

Wolfgang Wolff

BOARD MODEL  
UNO R3

OPEN-SOURCE ELECTRONICS®  
PROTOTYPING PLATFORM

MADE IN ITALY

Einführung für Schülerinnen und Schüler in  
[WWW.ARDUINO.CC](http://WWW.ARDUINO.CC)

# Mikrocontroller mit dem Arduino

NwT Schülerheft



Marie-Curie Gymnasium



If you get, give. If you learn, teach

---

Maya Angelou (April 4, 1928 – May 28, 2014), African-American author, poet, dancer, actress and singer

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

---

## Impressum

1. Mai 2019

Version 0.5

Wolfgang Wolff

Marie-Curie Gymnasium Kirchzarten

wolff@mcg-kirchzarten.de



Dieses Werk bzw. Inhalt steht unter einer Creative Commons Namensnennung-Nicht-kommerziell-Weitergabe unter gleichen Bedingungen 3.0 Unported Lizenz.

# Vorwort

Irrend lernt man.

---

Johann Wolfgang von Goethe (1749 - 1832),  
deutscher Dichter der Klassik, Naturwissenschaftler  
und Staatsmann

Dieses Skript ist nicht fehlerlos. Fehler beschreiben meiner Meinung nach einen Prozess. Wenn man etwas neues lernen möchte, dann sind sie unumgänglich! Deshalb sollte man auch keine Angst davor haben Fehler zu begehen. Solange man bereit ist aus ihnen zu lernen.

In diesem Skript gibt es zwei Arten von Fehlern, einfache Schreibfehler und inhaltliche Fehler. Schreibfehler sind ärgerlich, aber solang der Sinn des Textes nicht verändert wird nicht tragisch. Bei inhaltlichen Fehlern sieht es ganz anders aus. Wenn in einem Beispiel-Sketch ein Fehler ist, dann wird dieser Sketch nicht mehr funktionieren! Wenn in einem Schaltplan zwei PINs vertauscht sind, kann sogar ein elektronisches Bauteil zerstört werden. Solltest du einen Fehler (egal welcher) finden, dann würde ich mich freuen, wenn du ihn mir mitteilen würdest.

Du wirst dich aber auch mit deinen eigenen Fehlern befassen müssen! Wenn du wirklich lernen möchtest, wie ein Mikrocontroller funktioniert und wie du ihn programmierst (wovon ich ausgehe), dann ist es sehr wichtig, dass du alle Aufgaben selber löst! Du wirst Fehler machen, deine Schaltungen werden nicht funktionieren, andere Gruppen werden viel schneller sein! Dies sollte dich nicht entmutigen. Wenn du ehrlich die Aufgaben löst wirst du feststellen, dass ab einem gewissen Punkt du auf einmal viel schneller wirst! Eben wann, wenn du kapiert hast, wie die ganze Sache funktioniert.

**Was solltest du tun, wenn du nicht mehr weiter kommst?** Wenn etwas nicht funktioniert, dann Frage deine Mitschüler, deinen Lehrer. Versuche deinen Fehler zu finden und zu verstehen. Es macht keinen Sinn einfach weiterzumachen oder einen richtigen Sketch von der Nachbargruppe zu benutzen! Wichtig ist, dass du alle deine selbst

---

geschriebenen Sketche gut dokumentierst. So schafft du dir selber eine gute Basis auf der du weitere Aufgaben lösen kannst.

# Inhaltsverzeichnis

<b>1. Einführung</b>	<b>10</b>
1.1. Das Arduino Projekt . . . . .	11
1.2. Das Arduino Uno Board . . . . .	16
1.3. Software: die Arduino-IDE . . . . .	17
1.4. Blink, dein erster Arduino Sketch . . . . .	19
1.5. Der Arduino Sketch . . . . .	25
1.6. Mit dem PC kommunizieren . . . . .	27
1.7. <b>C Sprachelemente: Variablen</b> . . . . .	30
<b>2. Sensoren</b>	<b>36</b>
2.1. Analoger Sensor: Der Temperatur-Sensor LM35 und LM36 . . . . .	37
2.2. <b>C Sprachelemente: Unterprogramme</b> . . . . .	41
2.3. Widerstände und Spannungsteiler . . . . .	46
2.4. Lichtsensoren . . . . .	48
2.5. <b>C Sprachelemente: Kontrollstrukturen</b> . . . . .	50
2.6. Reflexoptokoppler . . . . .	58
2.7. Digitaler Sensor: Der Ultraschallsensor . . . . .	62
2.8. Umwelt Sensoren . . . . .	65
2.9. <b>Daten Speichern und Auslesen</b> . . . . .	67
2.10. <b>C Sprachstruktur: Schleifen</b> . . . . .	69

---

<b>3. Actoren</b>	<b>73</b>
3.1. RGB-LEDs . . . . .	74
3.2. C Sprachelemente: millis() versus delay()	76
3.3. Lautsprecher . . . . .	77
3.4. Der Transistor als Schalter oder zur Verstärkung . . . . .	79
3.5. Peltier-Element . . . . .	84
3.6. Servomotor . . . . .	86
3.7. C Sprachstruktur: Library . . . . .	93
3.8. DC-Motoren und externe Spannungsquelle . . . . .	94
3.9. Integrierte Schaltkreise, IC . . . . .	95
3.10. Der IC L293D zur Motorsteuerung . . . . .	96
3.11. Schrittmotoren . . . . .	102
3.12. Display . . . . .	104
<b>4. Projekte mit dem Arduino</b>	<b>105</b>
4.1. Projektmanagement . . . . .	106
4.2. Das optische Tee-Tassen-Thermometer ( $oT^3$ ) . . . . .	107
4.3. Das eigene Motor Arduino Shield . . . . .	108
4.4. SchulBot ein autonomes Fahrzeug . . . . .	109
4.5. Paperduino-UNO, das selbstgebaute Arduino-Board . . . . .	112
4.6. Roboter Arm . . . . .	113
<b>5. Die Stationskarten</b>	<b>115</b>
<b>Anhänge</b>	<b>122</b>
<b>A. Hintergrundwissen Arduino</b>	<b>123</b>
A.1. Fehlersuche . . . . .	124
A.2. Analoge und digitale Signale . . . . .	125
A.3. Arduino's Verbindung zur Aussenwelt: PINs . . . . .	127
A.4. Interruptsteuerung . . . . .	135

---

---

<b>B. Data Kommunikations Systeme</b>	<b>138</b>
B.1. Serielle Schnittstelle - die Verbindung mit dem PC . . . . .	139
B.2. I2C Schnittstelle . . . . .	146
<b>C. Hintergrundwissen: Programmieren</b>	<b>147</b>
C.1. Rechnen im dem Mikrocontroller . . . . .	148
C.2. Programmablaufplan . . . . .	149
<b>D. Hintergrundwissen: Elektronik</b>	<b>150</b>
D.1. Multimeter . . . . .	151
D.2. Farbcodes von Widerständen . . . . .	152
D.3. Hintergrundwissen LED . . . . .	154
D.4. Schaltungsentwicklung mit Fritzing . . . . .	155
<b>E. Hintergrundwissen: Free Software Foundation and Creative Commons</b>	<b>156</b>
E.1. Open Source und Creative Commons . . . . .	157
Literatur . . . . .	158

# 1 — Einführung

„Ich denke, dass es einen Weltmarkt für vielleicht fünf Computer gibt.“

---

Thomas Watson, IBM-Vorsitzender, 1943

Der Arduino ist eine einfache und vielseitige Experimentierplattform, eine Art kleiner Mini-Computer. Sie besteht aus der Arduino-Hardware, einer handtellergroßen Platine mit einer Vielzahl von Anschlüssen und der Arduino-Software, einer Entwicklungsumgebung (IDE) für den PC, in der man die Programme schreibt, die auf dem Gerät ablaufen. Das Besondere an dem Arduino-Board ist, dass der Schaltplan freie Hardware (eng. open source hardware) ist.

Nach dem du dieses Kapitel gelesen und alle Aufgaben gemacht hast, sollten dir folgende Begriffe bekannt sein.

- Arduino Board
- IDE
- Sketch
- *setup()* und *loop()*

## Info:

open source hardware ist eine Hardware oder ein sonstiges technisches Gerät, die nach lizenzkostenfreien Bauplänen hergestellt werden.

---

## 1.1. Das Arduino Projekt

Die Arbeit mit Mikrocontrollern und die Ansteuerung von Hardware hat selbst für Informatiker immer etwas Geheimnisvolles an sich – dafür müsse man wochenlang Datenblätter lesen und dann kryptischen Code schreiben, so die Vorstellung. Aber es geht auch anders! Jedenfalls seit 2005: Da waren es die Studierenden des Interaction Design Institute im italienischen Ivrea leid, mit ihren Computern nur über Tastatur und Maus zu kommunizieren. Sie suchten nach einer einfachen Möglichkeit, um ihre Ideen für neuartige Interaktionen zwischen Mensch und Maschine, ihre Kunst- und Roboterprojekte in funktionsfähige Prototypen umzusetzen.

Massimo Banzi entwickelte daraufhin mit einer handvoll Mitstreitern ein einfaches, günstiges Mikrocontroller-Board samt Programmiersprache und Entwicklungswerkzeug und nannte es Arduino, nach einem lokalen König aus dem elften Jahrhundert.

Das kleine Entwicklungs-Board trat schnell einen Siegeszug an – gerade weil es sich nicht so sehr an lötende Nerds, sondern an Quereinsteiger wie Designer, Künstler oder auch Schüler richtete, die vorher nur in Ausnahmefällen Software entwickelt oder gar eigene Hardware gebaut hatten.

### 1.1.1. Was kann man mit dem Arduino alles machen?

Das Netz ist voller Projekte, die von einer Laserharfe über twitternde Topfpflanzen bis zu autonom fliegenden Luftschiffen reichen.

Ich möchte das Projekt von Zoe Romano: [A low-cost robotic hand \(tutorial\) mirroring your own fingers](#) (siehe Video) genauer anschauen. Ziel dieses Projektes ist es die Bewegungen der Hand zu erfassen und auf eine RoboterHand zu übertragen.

Dieses Projekte besteht aus drei Teilen (siehe Abb. 1.1):

- Einem Sensor, der Handschuh mit den Biegesensoren, die die Fingerbewegungen messen.
- Einer Logik, das Arduino Board, das die Messwerte verarbeitet und die Bewegung steuert.
- Einem Aktor, der RoboterHand, die die Bewegungen ausführt.

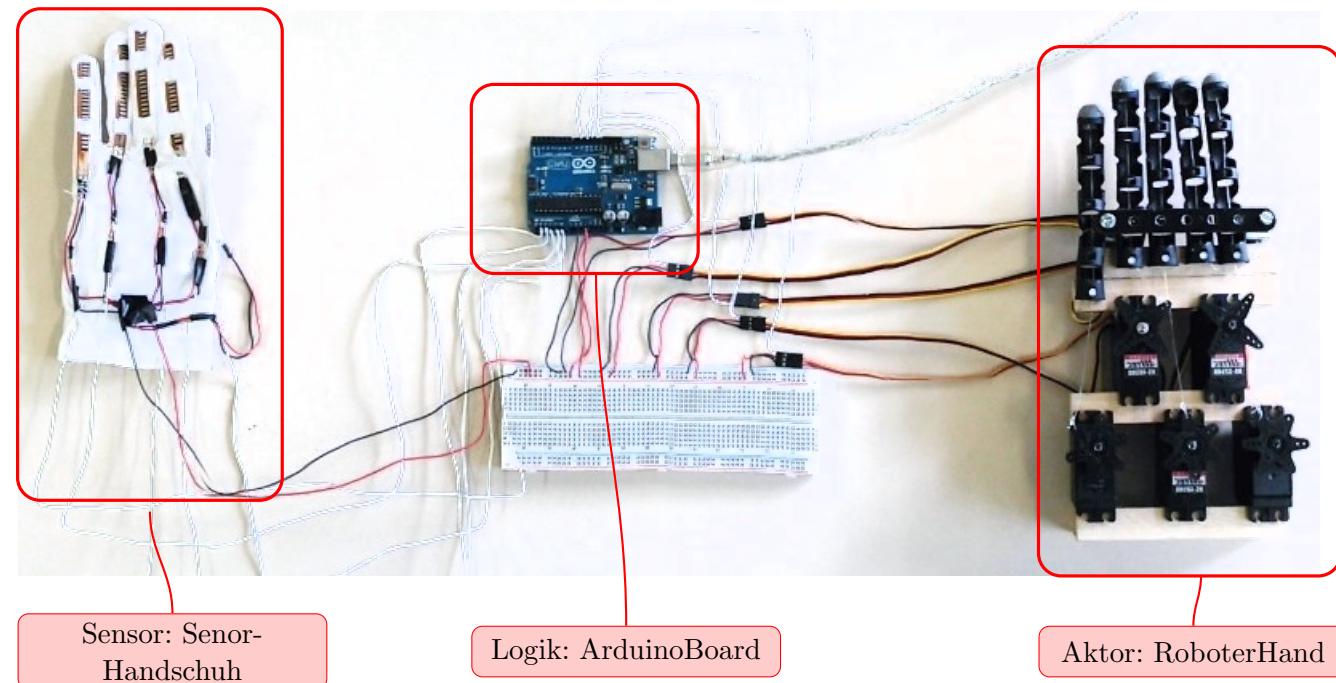
Diese Projekt ist ein gutes Beispiel für die Arbeitsweise eines Mikrocontrollers. Ein Mikrocontroller kann über Sensoren Daten einlesen (Eingabe), diese Daten verarbeiten (Verarbeiten) und Aktoren (LEDs, Motoren, Lautsprecher, ...) steuern (Ausgabe).

#### Info:

Eingabe, Verarbeiten und Ausgabe: das EVA-Prinzip

---

Genau wie dieses Projekt ist auch dieses Skript aufgebaut. Im ersten Kapitel erfährst du einiges über die Funktionsweise des Arduino-Boards. In Kapitel 2 wirst du mit verschiedenen Sensoren arbeiten und in Kapitel 3 wird schließlich die Funktionsweise verschiedener Aktoren thematisiert.



**Abbildung 1.1.:** Aufbau eines Arduino-Projektes

### 1.1.2. Projekte und Ideen, die ohne Arduino wirklich schwer wären!

Bevor du dich ausführlich mit dem Arduino Uno beschäftigst, möchte ich dir eine kurze Auswahl verschiedener Projekte vorstellen

---

## Manufaktur – Arduino als Maker

Werkstücke mit Hilfe eines PC fertigen, war bis vor kurzer Zeit nur mit sehr teuren CNC-Maschinen und teurer Spezialsoftware möglich. Durch die Entwicklung des Arduino-Projekts (und anderer ähnlicher Projekte) wurde es möglich CNC-Maschinen und 3D-Drucker viel billiger zu bauen. Natürlich sind diese selbstgebauten Maschinen nicht mit industriellen Maschinen vergleichbar, dafür sind die aber um den Faktor 100-1000 billiger und somit in Bereichen einsatzbar die zuvor finanziell nicht möglich waren.

