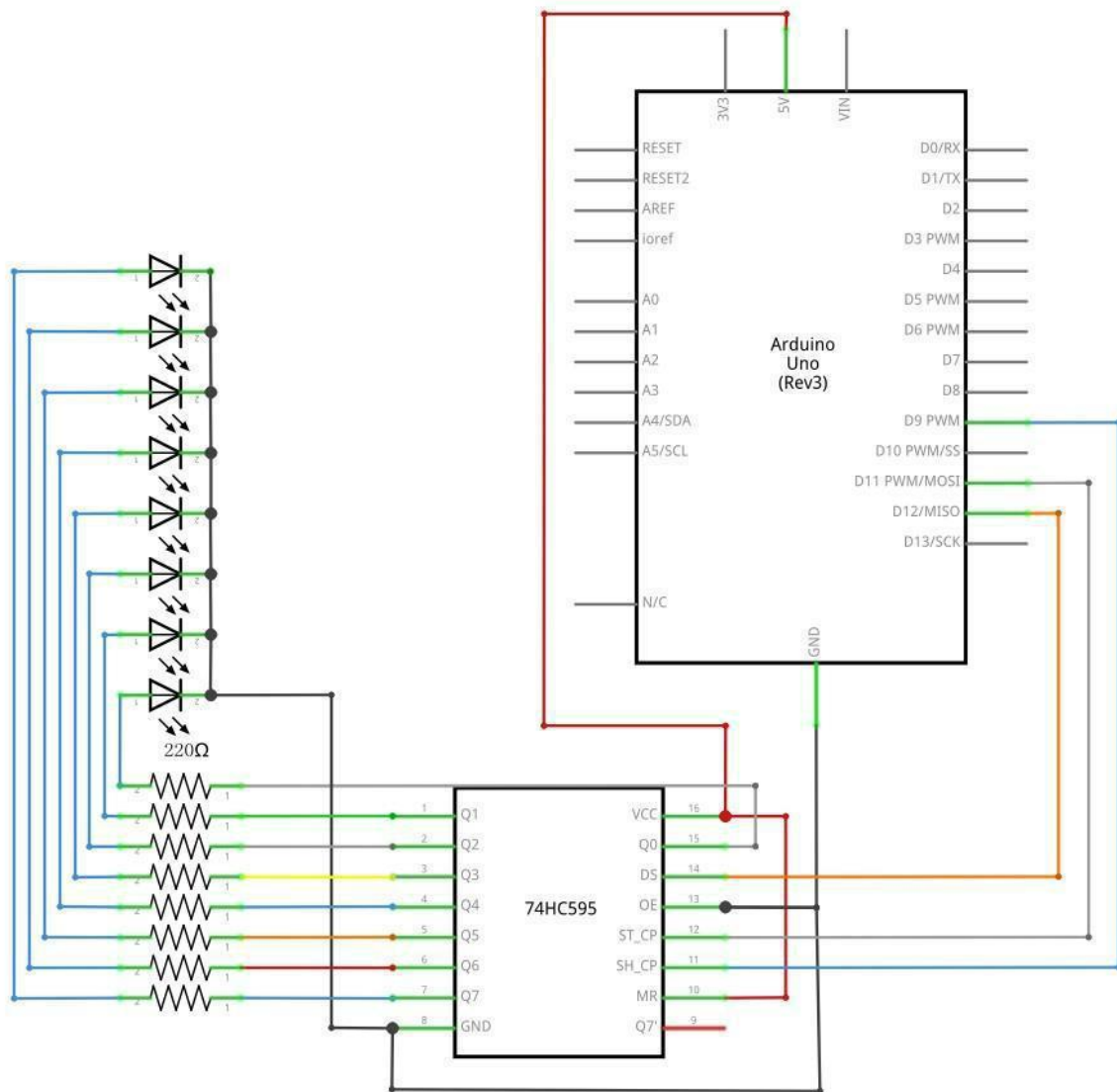




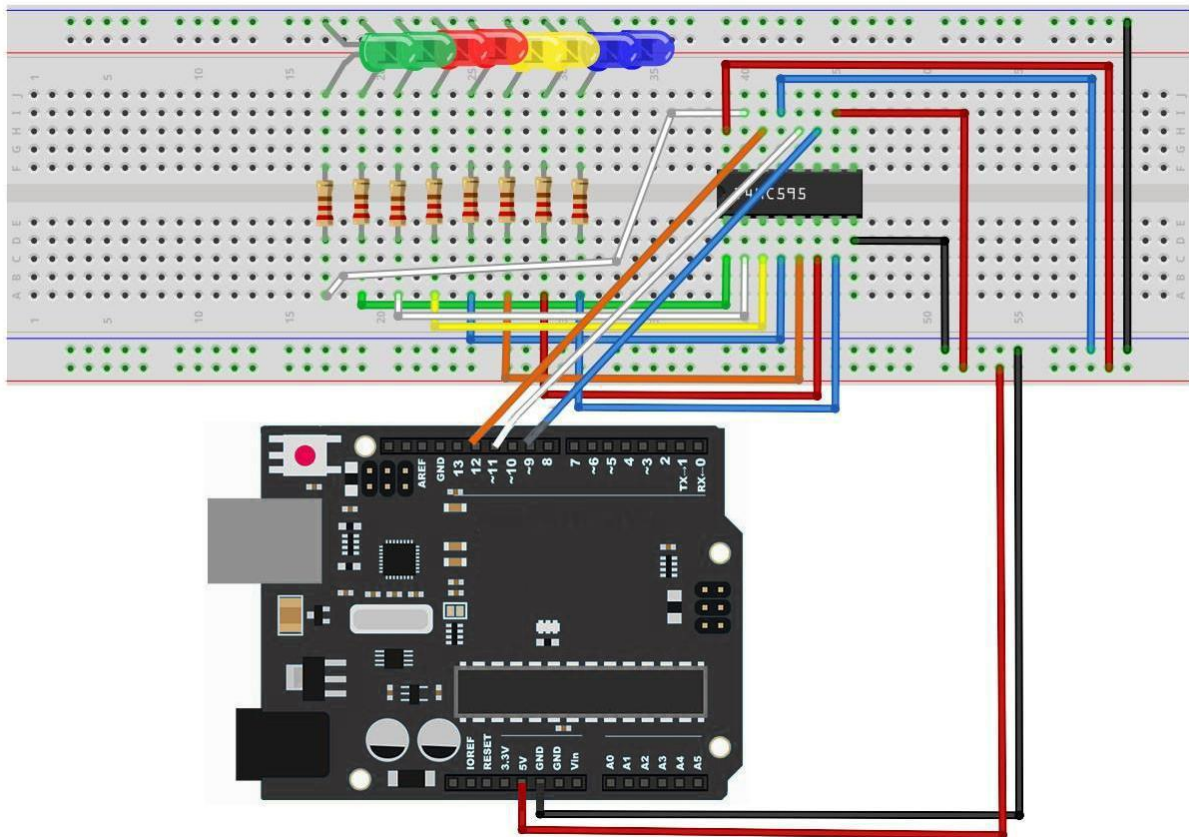
Der Clock-Pin muss acht Impulse empfangen. Bei jedem Impuls wird, wenn der Daten-Pin „high“ ist, eine 1 in das Schieberegister geschoben, andernfalls eine 0 („low“). Wenn alle acht Impulse empfangen wurden, kopiert das Aktivieren des 'Latch'-Pins diese acht Werte in das Latch-Register. Dies ist notwendig, da sonst die falschen LEDs flackern würden, während die Daten in das Schieberegister geladen werden.

Der Chip verfügt auch über einen Output-Enable-Pin (OE), mit dem die Ausgänge auf einmal aktiviert oder deaktiviert werden können. Man könnte diesen Pin mit einem PWM-fähigen UNO-Pin verbinden und 'analogWrite' verwenden, um die Helligkeit der LEDs zu steuern. Dieser Pin ist active low, also verbinden wir ihn mit GND.

## Schaltplan



## Anschlussplan



Da wir acht LEDs und acht Widerstände anschließen müssen, gibt es tatsächlich eine ganze Reihe von Verbindungen herzustellen.