Zwei dieser Projekte möchte ich herausheben:

- [Shapeoko An Open Hardware project by Edward Ford](#)
- [3Drag: the Open Electronics way to RepRap](#)

### Info:

CNC (Computerized Numerical Control) Computergestützte numerische Steuerung



(a) Shapeoko open hardware CNC-Fräse



(b) 3Drag open electronic 3D-Drucker

**Abbildung 1.2.:** Manufaktur: Arduino steuert CNC-Maschinen

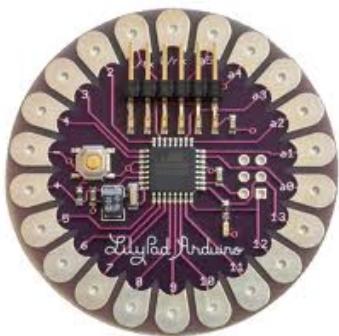


Abbildung 1.3.: Arduino LilyPad

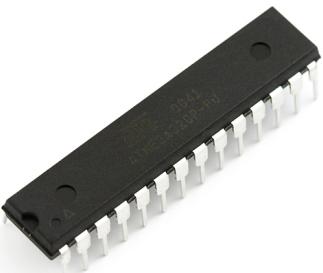


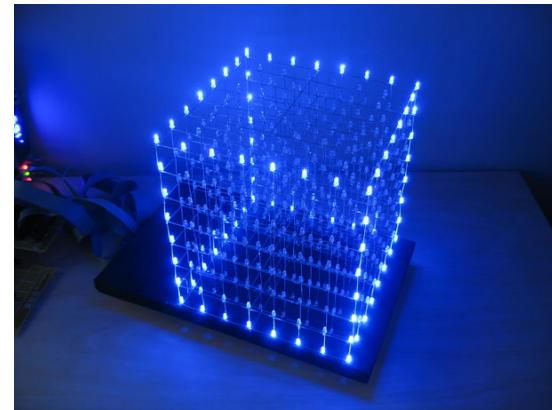
Abbildung 1.5.: Der ATmega328-Mikrocontroller (CC BY-NC-SA 3.0 by sparkfun.com)

### 1.1.3. Art und Design – Arduino als Künstler

Das Arduino-Board wurde ursprünglich für die universitäre Ausbildung zum „Interaction Designer“ entwickelt. Die Motivation liegt darin Elektronik als kreatives Material einzusetzen. So kann mit Hilfe eines LED-Cubes (siehe Abb. 1.4(a)) Musik als fallende Tropfen oder Silvesterraketen dargestellt werden.

Mit einem Arduino LilyPad können intelligente Kleidungsstücke designed werden. Zum Beispiel eine Tasche, mit steuerbaren LEDs (siehe Abb. 1.4(b)).

- [LED cube 8x8x8 demo by chrmoe](#)
- [LilyPad Arduino Blinking Bike Safety Patch by bekathwia](#)



(a) LED Cube 8x8x8 by chr



(b) LilyPad Arduino Blinking Bike Safety Patch by bekathwia (by-nc-sa)

Abbildung 1.4.: Art: Arduino als Künstler

### 1.1.4. Der Mikrocontroller

Der Kern eines Arduino ist der Mikrocontroller. Ein Mikrocontroller ist aus miniaturisierten elektrischen Schaltkreisen – den integrierten Schaltkreisen aufgebaut.

---

ICs sind komplexe elektrische Schaltungen, die auf kleinstem Raum in einem Siliziumkristall platzen finden. Was zu den Pionierzeiten der Elektronik noch mit unzähligen Bauteilen (Dioden, Transistoren, Widerständen und Kondensatoren) platzraubend Aufgebaut werden musste, findet heute auf kleinstem Raum Platz in unterschiedlich großem schwarzen Plastikgehäusen mit einer bestimmten Anzahl von Beinchen, den sogenannten PINs.

In der Abb. 1.5 siehst du den ATmega328-Mikrocontroller, der auch auf dem Arduino Uno Board verbaut ist. Du könntest ihn in dieser Form mit Hilfe einer geeigneten Spannungsversorgung und einem Taktgebers benutzen. Leider ist das aber sehr kompliziert. Aus diesem Grund wird der IC auf dem Arduino Uno Board montiert. Aufgabe dieses Entwicklungs-Boards ist es dir die Welt der Mikrocontroller auf möglichst einfache Weise zugänglich zu machen.

#### Info:

IC - Integrated Circuit

### Der innere Aufbau eines $\mu$ Cs

Der Aufbau eines Mikrocontrollers entspricht dessen eines Computers. Der Unterschied besetzt in der Leistungsfähigkeit, der Rechenleistung und im Speichervolumen.

In der Abb. 1.6 sind die Bestandteile eines ATmega schematisch dargestellt.

- **Die CPU** steuert und kontrolliert die anderen Teile des Mikrocontrollers durch dekodieren und ausführen von (Maschinen-)Befehlen. Sie kann den Speicher adressieren, Ein- bzw. Ausgänge verwalten und auf sogenannte Interrupts reagieren.
- **Der Datenbus** verbindet alle Bestandteile des Mikrocontrollers miteinander. Die CPU fordert zum Beispiel Daten aus dem Speicher an, die Daten werden auf den Bus gelegt und können unmittelbar von der CPU verarbeitet werden.
- **Die Interrupt Steuerung (IRW)** kann auf stattfindende Ereignisse reagieren. Dazu wird die aktuelle Aufgabe unterbrochen um sofort auf das Ereignis reagieren zu können.
- **Speicher RAM und ROM** (siehe Aufgabe)

Abbildung 1.6.: Blockdiagramm

#### 1.1.5. Aufgabe:

Was ist eigentlich der Unterschied zwischen RAM und ROM Speicher. Liese den Artikel <http://de.ccm.net/faq/3349-der-unterschied-zwischen-ram-und-rom> und erkläre die jeweilige Funktion in eigenen Wörtern.

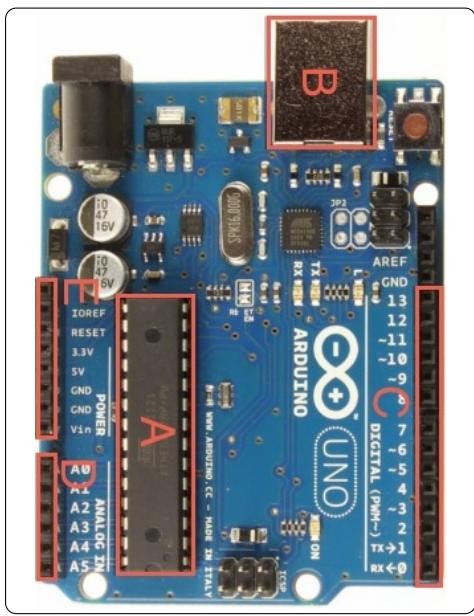


Abbildung 1.7.: Arduino Uno R3

#### Info:

PWM für pulse width modulation

## 1.2. Das Arduino Uno Board

Hier ist die neuste Version der Arduino-Hardware (Stand: März 2012) abgebildet, der Arduino Uno (R3). Die wichtigsten Anschlüsse und Bestandteile sind die folgenden:

- A **Der Mikrocontroller** ist das wichtigste Bauteil des Arduinos. Beim Arduino Uno ist ein Atmel ATmega328 verbaut.
- B Über den **USB-Anschluss** verbindet man den Arduino mit dem PC. Über diese Verbindung überträgt man den Programmcode, das der Arduino ausführen soll, in den ROM Speicher des Mikrochips. Während das Programm ausgeführt wird, kann außerdem Daten zwischen PC und Arduino austauscht werden.
- C Die rechte Buchsenleiste besteht aus 14 **digitalen Ein- oder Ausgängen**, sogenannten digitalen PINs. Ein digitaler PIN kann mit Hilfe eines Schalters oder Temperatur-Sensors Daten empfangen oder mit Hilfe eines Lautsprecher, einer Leuchtdiode oder einem anderem Bauteile Daten aussenden. Digital bedeutet, dass an den digitalen PINs entweder 0 V (digital 0/LOW) oder 5 V (digital 1/HIGH) Spannung anliegt. Ein digitaler PIN kann maximal 40mA Strom liefern.  
Sechs digitale PINs besitzen noch die Zusatzfunktion der **PWM** (Pulsbreitenmodulation). Diese PINs sind zusätzlich mit einer  $\circ$  gekennzeichnet.
- D Der untere Teil der linken Buchsenleiste ist mit **Analog in** beschriftet. Mit Hilfe der analogen PINs können Spannungen zwischen 0V und 5V gemessen werden.
- E die linke obere Buchsenleiste trägt die Bezeichnung **Power**: "Vin" (Voltage in) Eingangsspannung, "Gnd" (Ground) 0 Volt, "5V" (5 Volt), "3V3" (3,3 Volt), "Reset"

Es gibt noch weitere Bauteile: Stromversorgung, Reset-Taster, Status-LEDs, Kondensatoren, Widerstände und einen Quarz.

### 1.2.1. Aufgaben

#### Aufgabe 1:

Im PDF-File [Arduino Uno V3 Pinout Diagram](#) ist der Aufbau des Arduino Uno nochmals sehr genau erklärt. Suche und kennzeichne alle GND-PINs, den Reset-Taster.

### 1.3. Software: die Arduino-IDE

Die Arduino Entwicklungsumgebung ist für Linux, OS X und Windows kostenlos verfügbar und kann von der Arduino Homepage heruntergeladen werden: [Download the Arduino Software](#).

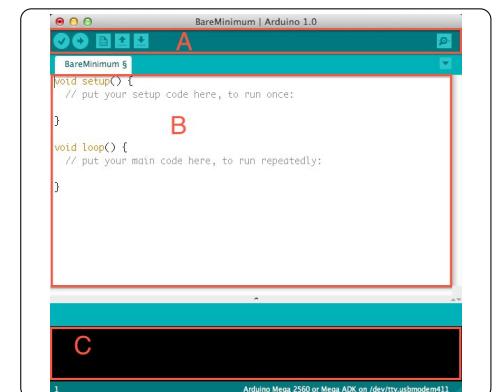
Die Entwicklungsumgebung besteht aus folgenden Teilen

- A den Menüs
- B Texteditor zum schreiben des Programm-Codes
- C einer Nachrichtenkonsole

Die Entwicklungsumgebung stellt automatisch eine Verbindung zur Arduino Hardware her um Programme hochzuladen und um weiter Daten mit dem Arduino auszutauschen. Ein fertiges Programme wird "Sketch" genannt. Ein "Sketch" wird mit einem Text-Editor geschrieben und automatisch mit der Dateiendung ".ino" gespeichert. Jeder Sketch wird automatisch in einem Eigenen Ordner gespeichert. Der Nachrichtenbereich gibt Feedback beim Speichern und Exportieren von Sketchen und eventuell auftretenden Fehlern.

Die Bedeutung der Icons in der Symbolleiste A sind:

-  **Verify**: Überprüft deinen Code auf syntaktische Fehler.
-  **Play**: Kompiliert deinen Programmcode und lädt ihn auf das Arduino Board.
-  **New**: Erschafft einen neuen "sketch".
-  **Open**: Öffnet ein Menu mit allen Sketchen, die auf dem PC (deine eigenen Sketche und Beispiel-Sketche) gespeichert sind. Durch anklicken kannst du einen Sketch öffnen.
-  **Save**: Speichert deinen "sketch".
-  **Serial Monitor**: öffnet eine Konsole, die über die USB-Schnittstelle eine serielle Verbindung zum Arduino aufbaut.



**Abbildung 1.8.:** Die Arduino IDE  
(Version 1.0)

---

### **1.3.1. Aufgaben**

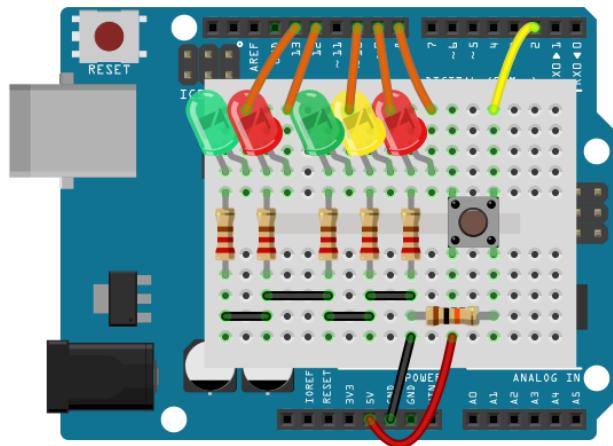
Starte die Arduino IDE. Verbinde mit dem UBS-Kabel dein Arduino Uno Board mit dem PC. Schaue im Menu “Werkzeuge - Port” nach, ob dein Arduino verbunden ist. Wenn dein Arduino richtig verbunden ist, dann kannst du ebenfalls im Menu “Werkzeuge” den “Serieller Plotter” öffnen. Erkläre die Funktion des Serieller Plotter’s.

---

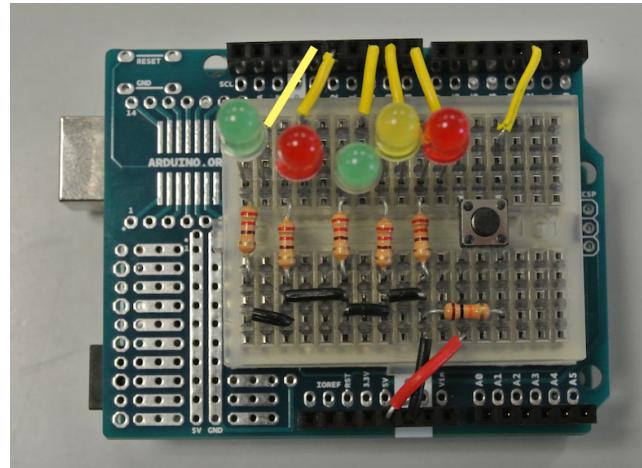
## 1.4. Blink, dein erster Arduino Sketch

### Vorbemerkungen

Im ersten Teil dieser Einführung wirst du eine Ampelanlage mit Auto- und Fußgängerampel aufbauen und programmieren. Die benötigte Schaltung und den Sketch wirst du in den folgenden Abschnitten Stück für Stück erarbeiten. Es ist deshalb wichtig, dass du die jeweiligen Schaltpläne möglichst genau aufbaust. So sparst du Zeit, da du deine Schaltung nicht umbauen musst.



(a) Schaltplan



(b) Reale Schaltung

**Abbildung 1.9.:** Die fertige Ampel

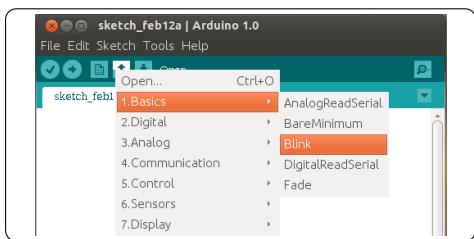


Abbildung 1.10.: Öffnen eines Sketches



Abbildung 1.11.: LED und Vorwiderstand

#### Info:

Achtung verbinde nie eine LED mit dem PIN Vin. Je nach Versorgungsspannung wird die LED explosionsartig zerstört!

## Jetzt geht es los!

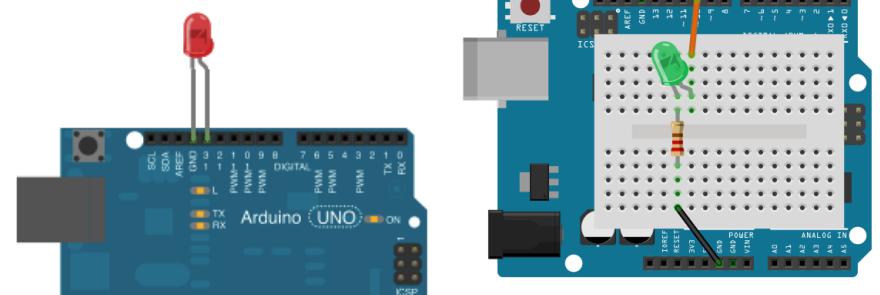
Um deinen ersten Arduino Sketch auf den Arduino zu übertragen und auszuführen musst du folgende Schritte nacheinander ausführen:

1. Starte die Arduino IDE und öffne das Beispiel "Blink" (siehe Abb. 1.10).
2. Verbinde nun das Arduino-Board und deinen PC mit Hilfe des UBS-Kabels.
3. Drücke in der Arduino-IDE das Icon um den Sketch zu übersetzen, auf den Arduino zu übertragen und um das Arduino-Programm auszuführen.

Jetzt sollte die kleine LED mit der Beschriftung "L" periodisch blinken.

### 1.4.1. Eine externe LED blinken lassen

Es kann auch eine "externe" LED zum Leuchten gebracht werden. Am schnellsten geht das, wenn man die LED direkt auf dem Arduino-Board in den digitalen Anschluß PIN13 und den daneben liegenden Anschluß GND steckt (siehe Abb. 1.12). Sollte die LED von PIN13 blinken aber die externe LED nicht so hast du die LED falsch herum angeschlossen.



(a) Schnelle Methode (b) LED mit Vorwiderstand an PIN 10

Abbildung 1.12.: Anschluss einer externen LED

---

Die schnelle Methode sollte nur zum Testen verwendet werden, da die LED kaputt gehen kann. Deshalb sollte eine LED immer zusammen mit einem Vorwiderstand betrieben werden (siehe Abb. 1.12(b)). Die Funktion des Vorwiderstandes ist es, die maximale Leistungsaufnahme der LED zu begrenzen. Für eine normale LED ist ein Vorwiderstand von  $R = 220\Omega$  ausreichend. Mehr Wissen zum Thema Elektronik findest du im Anhang D Hintergrundwissen Elektronik.

### Farb-Code von Widerständen

Wenn du die Abbildung 1.11 den Widerstand genauer ansiehst, dann erkennst du die vier farbigen Ringe, die ersten drei Ringe ergeben den Widerstandswert, der vierte Ring die Fertigungstoleranz an. Es gibt viele Internetseiten, auf denen du die Farbcodes nachschauen kannst. Mein persönlicher Favorit ist die Seite <http://resisto.rs>.



---

### 1.4.2. Ein kurzer Blick auf den Sketch-Code

Der Arduino wird in der Computersprache C programmiert. C ist zwar alt, aber immer noch eine der beliebten Programmiersprachen. Schau dir den Programm-Code des Sketches "Blink" (siehe Listing 1.1) einmal genauer an.

```
1  /*
2   * Blink
3   * Turns on an LED for one second, then off
4   *     for one second, repeatedly.
5   * This example code is in the public domain
6   *
7   * void setup() {
8   *     // initialize the digital pin as an
9   *     // output.
10  *     // PIN 13 has an LED connected on most
11  *     // Arduino boards:
12  *     pinMode(13, OUTPUT);
13  *
14  *     void loop() {
15  *         digitalWrite(13, HIGH);
16  *         // set the LED on
17  *         delay(1000);    // wait for a second
18  *         digitalWrite(13, LOW);
19  *         // set the LED off
20  *         delay(1000);    // wait for a second
21  *     }
22  }
```

Kommentarbereich: Hier wird die Aufgabe, Funktionsweise und weiter wichtige Informationen dokumentiert.

Die setup()-Methode wird nur einmal ausgeführt

PIN13 wird als OUTPUT definiert.

Die loop()-Methode wird immer wieder ausgeführt. Dadurch wird die LED für eine Sekunde (1000ms) eingeschalten und ...

... anschließend ausgeschaltet.  
Nach einer weiteren Sekunde wiederholt sich das Einschalten.

**Listing 1.1:** Blink

### 1.4.3. Die bisher verwendeten C Befehle

In dem Sketch "Blink" werden unterschiedliche Strukturen der Programmiersprache C verwendet. Auf der Arduino Home-page findest du unter [www.arduino.cc/en/Reference/HomePage](http://www.arduino.cc/en/Reference/HomePage) alle C Sprachelemente, die du für die Programmierung eines Arduino nötig sind.

Die Tabelle 1.1 listet die bisher verwendeten Befehle auf und beschreibt kurz ihre Funktion.

Name	Funktion
<code>pinMode(PIN, MODE)</code>	Konfiguriert den angegebenen PIN als INPUT oder OUTPUT.
<code>digitalWrite(PIN, VALUE)</code>	Schreibt den VALUE mit einem Wert HIGH oder LOW auf den digitalen PIN.
<code>delay(MS)</code>	Stoppt das Programm für MS Millisekunden. 1000ms entspricht einer Sekunde.

#### Info:

C wurde vom Informatiker Dennis Ritchie in den frühen 1970er Jahren an den Bell Laboratories für die Systemprogrammierung des Betriebssystems Unix entwickelte.

**Tabelle 1.1.:** Übersicht der verwendeten C Befehle

### 1.4.4. Aufgaben

#### Aufgabe 1

- Erstelle eine Tabelle, in der du alle C Sprache-Elemente und Strukturen dokumentierst. Orientiere dich an dem Aufbau der Tabelle 1.1.
- Lies auf [www.arduino.cc/Reference/HomePage](http://www.arduino.cc/Reference/HomePage) die ausführliche englische Dokumentation der folgenden Befehlen
  - `pinMode()` ([www.arduino.cc/en/Reference/PinMode](http://www.arduino.cc/en/Reference/PinMode))
  - `digitalWrite()` ([www.arduino.cc/en/Reference/DigitalWrite](http://www.arduino.cc/en/Reference/DigitalWrite))
  - `delay()` ([www.arduino.cc/en/Reference/delay](http://www.arduino.cc/en/Reference/delay))

Gibt es zusätzliche Informationen, die für dich wichtig sein können. Beantworte folgende Fragen schriftlich:

- Was passiert, wenn du einen digitalen PIN zum Steuern einer LED mit `pinMode()` auf INPUT setzt?
- Gibt es für digitale PIN noch weiter Modi?

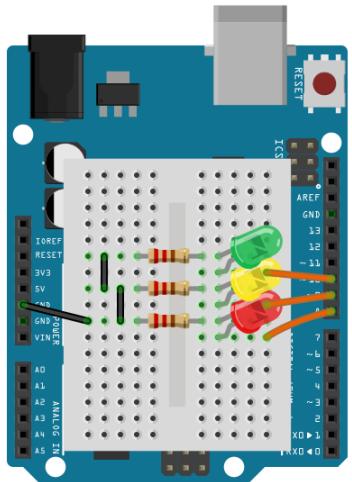


Abbildung 1.13.: LEDs an PINs 10, 9 und 8

- Gibt es noch mehrere `delay()`-artige Befehle? Eine Millisekunde ist zwar für einen Menschen eine unheimlich kurze Zeitspanne, für einen Arduino aber schon ziemlich lange!

Ergänze deine Tabelle entsprechend.

- c) Verändere den Sketch "Blink" so, dass eine LED am PIN 10 SOS<sup>1</sup> blinkt. Den Schaltplan findest du in Abb. 1.12(b) Speichere den veränderten Sketch unter dem Namen "SOS" ab.
- d) Erweitere deine Tabelle mit der Befehlsübersicht um den folgenden Befehl:
1. `random()` ([www.arduino.cc/en/Reference/Random](http://www.arduino.cc/en/Reference/Random)).
- ein.
- d) Erstelle den neuen Sketch "RandomBlink". Die LED soll an PIN13 nun zufällig an- und ausgehen. Dazu musst du in der `loop()`-Routine folgenden Befehl ergänzen "`delay(random(2000))`".

## Aufgabe 2: Steuern einer Autoampel

Schließe an die digitalen PINs 10, 9 und 8 je eine grüne, gelbe und rote LED mit Vorwiderstand an. Die Schaltung findest du in Abb. 1.13. Erstelle den neuen Sketch 'Ampel', der die 3 LEDs in der Lichtfolge einer Ampel leuchten lässt. Zuerst soll die grüne LED leuchten, achte dann auf die richtige Reihenfolge!

<sup>1</sup>Im Morse Alphabet: 3 mal kurz (für S) 3 mal lang (für 'O') 3 mal kurz (für 'S') gefolgt von einer langen Pause und dann wieder vor vorne

## 1.5. Der Arduino Sketch

Ein Arduino Sketch besteht aus mehreren Bereichen:

- dem Kommentarbereich zur Dokumentation und Funktionsweise des Arduino Sketches. Ein ausführlicher Kommentarbereich hilft dir und anderen deinen Sketch zu verstehen. Du wirst oft auf deine selbst erstellten Sketche zurückgreifen.
- dem Definitionsbereich von Variablen und Libraries
- dem Methodenbereich, mit der `setup()`-Methode, die einmal ausgeführt wird und der `loop()`-Methode, die immer wieder ausgeführt wird. Zusätzlich können noch weitere Methoden dazukommen (dazu aber später mehr). `setup()`- und `loop()`-Methode müssen in jedem Sketch vorhanden sein.

```
1  /*
2   * Sketch Name und Beschreibung
3
4   * Datum:          ○
5   * Author:         ○
6   * Version:        ○
7   */
8
9 // Definition von Variablen    ○
10
11 void setup() {                ○
12
13 }
14
15 void loop() {                ○
16
17 }
```

**Listing 1.2:** Bare Minimum

### Info:

Aus Platzgründen wir im folgendem der Kommentarbereich bei den Beispiel-Sketchen immer weggelassen. Du solltest aber bei jedem deiner Sketche einen ausführlichen Kommentarbereich schreiben.

---

### **1.5.1. Aufgaben**

#### **Aufgabe 1**

Dokumentiere die beiden Sketche “SOS” und “Ampel”.

## 1.6. Mit dem PC kommunizieren

Bisher hast du, oder besser die ArduinoIDE mit dem Arduino "gesprochen". Jedes mal, wenn du einen Sketch auf das Board geladen hast. Natürlich kann man auch direkt mit dem Arduino sprechen. Dies geschieht über eine sogenannte Serielle Schnittstelle. Um zu verstehen wie das funktioniert, öffnest du den Beispielsketch "DigitalReadSerial" (siehe Abb. 1.14).

Der Beispielsketch 'DigitalReadSerial' ist in Listing 1.3 abgebildet. In der `setup()`-Methode wird zunächst eine serielle Verbindung zum PC hergestellt. Wichtig dabei ist die Übertragungsrate von 9600 Baut/s. Dann wird der digitale PIN2 auf INPUT gesetzt.

```
1 void setup() {
2   Serial.begin(9600);
3   pinMode(2, INPUT);
4 }
5
6
7 void loop() {
8   int sensorValue = digitalRead(2);
9   Serial.println(sensorValue);
10 }
```

Serielle Kommunikation startet

Aktuellen Sensor-Wert übermitteln

Listing 1.3: digitalReadSerial

Damit der Arduino etwas an den PC übermitteln kann, brauchst du noch einen Taster (eng. Button) mit dessen Hilfe du zwischen LOW und HIGH umschalten kannst.

### 1.6.1. Aufbau der Schaltung

Für die Schaltung brauchst du einen Taster und einen  $10\text{k}\Omega$  Widerstand.

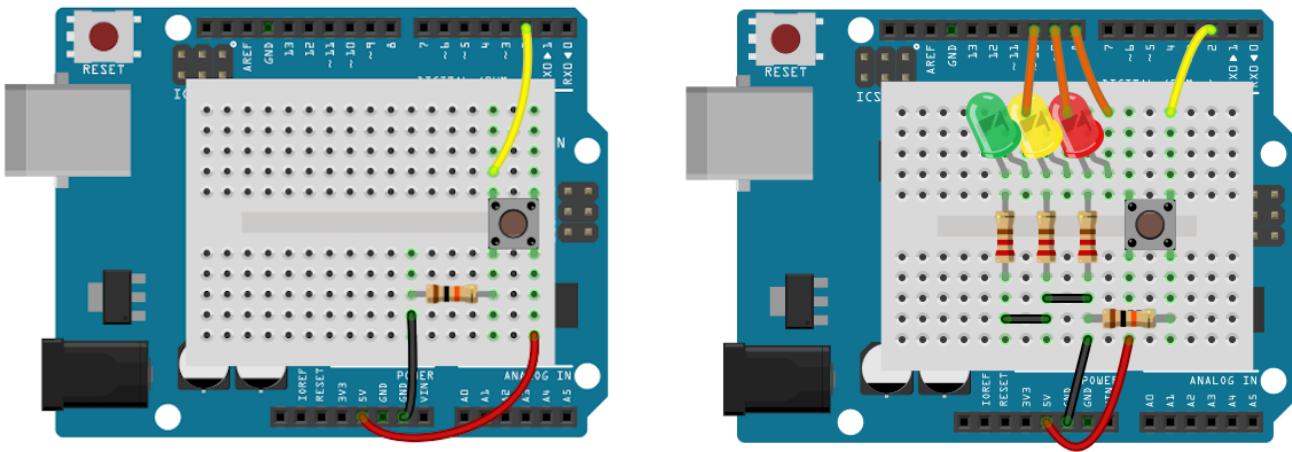
### 1.6.2. Aufgaben

Erweitere deine Schaltung entsprechend Abb. 1.16. Aus Platzgründen ist der PushButton näher an die rote LED eingebaut. Rechts wird der Platz noch gebraucht!



Abbildung 1.14.: "DigitalReadSerial" öffnen





(a) Grundschaltung

(b) Autoampel mit PushButton

**Abbildung 1.15.:** Das Schaltbild zum Beispiel Sketch “DigitalReadSerial”

### Aufgabe 1

Testet die Schaltung zunächst mit dem Sketch “DigitalReadSerial” (siehe Listing 1.3). Öffne den “Serieller Monitor” im Menu “Werkzeuge” der Arduino LED. Schau dir die Ausgabe im “Serieller Monitor” an.

### Aufgabe 2:

Lies die Dokumentation zum DigitalReadSerial auf der Seite [www.arduino.cc/en/Tutorial/DigitalReadSerial](http://www.arduino.cc/en/Tutorial/DigitalReadSerial) genau durch und dokumentiere die folgenden Befehlen

1. Serial.begin() ([www.arduino.cc/en/Reference/Serial/Begin](http://www.arduino.cc/en/Reference/Serial/Begin))
2. Serial.println() ([www.arduino.cc/en/Reference/Serial/Println](http://www.arduino.cc/en/Reference/Serial/Println))

---

### Aufgabe 3:

Öffne dann den Sketch "Button" aus Beispiel/Digital. Schau die den Sketch genau an. Ergänze die Befehle (die noch nicht in deiner Tabelle dokumentiert sind) in deiner Tabelle. Schreibe den Sketch so um, dass die grüne LED verwendet wird.

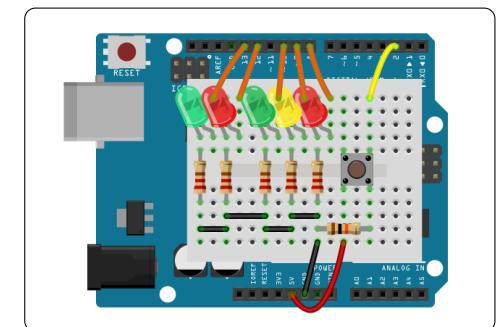
### Aufgabe 4: Projekt

Erweitere deine Schaltung aus Aufgabe 3 entsprechend Abb. 1.16, so dass du eine Autoampel (rot, gelb und grün) mit Fußgängerampel (rot und grün) mit Push-Button aufgebaut hast. Wenn ein Fußgänger auf den Kopf drückt (gedrückt hält), soll die Autoampel von grün auf rot schalten und die Fußgängerampel grün werden. Nach einer gewissen Zeit, soll die Autoampel wieder auf grün schalten.

```
1 if (digitalRead(2)) {  
2     // Fussgaengerample soll von rot auf  
3     // gruen schalten und nach vier  
4     // Sekunden wieder auf rot  
5 }
```

**Listing 1.4:** Fußgängerampel mit Push-Button

Hier reagiert der Arduino auf das drücken des Push-Buttons



**Abbildung 1.16.:** Push-Button mit LED

#### Info:

Der Button muss lange gedrückt werden, da der Sketch nur an einer Stelle den Wert des Buttons auslesen kann. Später lernst du, wie du den Arduino programmieren kannst, damit er zu jeder Zeit den Button auslesen kann.

---

## 1.7. C Sprachelemente: Variablen

Daten zu speichern und zu verarbeiten ist eine zentrale Aufgabe eines Mikrocontrollers oder Computers. Die Programmiersprache stellt für diesen Zweck extra Variablen bereit.

Eine Variable kannst du dir wie eine Art Box vorstellen. In so einer Box kannst du Zahlen, Wörter oder andere Dinge ablegen und du kannst sie später wieder hervorholen, um mit ihnen zu arbeiten. Zum Beispiel können Messdaten eines Sensors gespeichert oder/und der gespeicherte Wert für eine Berechnung oder Ausgabe ausgelesen und/oder übergeben werden.

In diesem Abschnitt lernst du, wie Variablen für unterschiedliche Aufgaben erzeugt und manipuliert werden können.