Es ist am einfachsten, den 74HC595-Chip zuerst anzuschließen, da so ziemlich alles andere mit ihm verbunden ist. Setzen Sie ihn so ein, dass die kleine U-förmige Kerbe oben auf dem Breadboard liegt. Der Pin 1 des Chips befindet sich links von dieser Ausparung.

Digital 12 von der UNO geht an Pin #14 des Schieberegisters

Digital 11 von der UNO geht an Pin #12 des Schieberegisters

Digital 9 vom UNO geht an Pin #11 des Schieberegisters

Alle Ausgänge des ICs bis auf einen befinden sich auf der linken Seite des Chips. Der Einfachheit halber befinden sich daher auch die LEDs dort.

Setzen Sie anschließend die Widerstände ein. Sie müssen darauf achten, dass sich die Leitungen der Widerstände nicht berühren. Das sollten Sie noch einmal überprüfen, bevor Sie den Strom an den Arduino UNO anschließen. Wenn Sie Schwierigkeiten haben, die Widerstände so anzuordnen, dass sich ihre Leitungen nicht berühren, können Sie die Drähte kürzen. Dadurch liegen Sie näher an der Oberfläche vom Breadboard.

Als Nächstes platzieren Sie die LEDs auf dem Breadboard. Die langen Beinchen der LEDs (positiv) müssen alle in Richtung des Chips zeigen, unabhängig davon auf welcher Seite des Breadboards sie sich befinden.