### 1.7.1. Das Erzeugen von Variablen

Bevor eine Variable im Sketch verwendet werden kann, muss die Variable deklariert werden, d.h. es muss eine Box erstellt werden. Deklarieren einer Variablen bedeutet, dass im Mikrocontroller Speicherplatz für die Variable zur Verfügung gestellt wird. In diesem Speicherbereich wird der Typ, und gegebenenfalls schon ein Wert gespeichert. Variablen müssen nicht gleich initialisiert (ein Wert zugewiesen) werden, wenn sie deklariert werden, aber es ist oft nützlich ihnen einen definierten Anfangswert zuzuweisen.

#### Info:

Das Erstellen einer Variablen wird deklarieren genannt.

#### Info:

Das Zuweisen eines Wertes wird initialisieren genannt.

The diagram illustrates the code from Listing 1.5, which declares and initializes two integer variables. A green rectangular highlight covers the first three lines of code. Three red arrows point from the right side of the code to three callout boxes on the right. The top arrow points to the declaration of `inVar1`. The middle arrow points to the initialization of `inVar1`. The bottom arrow points to the declaration and initialization of `inVar2`.

```
1 int inVar1;          // Deklaration einer Variablen inVar1
2 inVar1 = 9;           // Initialisation einer Variablen inVar1
3
4 int inVar2 = 0;       // Deklaration und Initialisierung der Variablen inVar2
```

**Listing 1.5:** Deklaration und Initialisation

Deklaration einer Variablen `inVar1`

Initialisation einer Variablen `inVar1`

Deklaration und Initialisierung der Variablen `inVar2`

### 1.7.2. Sinnvolle Namensgebung für Variablen

Es ist extrem wichtig sinnvolle Namen für Variablen zu vergeben, damit sind Namen gemeint, die sich nach Möglichkeit selbst erklären. Doener1, doener2 mag zwar sehr lustig sein, aber du hast in zwei Wochen keine Ahnung mehr was du

---

damit speichern wolltest! Darüber hinaus stellen die Variablen Doener und doener zwei Unterschiedliche Boxen dar. C unterscheidet Groß- und Kleinschreibung!

#### Tipps für sinnvolle Namen von Variablen:

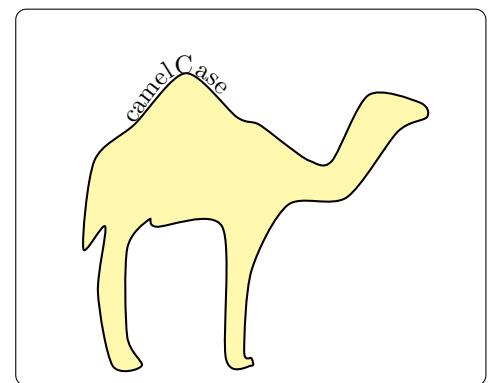
Wenn du folgende Tipps befolgst, dann wirst du weniger Probleme haben!

- Benenne deine Variable nach dem Inhalt: Wenn der Messwert eines Temperatur-Sensors gespeichert werden soll ist ein sinnvoller Name measTemp (eng. measure messen).
- Benutzte die camelCase-Notation, d.h. der erste Buchstabe eines Variablenamens wird immer klein geschrieben. Wenn die Variable aus mehreren Wörtern zusammengesetzt ist, wird jedes Wort mit einem großen Buchstaben angefangen. Bekanntes Beispiel dieser Notation sind die Namen von Apple-Produkten: iPhone, iPad, iPod, iMac aber auch McDonalds.
- Es gibt verschiedene Variable-Typen. Oft ist es sinnvoll dass der Type der Variablen im Namen ablesbar ist. Zum Beispiel eine Variable die nur ganze Zahlen enthält wäre intVar eine gute Wahl.
- Verwende nie ä, ö, ü oder ß. Diese Buchstaben werden vom C und der ArduinoIDE nicht unterstützt. Sonderzeichen (!, ?, /, -, # etc.) gehen auch nicht, diese Zeichen sind für (Rechen-) Operationen in C reserviert.

Du wirst viele Beispiel-Skizzen kennenlernen, die nicht der camelCase-Notation folgen. Das muss man auch nicht! Es macht aber Sinn, bestimmte sinnvolle Regeln bei der Vergabe von Namen in Programmen und für Dateinamen zu befolgen. Da sonst (bei größeren Programmen und vielen Dateien) sehr viel Zeit für die Suche nach Fehlern oder der richtigen Datei verloren geht! Deshalb ist es sinnvoll, die camelCase-Notation von Anfang an immer zu benutzen.

#### Info:

C ist case sensitiv, das bedeutet die Variablennamen name, Na-me, NAME und naMe bezeichnen verschiedene Variablen



#### 1.7.3. Boxen mit Inhalt füllen

Wenn eine Variable deklariert ist, kann ihr während des Programmablaufes mit Hilfe des Zuweisungsoperators ("=" Einfaches Gleichheitszeichen) ein Wert zugewiesen werden. Das Gleichheitszeichen "=" hat die Aufgabe den Inhalt rechts, in die links stehend Box zu legen. Das Gleichheitszeichen hat in C eine andere Bedeutung als in der Mathematik! Besser wäre die Verwendung eines Pfeils "< -", leider hat man sich für das "=" Zeichen entschieden.

#### Info:

Das Gleichheitszeichen ist in C ein Zuweisungsoperator!

---

```
1 intVar = 7;          ○ Weist der Variablen intVar den Wert 7 zu.  
2  
3 inputVariable2 = analogRead(A2); ○ Weist der Variablen intVar2 den  
Wert des analogen PIN A2 zu.
```

**Listing 1.6:** Verwenden von Variablen

Sobald eine Variable gesetzt wurde (ein Wert zugewiesen), kann mit ihrem Wert gearbeitet werden. Z.B. kann überprüft werden, dass bestimmte Bedingungen erfüllt sind, oder der Wert kann auch direkt verwendet werden.

#### 1.7.4. Für jeden Inhalt die passende Box

VariablenTyp	Bedeutung	Beschreibung
int	Integer	ganze Zahlen (-32.768 bis 32.767)
long	ganze Zahlen	(-2 Milliarden bis 2 Milliarden)
float	Fließkommazahl	gebrochene Zahlen
char	Character	Alphanumerische Zeichen (Buchstaben, Zahlen, Sonderzeichen)
array	Variablenfeld	mehrere Werte eines VariablenTyps können gespeichert werden
String	Zeichenkette	Array aus mehreren alphanumerischen Zeichen

**Tabelle 1.2.:** Auswahl an verschiedenen VariablenTypen

Die beiden VariablenTypen array und String sind etwas komplizierter aufgebaut und werden deshalb hier noch näher besprochen:

---

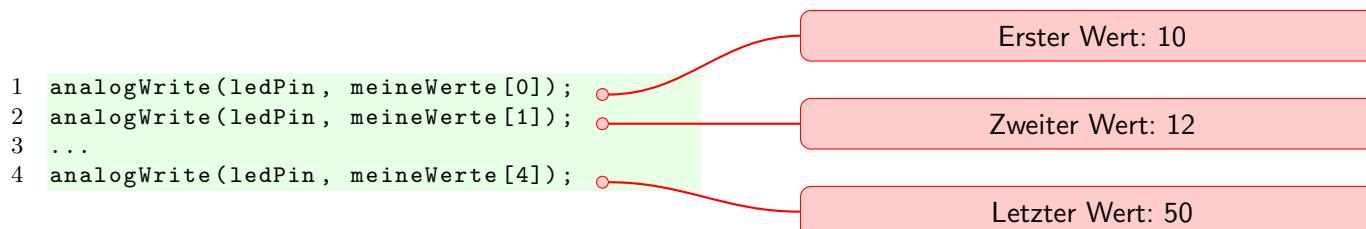
## Arrays

Bei Arrays handelt es sich im Grunde nicht um einen eigenen Typ von Variablen, sondern um eine Gruppierung mehrerer Variablen eines Typs.

```
1 int meineWerte [5] = {10,12,32,46,50};
```

Im Beispiel wird als erstes ein Array vom Typ int angelegt. Die 5 in eckigen Klammern hinter dem Namen der Variable bestimmt die Anzahl der Speicherplätze, die das Array bereit stellt. Man nennt die Anzahl der Speicherplätze auch die Länge des Arrays.

Im Programm kann man durch die Verwendung eines Index auf die Speicherplätze des Arrays zugreifen. Die erste Stelle im Array ist die Stelle 0: `meineWerte[0]`, der Wert ist 10 usw.



## String

Es gibt verschiedene Möglichkeiten Textstrings darzustellen. Man kann entweder den Datentyp String verwenden oder einen String aus einem Array von Daten des Types char erstellen.

```
1 String str1 = "arduino";
2 char str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
3 char str3[] = "arduino";
4 char Str4[8] = "arduino";
```

Alle erzeugten Textstrings haben denselben Inhalt.

---

## Aufgabe 1

Es gibt weitere Variablen-Typen. Informiere dich auf der Arduino Homepage [www.arduino.cc/en/Reference/HomePage](http://www.arduino.cc/en/Reference/HomePage) über folgende Typen: boolean, unsigned int, short, double. Notiere ihre Bedeutung, ihre Beschreibung analog zur Tabelle 1.2.

## Aufgabe 2 (Projekt)

- `double`
  - `string` - char array
  - `String` - object
  - `array`
- Conversion**

**Abbildung 1.17.:** Aufzug Arduino Homepage

Wie dir vielleicht aufgefallen ist, steht auf der Arduino Homepage [www.arduino.cc/en/Reference/HomePage](http://www.arduino.cc/en/Reference/HomePage) zweimal der Begriff String. Einmal klein und einmal groß geschrieben (siehe Abb. 1.17). Das ist kein Fehler, sondern es sind wirklich zwei unterschiedliche Dinge.

**string** bezeichnet ein Array, bei dem in jedem Feld Daten des Typs char gespeichert sind.

**String**: wiederum ist ein Text-Objekt. Objekt bedeutet in diesem Zusammenhang, dass es spezielle Funktionen gibt, mit welchen die im Objekt gespeicherten Daten manipuliert werden können. Funktionen die zu einem Objekt gehören, werden Methoden genannt.

Objekte sind in der Informatik extrem wichtig. Mit dieser Aufgabe hast du die Möglichkeit einen ersten Blick auf sie zu werfen.

Auf der Seite [www.arduino.cc/en/Reference/StringObject](http://www.arduino.cc/en/Reference/StringObject) werden die speziellen Funktionen anhand von Beispielen erklärt. Arbeitet die zwei Punkte durch. Lade dazu den jeweiligen Beispiel-Sketch auf deinen Arduino und versuche zu verstehen was passiert. Beschreibe schriftlich in kurzen Sätzen deine Beobachtungen.

- [www.arduino.cc/en/Tutorial/StringConstructors](http://www.arduino.cc/en/Tutorial/StringConstructors)
- [www.arduino.cc/en/Tutorial/StringAdditionOperator](http://www.arduino.cc/en/Tutorial/StringAdditionOperator)

### 1.7.5. Geltungsbereich von Variablen

Für eine Variable wird immer (ihrem Typ entsprechend) eine bestimmte Größe an Speicherplatz reserviert. Die Größe dieses Speicherplatzes bestimmt den Wertebereich der Variablen.

Bei einer integer Variable ist der Wertebereich alle ganzen Zahlen zwischen –32768 und 32767. Wenn man nun versucht einen größeren Wert z.B. 32770 in einer integer Variablen zu speichern wird eine sogenannte Wertebereich-Überschreitung (oder Überlauf) erfolgen. Beim Arduino wird um den zu großen Wert zu speichern einfach zum negativen Bereich "weitergegangen". Das heißt es wird der Wert –32765 gespeichert.

---

```
1 gBer = -32768;
2 gBer = gBer - 1;    ○ Wert der Variable ist jetzt 32767
3
4 gBer = 32767;
5 gBer = gBer + 1;    ○ Wert der Variable ist jetzt -32768
```

**Listing 1.7:** Überlauf einer integer Variablen

### Aufgabe 3

Verändere **nichts** an deiner Schaltung. Erstelle den Sketch "Ueberlauf". Die Aufgabe ist, den Überlauf der integer Variablen **gBer** veranschaulichen. Führe dazu die Operationen aus Listing 1.7 in die `setup()`-Methode ein und übermittle mit Hilfe des Befehls `Serial.println(gBer)` den aktuellen Wert der Variablen `gBer`.

```
1 int gBer;
2
3 void setup() {
4   Serial.begin(9600);
5   ○ Füge die einzelnen Opera-
6   Serial.println(gBer);  ○ tionen aus Listing 1.7 ein
7 }
8
9 void loop() {
10}
```

Füge die einzelnen Operationen aus Listing 1.7 ein  
Gibt nach jeder Operation den aktuellen Wert der Variablen aus

**Listing 1.8:** Versuche zur Überlauf einer Variablen

Lade deinen fertigen Sketch auf das Arduino-Board und öffne die Serielle Verbindung zum Arduino-Board. Dokumentiere deine Ergebnisse.

# 2 — Sensoren

We can only see a short distance ahead, but we can  
see plenty there that needs to be done.

---

Alan Turing (1912-1954) britischer Logiker,  
Mathematiker und Kryptoanalytiker

## Was du in diesem Kapitel lernen wirst

- Funktionsweise von analogen und digitalen Sensoren
- Verwendung von Unterprogrammen und Kontrollstrukturen in C
- Funktionsweise eines Spannungsteilers
- Messen von Temperatur, Entfernung, Licht und Farben, Luftfeuchte
- Verwendung eines Multimeters
- Das Verwenden von Objekten und Methoden

## 2.1. Analoger Sensor: Der Temperatur-Sensor LM35 und LM36

Die Temperatursensoren LM35 und LM36 oder die Funktionsgleichen TMP35 TMP36 sind sogenannte analoge Sensoren. Analoge Sensoren haben meistens 3 PINs. Zwei PINs werden zur Spannungsversorgung ( $U_S$  und GND) und einer PIN ( $V_{out}$ ) für den Messwert benötigt. Dabei variiert die Spannung an  $V_{out}$  linear mit 10mV pro  $1^{\circ}\text{C}$ <sup>1</sup>. Beim LM35 entspricht einer Spannung von 0mV dem Temperaturwert von  $0^{\circ}\text{C}$ . Da mit dem TMP36 auch negative Temperaturen gemessen werden können entspricht eine Spannung von 0mV ca. dem Temperaturwert  $-50^{\circ}\text{C}$  (siehe Abb. 2.1).

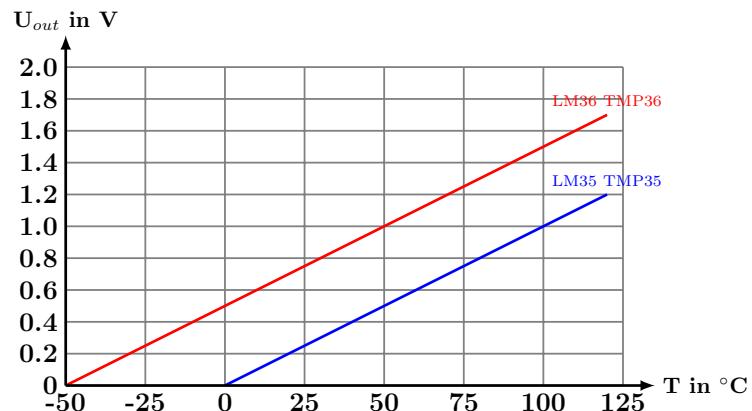


Abbildung 2.1.: Temperatur Ausgangsspannungs Verlauf des LM35

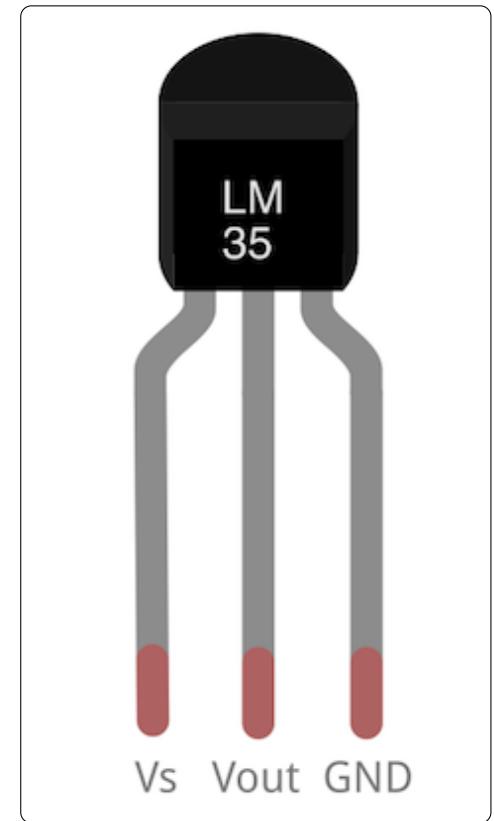


Abbildung 2.2.: PIN Belegung

Solche Informationen erhält man aus einem sogenannten Datenblatt. Jeder Hersteller liefert ein mehr oder weniger ausführliches Datenblatt, das den Aufbau und die Funktion des Bauteils beschreibt. Für die Temperatursensoren kann anhand dieser Informationen eine Formel zur Berechnung des Temperaturwerts hergeleitet werden. Für den LM35 und TMP35 ergibt sich die Formel (2.1) und für den LM36 und TMP36 die Formel (2.2):

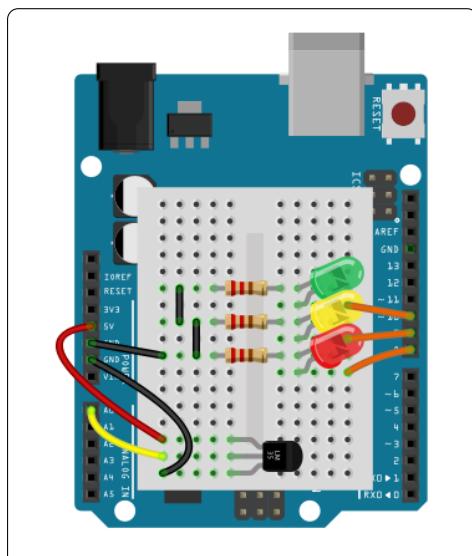
<sup>1</sup>Eigentlich werden Temperaturunterschiede in Grad Kelvin K angegeben.

$$T(V_{out}) = \frac{V_{out}}{10} \quad (2.1)$$

$$T(V_{out}) = \frac{V_{out} - 500}{10} \quad (2.2)$$

### 2.1.1. Der Beispiel-Sketch

Wenn zur Messung der Temperatur eine Verarbeitung der gemessenen Werte am analogen Eingang nötig sind, ist es für die Übersicht des Sketches sinnvoll, diese Berechnung in einer sog. Funktion auszulagern. Der Vorteil ist, dass wenn man den Sensor gegen einen anderen Sensor austauscht, bei dem die Messgröße anhand einer anderen Formel berechnet werden muss, dann findet die Anpassung nur in dieser Funktion statt.



```

1 int senPin = A0;          ○ Definition des Sensor PINs
2 int VS = 5000;            ○ Versorgungsspannung in mV
3 float temp = 0.0;
4
5 void setup() {
6   Serial.begin(9600);
7 }
8
9 void loop() {
10   temp = getTemperature(); ○ Die Funktion wird aufgerufen
11   Serial.print( temp );
12   Serial.println(" °C");
13 }
14
15 float getTemperature() {
16   int senV = analogRead(senPin);
17   float t = map(senV,0,1023,0,VS)/10.0; ○ Definition der Funktion für einen LM36
18   return t;                         muss diese Zeile abgeändert werden:
19 }                                     float temp =
                                            (map(senV,0,1023,0,VS)-500)/10;

```

**Listing 2.1:** Testsketch "TempTest" für einen Temperatursensor LM35

Abbildung 2.3.: Temperatursensor LM35CZ

---

## 2.1.2. Aufgaben

Baue die abgebildete Schaltung (Abb. 2.3) auf. Achte auf die Beschriftung und somit auf den richtigen Einbau des Temperatur-Sensors. Sollte sich der Temperatursensor, nachdem du den Arduino mit dem PC verbunden hast sehr schnell erwärmen, dann hast du den Sensor falsch eingebaut! Unterbreche die Verbindung zum PC und überprüfe den richtigen Einbau. Wenn der Temperatursenor sich zu sehr erwärmt, kann er zerstört werden!

### Aufgabe 1

Erstelle und lade den Beispiel-Sketch "TempTest" 2.1 auf deinen Arduino und öffne dann die serielle Verbindung. Messe verschiedene Temperaturen.

### Aufgabe 2

Baue mit Hilfe des Temperatursensor und den drei LEDs (grün, gelb und rot) ein Thermometer bei dem bis zur Temperatur 20°C nur die grüne LED leuchtet, bis 25°C die grüne und gelbe LED leuchten und ab 30°C die grüne, gelbe und rote LED leuchten. Füge dazu den Code in die loop()-Routine ein und ergänze entsprechend. In Kapitel 2.5.1 erfährst du mehr über bedingte Anweisungen.

```
1 if (temp > 20) {  
2     // gruen an, gelb rot aus  
3 }  
4 if (temp > 25) {  
5     // gruen gelb an, rot aus  
6 }  
7 if (temp > 30) {  
8     // gruen gelb rot an  
9 }
```

#### Info:

if ist eine bedingte Anweisung  
(siehe Kapitel 2.5.1)

### Aufgabe 3 (Projekt)

In Amerika werden normalerweise Temperaturen in Fahrenheit  $F$  angegeben. Die Umrechnung ist einfach:

$$F(T) = T \cdot 1.8 + 32 \quad (2.3)$$

Schreibe eine weiter Funktion `getFahrenheit()`, die die gemessene Temperatur in Fahrenheit berechnet. Übermittel mit Hilfe einer seriellen Verbindung den gemessenen Temperaturwert in Grad Celsius und Fahrenheit an den PC.

---

Die wichtigste Temperaturskala ist die Kelvin-Skala. Bei der Kelvin Skale entspricht die Temperatur am absoluten Nullpunkt 0K und bei 0°C umgerechnet 273,15K.

Es gibt noch weitere Temperaturskalen. Suche in Wikipedia nach dem Schlagwort "Temperaturskala". Entwerfe weiter Unterprogramme für die Temperaturskalen: Kelvin und Rømer

```
1 float GetFahrenheit(int vs) {  
2     int senV = analogRead(senPin);  
3     float temp = (map(senV,0,1023,0,vs)-500)  
        /10;  
4     return fahrenheit;  
5 }
```

Umrechnung von Grad Celsius nach Fahrenheit

## 2.2. C Sprachelemente: Unterprogramme

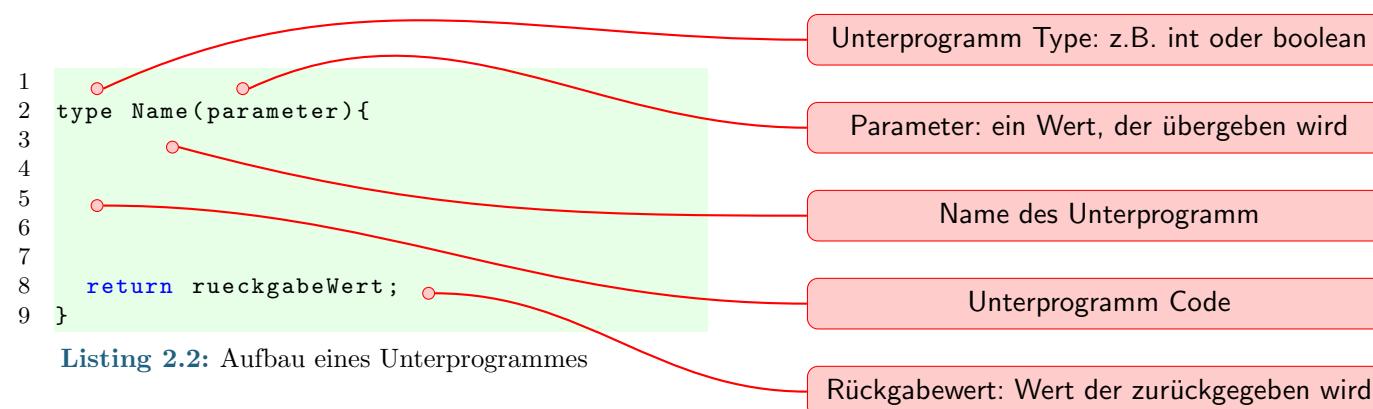
Der Einsatz eines Unterprogramm ist sinnvoll, um das Programm übersichtlich zu halten. Wenn z.B. das Auslesen und Berechnen einer Messgröße eines Sensors aufwendig ist, wird der Sketch übersichtlicher, wenn dieser Teil des Programms aus der `loop()`-Routine in ein eigenes Unterprogramm ausgelagert wird. Unterprogramme werden dann unverzichtbar, wenn einzelne Programmteile sich wiederholen. Wenn diese Teile nicht in ein Unterprogramm ausgelagert werden und später Änderungen vorgenommen werden, dann müssen diese Änderungen an verschiedenen Stellen vorgenommen werden. Wenn ein Unterprogramm benutzt wird, muss nur an einer einzigen Stelle etwas verändert werden!

**Info:**

Don't repeat yourself – DRY

### 2.2.1. Allgemeiner Aufbau eines Unterprogrammes in C

Ein Unterprogramm besteht aus zwei Teilen: einem Kopf und einen Körper. Im Kopf wird der Typ, der Name und eventuelle Eingabeparameter festgelegt. Der Körper besteht aus den eigentlichen Anweisungen und einem eventuellen Rückgabewert.



Unterprogramme können anhand ihres Types unterschieden werden. So kann man Unterprogramme ohne Rückgabewert Prozeduren und Unterprogramme mit Rückgabewert Funktionen nennen.

**Info:**

In vielen Programmiersprachen werden die Begriffe Unterprogramm und Funktion synonym verwendet.

## 2.2.2. Unterprogramm ohne Rückgabewert – Prozeduren

Unterprogramme die keinen Rückgabewert haben, werden Prozeduren genannt. Du solltest zwei Prozeduren schon gut kennen: Die `setup()`- und `loop()`-Prozeduren. Das Type-Schlüsselwort ist `void` (eng. für leer).

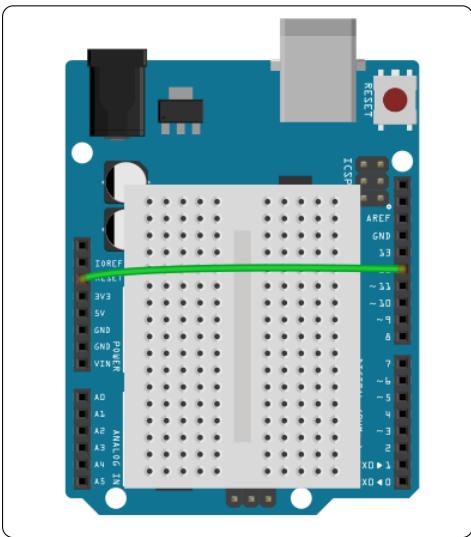


Abbildung 2.4.: Reset Prozedure

```
1 void Reset()
2 {
3     int resetPin=12;
4     pinMode(resetPin, OUTPUT);
5     digitalWrite(resetPin, HIGH);
6 }
7 }
```

Listing 2.3: Reset()-Prozedur

**void** bedeutet, dass die Prozedur keine Rückgabewert hat

Name des Unterprogramms

Der Arduino wird mit Hilfe des Reset-PINs und PIN 12 neu gestartet

Es gibt Lösungen ohne die Verwendung des Reset-PINs. Wenn man mit den Schlagwörtern Watchdog und Reset Function sucht, findet man schnell solche Lösungen. Aber es ist Vorsicht geboten, wenn eine solche Reset-Prozedure zu schnell aufgerufen wird, startet der Arduino neue bevor der Bootloader einen neuen Sketch aufspielen kann. Damit wird der IC unbrauchbar und muss ausgetauscht werden.

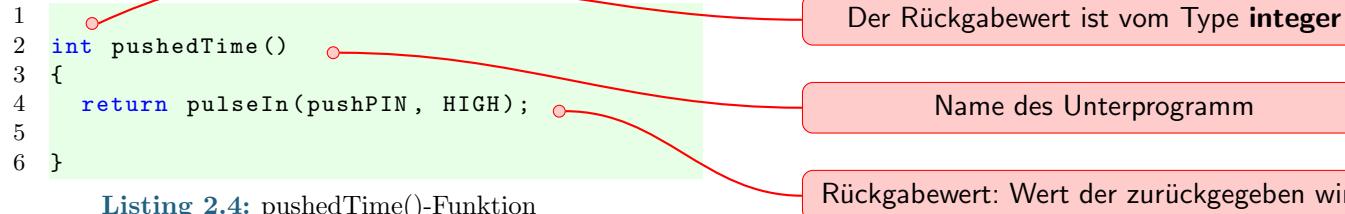
---

### 2.2.3. Unterprogramm mit Rückgabewert – Funktion

Unterprogramme werden als Funktion bezeichnet, wenn sie einen Wert zurückliefern.

#### Beispiel:

Oft wird die Zeit benötigt, die ein Taster gedrückt wird. Es gibt zwar einen Arduino Befehl dafür (`pulseIn()`), aber durch die Verwendung einer Funktion wird der Sketch übersichtlicher.



```
1 int pushedTime()
2 {
3     return pulseIn(pushPIN, HIGH);
4 }
5 }
```

Listing 2.4: `pushedTime()`-Funktion

Der Rückgabewert ist vom Type **integer**

Name des Unterprogramm

Rückgabewert: Wert der zurückgegeben wird

### 2.2.4. Unterprogramm mit Parameter

Parameter können an das Unterprogramm übermittelt werden, so ist es möglich, dass Informationen von Hauptprogramm zum Unterprogramm übermittelt werden.

---

```

1 void blinkLED(int ledPIN)
2 {
3     pinMode(ledPIN, OUTPUT);
4     digitalWrite(ledPIN, HIGH);
5     delay(1000);
6     digitalWrite(ledPIN, LOW);
7     delay(1000);
8 }

```

void also keine Rückgabewert  
Name des Unterprogramm  
Parameter: digitaler PIN  
Eine LED am blinkt mit einer Sekunde

**Listing 2.5:** blinkLED()-Parameter

## 2.2.5. Aufgaben:

### Aufgabe 1:

Du hast jetzt schon einige Arduino Befehle kennengelernt und in deiner Befehlsübersicht dokumentiert! Ordne die Befehle, die Unterprogramme sind in einer Tabelle. Unterscheide die Befehle nach Prozedur, Funktion und Unterprogramme mit Parameter. Einzelne Befehle können auch in mehrmals auftauchen. In Abb. 2.5 ein.

Unterprogramm Typ:	Arduino Befehle
Prozeduren	setup()
Funktion	digitalRead(PIN)
Unterprogramm mit Parameter	delay(1000)

**Abbildung 2.5.:** Arduino Befehle sind Unterprogramme

---

## Aufgabe 2: Der SOS-Sketch mit Unterprogrammen

In Kapitel 1 hast du den Sketch "SOS" geschrieben, der mit Hilfe einer LED an PIN 10 den Morsecode SOS (kurz kurz kurz, lang lang lang, kurz kurz kurz) optisch sendet. Öffne diesen Sketch und analysiere ihn. Zwei Sachverhalte wirst du wahrscheinlich feststellen:

1. Du wirst Schwierigkeiten haben deinen Sketch schnell zu verstehen!
2. Wenn du die Zeiten für die lange Leuchtdauer verdoppeln möchtest, dann muss du das an sehr vielen Stellen machen.