Bringen Sie die Jumperkabel wie oben gezeigt an. Vergessen Sie nicht die Brücke, die von Pin 8 des ICs zur GND-Spalte des Breadboards führt.

Laden Sie den Sketch hoch und probieren Sie ihn aus. Jede LED sollte der Reihe nach aufleuchten, bis alle LEDs an sind. Anschließend sollten sie alle wieder ausgehen und den Zyklus wiederholen.

## Code

```
int tDelay = 100;
int latchPin = 11;      // (11) ST_CP [RCK] on 74HC595
int clockPin = 9;       // (9) SH_CP [SCK] on 74HC595
int dataPin = 12;       // (12) DS [S1] on 74HC595

byte leds = 0;

void updateShiftRegister()
{
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, leds);
    digitalWrite(latchPin, HIGH);
}

void setup()
{
    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
}

void loop()
{
    leds = 0;
    updateShiftRegister();
    delay(tDelay);
    for (int i = 0; i < 8; i++)
    {
        bitSet(leds, i);
        updateShiftRegister();
        delay(tDelay);
    }
}
```

Als erstes definieren wir die drei Pins, die wir verwenden werden. Dies sind die digitalen UNO-Ausgänge, die mit den Latch-, Clock- und Daten-Pins des 74HC595 verbunden werden.

```
int latchPin = 11;
```

```
int clockPin = 9;
```

```
int dataPin = 12;
```

Als nächstes wird eine Variable namens "leds" definiert. Diese wird verwendet, um das Muster der LEDs zu speichern, die gerade ein- oder ausgeschaltet sind. Daten des Typs "Byte" stellen Zahlen mit acht Bits dar. Jedes Bit kann entweder ein- oder ausgeschaltet sein, so dass diese Variable perfekt geeignet ist, um folgende Werte zu speichern welche unserer acht LEDs ein- oder ausgeschaltet sind.

```
byte leds = 0;
```

Die Setup-Funktion deklariert die drei Pins als digitale Ausgänge.

```
void setup()
{
  pinMode(latchPin,    OUTPUT);
  pinMode(dataPin,     OUTPUT);
  pinMode(clockPin, OUTPUT);
}
```

Die Funktion "loop" schaltet zunächst alle LEDs aus, indem sie der Variable "leds" den Wert 0 gibt. Dann ruft sie 'updateShiftRegister' auf, dass das 'leds'-Muster an das Schieberegister sendet, so dass alle LEDs ausgeschaltet werden. Wie 'updateShiftRegister' funktioniert, wird im Anschluss erklärt.

Die Schleifenfunktion hält eine halbe Sekunde lang an und beginnt dann, mit Hilfe der 'for'-Schleife und der Variablen 'i' von 0 bis 7 zu zählen. Jedes Mal wird die Arduino-Funktion 'bitSet' verwendet, um das Bit zu setzen, das die LED in der Variablen 'leds' steuert.

Anschließend wird die Funktion "updateShiftRegister" aufgerufen, damit die LEDs aktualisiert werden, um den Wert in der Variablen "leds" wiederzugeben. Es gibt dann eine halbe Sekunde Verzögerung, bevor 'i' inkrementiert wird und die nächste LED aufleuchtet.

```
loop()
{
  leds = 0;
  updateShiftRegister();
  delay(500);
  for (int i = 0; i < 8; i++)
  {
    bitSet(leds, i);
    updateShiftRegister();
    delay(500);
  }
}
```

Die Funktion 'updateShiftRegister' setzt zunächst den latchPin auf low und ruft dann die UNO-Funktion 'shiftOut' auf, bevor sie den 'latchPin' wieder auf high setzt.

Diese Funktion benötigt vier Parameter, die ersten beiden sind die Pins, die für Data bzw. Clock verwendet werden sollen.

Der dritte Parameter gibt an, an welchem Ende der Daten man beginnen möchte. Wir beginnen mit dem rechten Bit, das als 'Least Significant Bit' (LSB) bezeichnet wird.

Der letzte Parameter sind die eigentlichen Daten, die in das Schieberegister geschoben werden sollen, in diesem Fall also 'leds'.

```
void updateShiftRegister()
{
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, LSBFIRST, leds);
digitalWrite(latchPin, HIGH);
}
```

Wenn Sie eine der LEDs nicht ein-, sondern ausschalten wollten, würden Sie eine ähnliche Arduino-Funktion (`bitClear`) mit der Variablen "leds" aufrufen. Dadurch wird das Bit von "leds" auf 0 gesetzt und Sie müssten dann nur noch einen Aufruf von "updateShiftRegister" folgen lassen, um die tatsächlichen LEDs zu aktualisieren.

## Ergebnis