Der Sketch ist unübersichtlich, da sehr oft hintereinander sehr ähnlicher Code steht. Mit Hilfe der Prozeduren lang() und kurz() kann die loop()-Methode vereinfacht werden. Füge den Beispielcode aus Listing 2.6 nach der loop()-Methode in den Sketch ein und vervollständige die Prozedur kurz(). Ersetzt in deiner loop()-Methode den jeweiligen Code durch den Aufruf der Prozedur kurz() oder lang() (siehe Listing 2.6).

```
1 void lang() {
2     // led soll 2 Sekunden leuchten
3     digitalWrite(10, HIGH);
4     delay(2000);
5     digitalWrite(10, LOW);
6     delay(1000);
7 }
8
9 void kurz() {
10    // led soll 1 Sekunden leuchten
11 }
```

**Listing 2.6:** Die Prozeduren lang() und kurz()

```
1 void blink(int time) {
2     digitalWrite(10, HIGH);
3     delay(time);
4     digitalWrite(10, LOW);
5     delay(1000);
6 }
```

**Listing 2.7:** Die Prozeduren blink(int time)

Jetzt sollte deine loop()-Prozedur schon viel einfacher aussehen. Aber eigentlich machen deine beiden Prozeduren, mehr oder weniger das selbe. Erstelle jetzt eine Prozedur blink() mit dem Parameter time (siehe Listing 2.7). Vereinfache deinen Sketch weiter und dokumentier ihn ausführlich.

## 2.3. Widerstände und Spannungsteiler

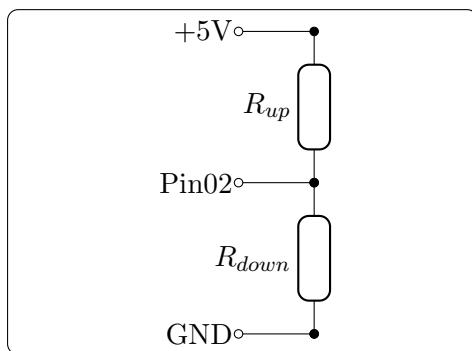


Abbildung 2.6.: Spannungsteiler

Würde man einen digitalen Pin mit 5 Volt verbinden, so läge (wenn der digitale Pin auf INPUT gestellt ist) der Zustand "HIGH" an. Auf der anderen Seite, den nun GND mit dem digitalen Pin verbunden wäre, so läge der Zustand "LOW" an. Was wäre, wenn man den digitalen Pin sowohl mit 5 Volt und mit GND verbindet? Ist der anliegende Zustand dann "HIGH" oder "LOW"? Leider würde es einen Kurzschluss geben, da 5 Volt direkt mit GND verbunden wäre, und kein Strombegrenzer (LED mit Vorwiderstand, Lautsprecher, Motor oder ähnliches) vorhanden ist, steigt der Strom soweit an, bis er über den maximal zulässige Strom von 500mA liegt. Der Spannungsregler wird überhitzen und sich hoffentlich rechtzeitig abschalten, bevor etwas kaputt geht.

Interessanter wird es, wenn man die beiden Verbindungen vom digitalen Pin zu GND bzw. vom digitalen Pin zu 5V jeweils mit Widerständen verbindet. So entsteht kein Kurzschluss, da nur ein schwacher Strom durch die beiden Widerstände am Mikrocontroller vorbei fließt. Diese Schaltung bezeichnet man als Spannungsteiler (siehe Abb. 2.6).

Der Pull-Up-Widerstand  $R_{up}$  und der Pull-Down-Widerstand  $R_{down}$  teilen sich die Gesamtspannung entsprechend ihrer Anteile am Gesamtwiderstand in die Spannung  $U_{up}$  und  $U_{down}$  auf. Das Potential am digitalen Pin des Arduinos entspricht  $U_{down}$  und lässt sich aus den Widerstandswerten und der Gesamtspannung  $U_{ges}$  berechnen:

$$U_{down} = U_{ges} \cdot \frac{R_{down}}{R_{up} + R_{down}} \quad (2.4)$$

### 2.3.1. Die Funktionsweise der Push-Button-Schaltung

In Kapitel 1 hast du einen Push-Button schon kennengelernt und eingesetzt (siehe Abb. 2.7). Ein Pushbutton kann als analoger Sensor aufgefasst werden, der wenn er nicht eingedrückt wird einen sehr hohen Widerstand besitzt, wenn er aber gedrückt wird, dann besitzt er einen sehr kleinen Widerstand. D.h. dass es mit einer Widerstandsmessung möglich ist auf einen Unterschied zu reagieren. Wenn nun den Widerstand  $R_{up}$  durch ein Stück Draht ersetzt, liegt am Pin02 immer der Zustand "HIGH". Wenn dieser Draht entfernt wird liegt der Zustand "LOW" an Pin02 an. Genau dies leistet ein Push-Button:

- Push-Button ist nicht gedrückt, d.h. am digitalen Pin liegt der logische Zustand "LOW", da der Arduino intern einen Pull-Down Widerstand eingebaut hat.
- Push-Button ist gedrückt, so liegt am digitalen Pin die Spannung  $U_{down}$  diese ist gleich 5 Volt. Am digitalen Pin liegt der logische Zustand "HIGH".

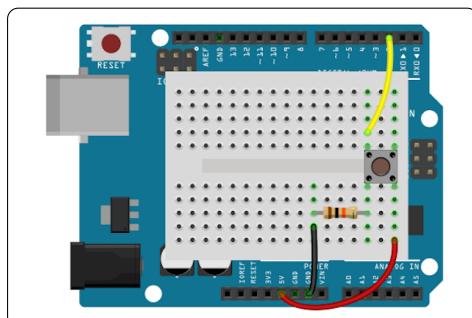


Abbildung 2.7.: Push-Button als Spannungsteiler

---

## 2.3.2. Aufgaben

Diese Aufgaben dienen zur Vorbereitung für die Verwendung des Spannungsteilers beim Messen mit analogen Sensoren.

### Aufgabe 1

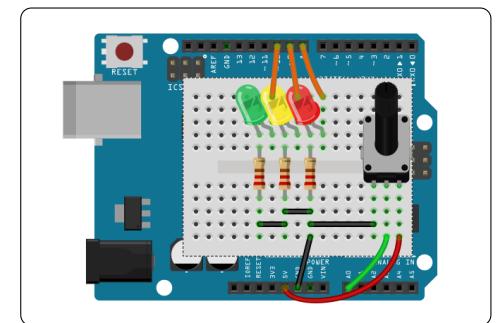
Zeichne ein Schaltbild der Pushbutton-Schaltung, die in Abbildung 2.7 zu sehen ist. Messe mit Hilfe eines Multimeters, die Spannungen (gedrückt und nicht gedrückt), die am digitalen PIN anliegen. Dazu musst du die Schaltung eventuell nochmals aufbauen.

### Aufgabe 2

Baue deine Schaltung aus Aufgabe 1 so um, dass der Zustand "HIGH" am digitalen Pin02 anliegt, wenn der Pushbutton nicht gedrückt wird und "LOW", wenn der Pushbutton gedrückt wird.

### Aufgabe 3 (Projekt)

Baue deine Schaltung mit Hilfe des Abb. 2.8 um. Ändere anschließend den Beispielsketch "Blink", so ab, dass die gelbe LED mit Hilfe des Potentiometer gedimmt werden kann.



**Abbildung 2.8.:** Das Potentiometer ist ein Spannungsteiler

## 2.4. Lichtsensoren

Als Lichtsensor (oder optischer Detektor, optoelektronischer Sensor, Photodetektor) werden elektronische Bauelemente bezeichnet, die Licht unter Benutzung des photoelektrischen Effekts in ein elektrisches Signal umwandeln oder einen von der einfallenden Strahlung abhängigen elektrischen Widerstand zeigen.

### 2.4.1. LDR, ein lichtempfindlicher Widerstand

Ein LDR (eng light dependent resistor) ist ein Fotowiderstand, d.h. der Widerstand eines LDR ändert sich mit der Helligkeit, des auf ihn einfallenden Lichtes.

#### Funktionsweise des LDR-Sensors

Die Grundschaltung eines Helligkeitsensors auf LDR-Basis ist die einer Spannungsteilungsschaltung. Wobei der Widerstand  $R_{up}$  durch den LDR ersetzt wird (siehe Abb. 2.6 und Abb. 2.9). Den Widerstand  $R_{down}$  wählt man in etwa so groß wie der Widerstand des LDR im abgedunkelten Zustand.

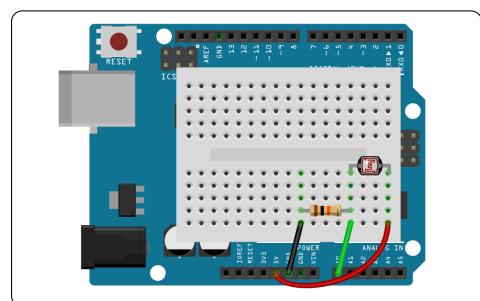


Abbildung 2.9.: Aufbau der Schaltung auf dem BreadBoard

### 2.4.2. Beispiel Sketch

```
1 int ldrPin = A0;          Analog Pin A0 wird verwendet
2
3 void setup(){
4     Serial.begin(9600);
5 }
6
7 void loop(){
8     int ldrWert = analogRead(ldrPin);  Spannung am LDR wird gemessen
9
10    Serial.println(ldrWert);
11    delay(250);                      Kleine Pause, da das Auslesen etwas dauert!
12 }
```

Listing 2.8: Testsketch "ldrTest" für einen LDR

---

Dieser Sketch misst die Spannung in den Werten 0 bis 1024 (0V bis 5V) am analogen Pin A0 und übermittelt den gemessenen Wert über eine serielle Verbindung an den PC.

### 2.4.3. Aufgaben

#### Aufgabe 1:

Nimm den LDR in die Hand und messe seinen Widerstand abgedunkelt und in direktem Licht. Verwende dazu die Widerstands-Messfunktion des Multimeters. Notiere deine Messwerte. Du benötigst diese Messwerte für die weiteren Aufgaben!

#### Aufgabe 2:

Baue nun eine Spannungsteilerschaltung auf (siehe Abb. 2.9), wobei du für den zweiten Widerstand einen Normwiderstand einbaust, der möglichst nahe an den gemessenen LDR-Widerstand herankommt. Welchen Werte misst du nun an A0? Benutze den "Serial Monitor", um sie am PC auszugeben. Berechne mit Hilfe der Gleichung des Spannungsteilers 2.4 den Widerstand des LDR und vergleiche mit deinen gemessenen Werten aus Aufgabe 1.

#### Aufgabe 3: (schwer!)

Benutze die Methode **map()**, um mit Hilfe der gemessenen Werte den gesuchten Widerstand zu berechnen.

#### Aufgabe 4: Projekt

In der Abb. 2.10 siehst du eine mögliche Anwendung für einen LDR. Die Funktion dieser Schaltung ist folgende: Wenn das auf den LDR fallende Licht unter einer bestimmten Intensität fällt, dann soll die eine LED ausgehen und die andere angehen. Baue die Schaltung auf und schreibe einen entsprechenden Sketch. Benutze dazu die Kontrollstruktur if:

```
1 if (analogRead(ldrPin)< wert) {  
2     digitalWrite(ledPin1,HIGH);  
3 }  
4 else {  
5     digitalWrite(ledPin2,LOW);  
6 }
```

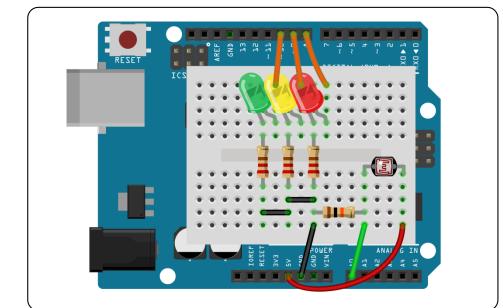


Abbildung 2.10.: Lichtautomatik

#### Info:

In Kapitel 2.5 erfährst du mehr über Kontrollstrukturen

## 2.5. C Sprachelemente: Kontrollstrukturen

Kontrollstrukturen werden verwendet, um den Ablauf eines Computerprogramms zu steuern. Eine Kontrollstruktur ist entweder eine bedingte Anweisung, eine Verzweigung oder eine Schleife. Hier soll zunächst nur die bedingte Anweisung und Verzweigung besprochen werden. Eine bedingte Anweisung oder eine Verzweigung wird über einen logischen Ausdruck, der die Werte true oder false besitzen kann gesteuert. Kontrollstrukturen werden mit Hilfe eines Diagramms, dem sogenannten Programm-Ablauf-Diagramms (PAD) in der Literatur visualisiert.

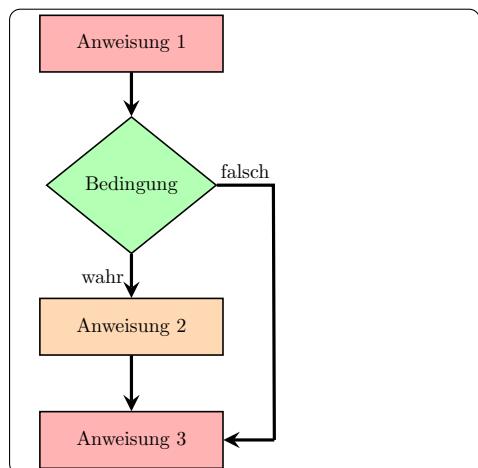
Bei einem Mikrocontroller ermöglichen Entscheidungen, z.B. auf ein Messergebnis eines Sensors entsprechend zu reagieren.

### 2.5.1. Kontrollstruktur – die bedingte Anweisung: if-Anweisung

Die sog. if-Anweisung ist eine bedingte Anweisung und stellt die einfachste Kontrollstruktur für den Programmablauf dar. Sie ermöglicht, dass eine Anweisung nur dann ausgeführt wird, wenn eine Bedingung erfüllt ist.

#### Nomenklatur

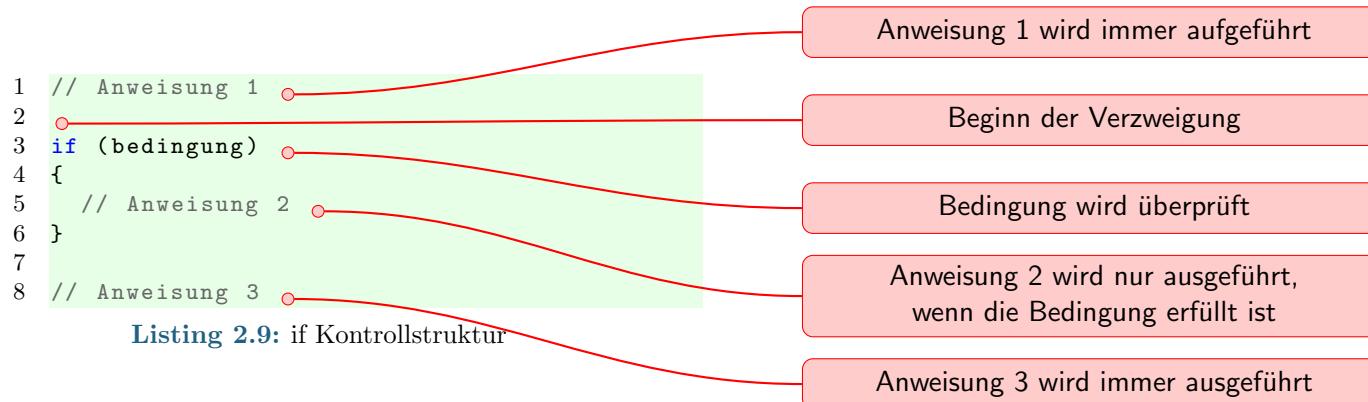
Eine bedingte Anweisung besteht aus zwei Teilen, dem Kopf mit dem Schlüsselwort if und einer Bedingung und dem Körper, der bei erfüllter Bedingung ausgeführt werden soll und die Anweisungen enthält.



#### Ablauf der if-Kontrollstruktur

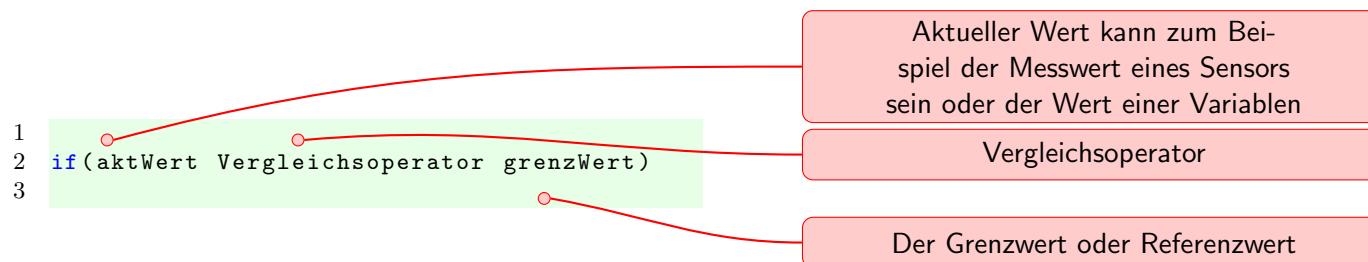
Beim verarbeiten der if-Bedingung in Listing 2.9 werden die Anweisungen 1 und 3 vor und nach der if-Bedingung immer ausgeführt. Die Anweisung 2 wird nur ausgeführt, wenn die Bedingung logisch wahr ist (siehe Abb. 2.11).

Abbildung 2.11.: PAD einer bedingten Anweisung



## Aufbau einer Bedingung

Meistens besteht die Bedingung aus einer Messgröße und einen Grenzwert, wird dieser Grenzwert z.B. unter oder überschritten, dann soll der Körper der if-Bedingung ausgeführt werden. Möglich ist aber auch ein Arduino Befehl oder eine Funktion vom Typ boolean.



## Info:

Achtung: das Gleichheitszeichen „`=`“ wird schon als Zuweisungsoperator für Variablen verwendet!

In der Tabelle 2.12 sind alle möglichen Vergleichsoperatoren aufgelistet. Vorsicht ist beim Vergleichsoperator `==` geboten. Dieser sollte nur bei ganzzahligen Vergleichswerten benutzt werden. Bei Kommazahlen müssen die Werte einander exakt entsprechen. In diesem Fall muss ein anderer Vergleichsoperator verwendet werden.

Operator	Erklärung
<code>x == y</code>	$x$ ist gleich wie $y$
<code>x != y</code>	$x$ ist nicht gleich wie $y$
<code>x &lt; y</code>	$x$ ist kleiner als $y$
<code>x &gt; y</code>	$x$ ist größer als $y$
<code>x &lt;= y</code>	$x$ ist kleiner oder gleich groß wie $y$
<code>x &gt;= y</code>	$x$ ist größer oder gleich groß wie $y$

Abbildung 2.12.: Die Vergleichsoperatoren

**Funktionen mit boolschem Rückgabewert:** Es gibt Funktionen und Arduino-Befehle, die als Rückgabewert ein Wahrheitszeichen (“true” oder “false”) liefern. Ein dieser Befehle ist zum Beispiel `digitalRead()`. Dadurch ist es möglich, dass auf das drücken eines Push-Buttons reagiert werden kann.

```
1  
2 if (digitalRead(senPin))  
3
```

Die Bedingung, ist hier der boolsche Rückgabewert der Funktion `digitalRead()`

---

## Aufgabe 1

Für diese Aufgabe musst du die Schaltung nicht verändern. Schreibe einen Sketch, der über die serielle Schnittstelle mit dem PC kommuniziert. Als Basis für die Loop-Routine kannst du Listing 2.10 benutzen, du musst den Sketch natürlich von vervollständigen!

Füge mindestens zwei weitere if-Anweisungen mit verschiedenen Vergleichsoperatoren ein. Dokumentiere die Lösungen!

## Aufgabe 2

Für diese Aufgabe brauchst du nur das Arduino Board und Kabel. Schreibe einen Sketch, der über die serielle Schnittstelle mit dem PC kommuniziert. Als Basis für die Loop-Routine kannst du Listing 2.11 benutzen, du musst den Sketch natürlich noch vervollständigen!

Verbinde nun den digitalen PIN 2 wahlweise mit 5V und GND und überprüfe mit der seriellen Konsole das Ergebnis.

```
1 int var = 0;
2 ...
3 void loop() {
4   if (var < 10) {
5     Serial.println(var ist kleiner 10);
6   }
7   if (var == 5) {
8     Serial.println(var ist gleich 5);
9   }
10  ...
11
12  var = var + 1;
13  delay(500);
14  if (var > 20) {
15    var = 0;
16    Serial.println(var hat jetzt wieder den
17                  Wert 0);
17 }
```

Listing 2.10: Vorlage für Aufgabe 1

```
1 int senPin = 2;
2 ...
3
4 void setup() {
5   pinMode(senPin, INPUT);
6   ...
7 }
8
9 void loop() {
10  if (digitalRead(senPin)) {
11    Serial.println(Der senPin ist mit 5V
12                  verbunden);
12  }
13  if (!digitalRead(senPin)) {
14    Serial.println(Der senPin ist mit GND
15                  verbunden);
15  }
16  delay(500);
17 }
```

Listing 2.11: Vorlage für Aufgabe 2

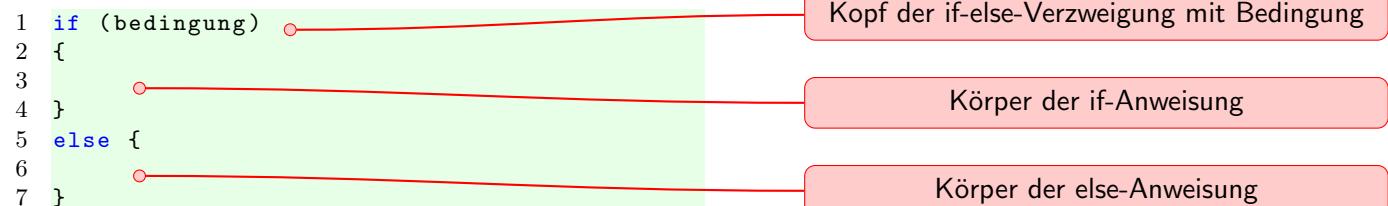
---

### 2.5.2. Kontrollstruktur – die Verzweigung: if-else-Verzweigung

Eine Verzweigung (auch Auswahl oder Selektion genannt) besteht aus einer Bedingung und zwei Codeabschnitten. Wieder wird erst die Bedingung ausgewertet, und falls sie nicht zutrifft, wird anschließend der zweite Codeabschnitt ausgeführt.

#### Nomenklatur

Eine Verzweigung besteht aus drei Teilen, dem Kopf mit dem Schlüsselwort **if** einer Bedingung und dem if-Körper, der bei erfüllter Bedingung ausgeführt werden soll und dem else-Körper, der bei nicht erfüllter Bedingung ausgeführt wird.



## Ablauf der Verzweigung

Die Bedingung hat bei einer Verzweigung dieselbe Struktur, wie bei einer bedingten Anweisung. Sie besteht meist aus einer Messgrösse und einen Grenzwert. Ist die Bedingung wahr, dann soll der Körper der if-Anweisung ausgeführt werden. Wenn die Bedingung falsch ist der Körper der else-Anweisung auszuführen.

```
1 // Anweisung 1
2
3 if (bedingung) {
4     // Anweisung 2a
5 } else {
6     // Anweisung 2b
7 }
8 // Anweisung 3
```

Anweisung 1 wird immer ausgeführt

Anfang der Verzweigung

Anweisung 2a wird bei wahrer Bedingung ausgeführt

Anweisung 2b wird bei falscher Bedingung ausgeführt

Anweisung 3 wird immer ausgeführt

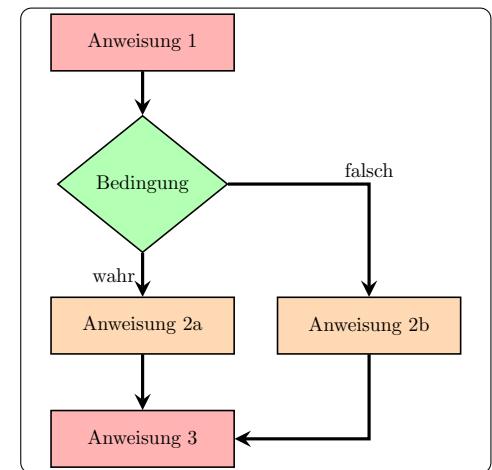


Abbildung 2.13.: PAD einer Verzweigung

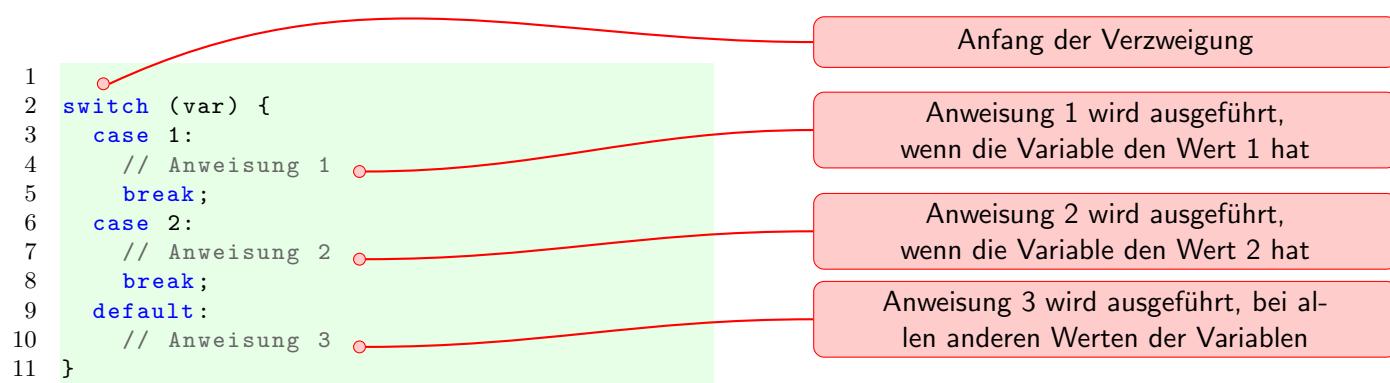
## Aufgabe 1

Ändere den Sketch, den du mit Hilfe des Listing 2.11 erstellt hast. Die zwei if-Anweisungen soll jetzt durch eine if-else-Anweisung ersetzt werden.

---

### 2.5.3. Kontrollstruktur – switch-case

Will man einen Wert auf verschiedene Zustände prüfen, bietet sich die switch-case-Abfrage an. Diese Kontrollstruktur, kann z.B. verwendet werden, wenn mit Hilfe einer seriellen Verbindung Zeichen an den Arduino übermittelt werden oder auf die Werte eines Sensors reagiert werden soll.



---

## Aufgabe 1:

Mit Hilfe des switch-case Kontrollstruktur kann ein sogenanntes "Human Interface" realisiert werden. Es soll die Annäherung der durch die Übermittelung der Wörter "bright, medium, dim, dark" beschrieben werden.

Für dieses Aufgabe benötigst du die Schaltung mit dem LDR und verwende den folgenden Sketch:

```
1 int sMin = 0;
2 int sMax = 600;
3
4 void setup() {
5     Serial.begin(9600);
6 }
7
8 void loop() {
9     int sVal = analogRead(A0);
10    int range = map(sVal, sMin, sMax, 0, 3); ○ Umrechnung des Sensorwertes
11
12    switch (range) {
13        case 0: ○ Hand bedeckt den Sensor
14            Serial.println("dark");
15            break;
16        case 1: ○ Hand ist etwas über dem Sensor
17            Serial.println("dim");
18            break;
19        case 2: ○ Hand ist ein paar Zenti-
20            meter über dem Sensor
21            Serial.println("medium");
22            break;
23        case 3: ○ Hand ist weit vom Sensor entfernt
24            Serial.println("bright");
25            break;
26        default: ○ Es trat ein Fehler auf
27            Serial.println("error");
28    }
29    delay(1);
}
```



## 2.6. Reflexoptokoppler

Reflexoptokoppler können die Helligkeit von Flächen messen. Ein Reflexoptokoppler kann als Liniensensor, Lichtschranke oder Barcodeleser eingesetzt werden.

Eine beliebte Aufgabe für Roboter ist das Liniensuchen. Doch um die Linie erst einmal zu erkennen braucht man einen Sensor der die Unterschiede des Bodens erkennen kann. Eine häufig eingesetzte Methode ist dabei die Erkennung über IR-Licht. Man braucht dazu lediglich eine IR-Diode zur Beleuchtung des Bodens und eine Photodiode oder einen LDR zur Messung des zurückgeworfenen Lichts.

Jede Lichtschranke besteht aus einem Sender und einem Empfänger. Das ausgesendete Licht muss also reflektiert oder zurückgestreut werden, damit es den Empfänger trifft. Dazu genügt ein helles Objekt, das sich in wenigen mm Abstand vor der Lichtschranke befindet.

Ähnlich wie ein Temperatursensor kann eine Lichtschranke an einen analogen Pin des Arduino angeschlossen werden. In einem Reflexoptokoppler sind eine IR LED als Lichtquelle und ein Fototransistor als Empfänger in einem gemeinsamen Gehäuse verbaut. Der Fototransistor empfängt die von der zu untersuchenden Oberfläche gestreute IR-Strahlung und gibt ein Signal an den Mikrocontroller. Im Betrieb sollte die Entfernung zur streuenden Fläche zwischen 1mm und 4mm betragen. Allerdings ist der Sensor empfindlich gegen Streulicht.

Der Strom durch den Fototransistor ist von der eingehenden Strahlungsintensität abhängig. Der elektrische Widerstand des Empfängers wird durch das einfallende Infrarotlicht verkleinert.

### 2.6.1. Funktionsweise des CNY70 Reflexoptokoppler

Das Prinzip dahinter steckt ist relativ einfach. Eigentlich misst man die Reflektionseigenschaften des Untergrunds und schließt darüber auf die Helligkeit des Untergrundes. Das heißt man sendet mit einer LED Licht aus und schaut wie viel davon dann beim Empfänger ankommt. Zum Messen wird in diesem Fall ein Phototransistor benutzt. Die einfachste Methode um dieses Signal auszuwerten besteht in einer Art Spannungsteiler.

### 2.6.2. Beispiele Sketch

Der Sketch 2.12 ist etwas komplexer aufgebaut. Die Idee ist die Spannung zwischen dem  $10\text{k}\Omega$ -Widerstand und dem Lichtwiderstand mit Hilfe des analogen PINs A0 zu messen. Diese Messung wird zweimal hintereinander ausgeführt. Das erste Mal mit eingeschalteter IR-LED<sup>2</sup>, beim zweiten Mal mit ausgeschalteter IR-LED, dann werden beide Werte über

<sup>2</sup>Dazu müsst ihr eure Schaltung umbauen, damit die IR-LED ihre Spannung nicht mehr über 5V bezieht, sondern über den digitalen PIN 09

Abbildung 2.14.: CNY90 (auf der Seite mit Schrift ist der Photo Transistor)

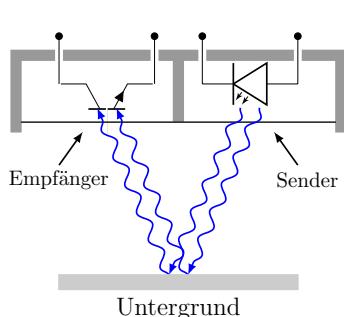
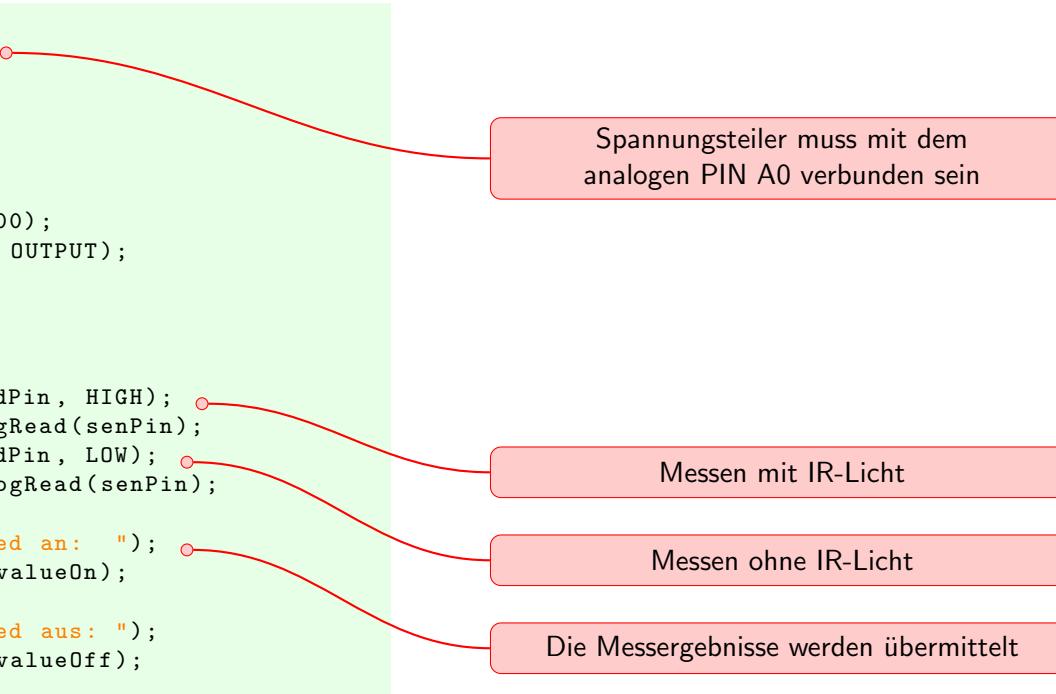


Abbildung 2.15.: Funktionsweise des CNY70

---

die Serielle Schnittstelle an den PC übertragen.

```
1 int ledPin = 5;
2 int senPin = A0; ○
3 int valueOn = 0;
4 int valueOff = 0;
5
6 void setup()
7 {
8     Serial.begin(9600);
9     pinMode(ledPin, OUTPUT);
10 }
11
12 void loop()
13 {
14     digitalWrite(ledPin, HIGH); ○
15     valueOn = analogRead(senPin);
16     digitalWrite(ledPin, LOW); ○
17     valueOff = analogRead(senPin);
18
19     Serial.print("Led an: ");
20     Serial.println(valueOn);
21
22     Serial.print("Led aus: ");
23     Serial.println(valueOff);
24 }
```



Spannungsteiler muss mit dem analogen PIN A0 verbunden sein

Messen mit IR-Licht

Messen ohne IR-Licht

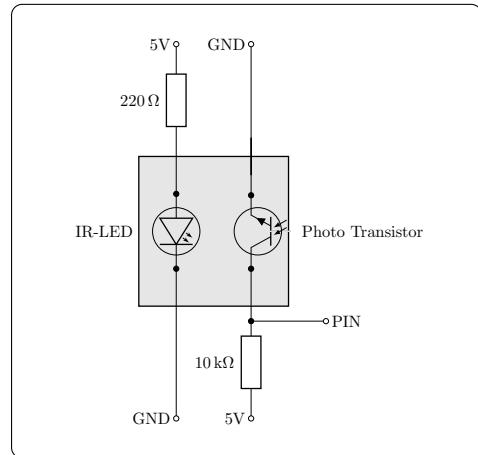
Die Messergebnisse werden übermittelt

**Listing 2.12:** Testsketch "cny90Test" für einen Reflexoptokoppler

Mit Hilfe dieses Sketches ist es möglich, mehr über die Funktionsweise eines CNY70s zu lernen. Vor allem im Sonnenlicht ist auch ein hoher IR-Lichtanteil, dies hat zu Folge, dass es garnicht so leicht ist mit einem LDR zuverlässig zu messen. Deshalb wir zwei mal gemessen! Ohne IR-LED Licht, um den Untergrund zu messen (z.B. Sonnenlicht) und mit IR-LED Licht, der eigentliche Messwert.

### 2.6.3. Aufbau der Schaltung auf dem BreadBoard

Die Schaltung ist etwas schwer aufzubauen! Aber mit den Bildern aus Abb. 2.17 sollte der Aufbau möglich sein. Wichtig ist, dass die Schaltung in zwei Etappen aufgebaut wird.



- 1.) **Sender:** Zuerst baust du den Reflexoptokoppler auf das Bread Board. Dies ist etwas schwierig, da du die vier Beinchen etwas auseinander biegen musst. Achte, dass die IR-LED (schimmert etwas bläulich) an der richtigen Stelle ist. Jetzt kannst du die IR-LED mit GND und über einen 220 $\Omega$  Widerstand mit 5V verbinden. Wenn du das Arduino Board mit dem PC ververbindest, sollte die LED leuchten. Leider sehen wir Menschen IR-Licht nicht! Um die leuchtende LED zu sehen musst du eine digitale Kamera verwenden.
- 2.) **Empfänger:** Der Empfänger besteht aus einem Spannungsteiler mit dem Phototransistor und einem 10k $\Omega$  Widerstand. Die eine Seite des Phototransistors ist schon mit GND verbunden. Bau den 10k $\Omega$  Widerstand auf das Bread Board und verbinde ihn ebenfalls mit 5V. Dann musst du noch den analogen Pin A0 richtig anschließen.

Wenn ihr alles richtig gemacht habt, dann sollte die Schaltung mit Hilfe des Sketchs aus Listing 2.12 funktionieren.

Abbildung 2.16.: Spannungsteiler mit CNY70

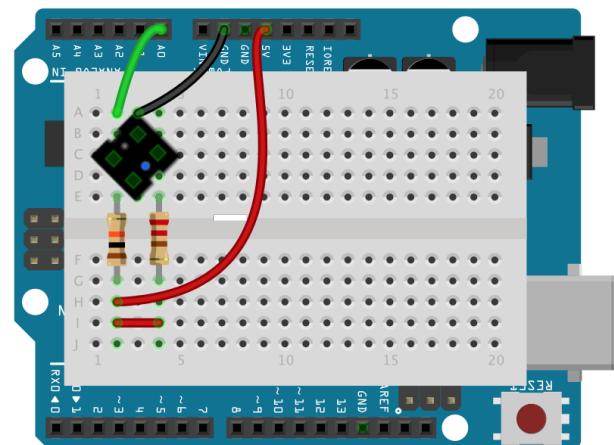


Abbildung 2.17.: Aufbau der Schaltung

---

## 2.6.4. Aufgaben

### Aufgabe 1

In Abb. 2.17 ist eine funktionierende Schaltung abgebildet. Baue diese Schaltung nach, überprüfe zunächst die Funktion der IR-LED mit Hilfe einer Digitalkamera (z.B. deiner Smartphone-Camera<sup>3</sup>). Wenn dieser Teil der Schaltung funktioniert, dann kann die restliche Schaltung aufbauen werden. Achte besonders darauf keinen Kurzschluss zu verursachen und die richtigen Widerstände zu benutzen.

Lade anschließend den Sketch 2.12 auf den Arduino und überprüfe mit Hilfe der seriellen Schnittstelle die Funktion deiner Schaltung.

### Aufgabe 2

Um den Sketch voll zu nutzen musst du deine Schaltung so umbauen, dass die IR-LED über den digitalen PIN 05 mit Spannung versorgt werden kann.

Untersucht nun den Einfluss des Umgebungslichtes auf deine Messergebnisse. Wie kannst du den Einfluss reduzieren. (Physikalische aber auch programmtechnische Lösungen sind möglich)

### Aufgabe 3

Versucht mit Hilfe deiner Schaltung eine schwarze Linie zu erkennen, indem du deine Schaltung über das Papier bewegt und die entsprechenden Werte für weißes Papier bzw. schwarze Linie notierst. Wenn du nun mit dem Reflexoptokoppler über eine schwarze Linie fährst, soll eine grüne LED aufleuchten.

---

<sup>3</sup>Neuere Smartphones funktionieren nicht, da oft die Camera einen guten IR-Filter besitzt



## 2.7. Digitaler Sensor: Der Ultraschallsensor

Ein Ultraschallsensor, wie der "Seeed Ultrasonic Sensor" von Seeed oder "Ping))) Ultrasonic Range Sensorist" von Parallax soll unser erstes Beispiel für einen digitaler Sensor sein. Wenn du die Rückseite des Sensors anschaust, dann siehst du sehr viel Elektronik und man ahnt schon, dass dieser Sensor wohl anderst funktioniert als z.B. ein Photowiderstand.

### 2.7.1. Funktionsweise des Ultraschallsensors

#### Info:

Die Steuerung des Sensors und das Empfangen von Daten über digitale Pins wird auch Protokoll genannt! Im Falle des Ultraschallsensors ist das verwendete Protokoll aber sehr einfach. Das Protokoll, das wahrscheinlich am öftesten verwendet wird ist "IPv4". "IPv4" regelt LAN oder W-LAN Verbindung zwischen elektronischen Geräten.

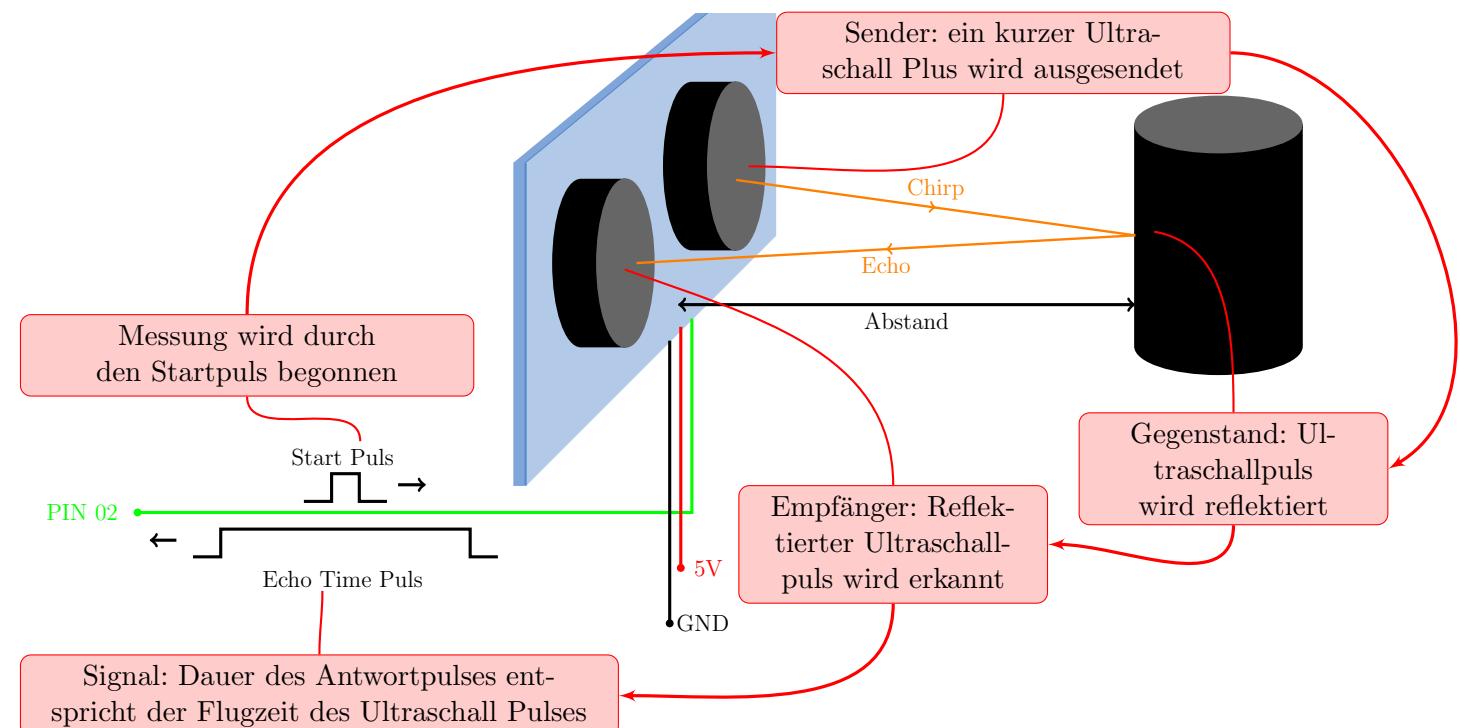


Abbildung 2.18.: Funktionsweise eines Untraschallsensors

## 2.7.2. Beispiel Sketch

```
1 int pingPin = 2;          ○ Digitaler PIN02 ist der Signal-Pin
2
3 void setup() {
4     Serial.begin(9600);
5 }
6
7 void loop() {
8     pinMode(pingPin, OUTPUT); ○ PIN02 wird auf "OUTPUT" gesetzt
9     digitalWrite(pingPin, LOW);
10    delayMicroseconds(2);
11    digitalWrite(pingPin, HIGH); ○ Start Puls wird gesendet
12    delayMicroseconds(15);
13    digitalWrite(pingPin, LOW);
14    delayMicroseconds(20);
15
16
17    pinMode(pingPin, INPUT); ○ PIN02 wird auf "INPUT" gesetzt
18    long cm = pulseIn(pingPin, HIGH)/29/2; ○ Die Pulsdauer wird mit Hilfe der Funktion "pulseIn()" gemessen und mit Hilfe der Schallgeschwindigkeit in cm umgerechnet
19
20    Serial.print(cm);
21    Serial.println(" cm");
22
23    delay(100);
24 }
```

Listing 2.13: Testsketch für einen UltraschallSensor

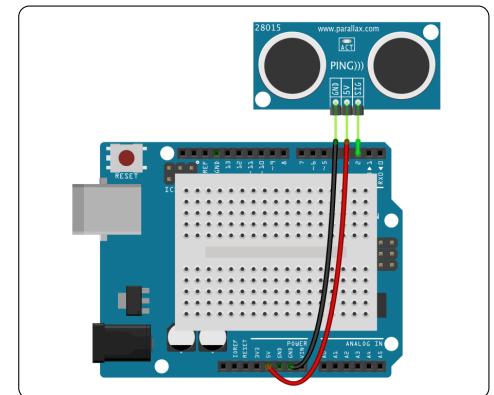


Abbildung 2.19.: Ultraschall Sensor (Ping))

## 2.7.3. Aufgaben

Es gibt auch Ultraschallsensoren mit vier Beinchen. Dabei wird ein Pin zum senden des Start Pulses verwendet und der andere zum empfangen des Echo Pulses. Schreibe deinen Sketch so um, dass du den HC-SR04 Sensor verwenden kannst. Diese Sensoren sind um den Faktor fünf billiger als die mit drei Beinchen. Finde anhand des Datenblattes zu deinem Ultraschallsensor den Messbereich und die Genauigkeit des Ultraschallsensors heraus. Baue dann die Schaltung entsprechend Abb. 2.19 oder 2.20 auf. Du muss unbedingt die Bezeichnung der Pins auf dem Sensor beachten, alle Pins

### Info:

Um das Datenblatt im Internet zu finden, verwendest du die Suchwörter "Datasheet" und die Bezeichnung des Sensors

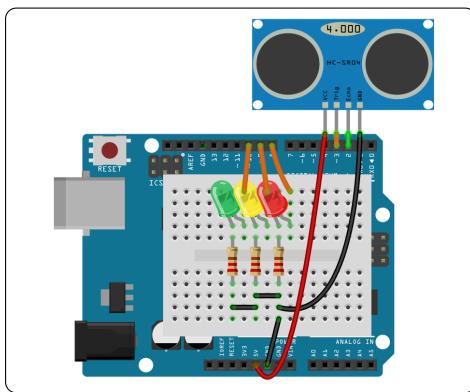


Abbildung 2.20.: Ultraschall Sensor HC-SR04

müssen richtig verbunden sein, ansonsten kann der Sensor zerstört werden (und die sind leider teuer).

### Aufgabe 1

Lade den Sketch aus Listing 2.13 auf deinen Arduino und ändere ihn gegebenenfalls ab. Mach dich mit der Funktion vertraut und überprüfe den Messbereich und die Genauigkeit des Sensors.

### Aufgabe 2

Bau einen Ultraschallsensor und drei farbige LEDs (grün, gelb und rot) auf dein BreadBoard. Schreibe einen Sketch der einen optischen Abstandsmelder realisiert.

```
1 int trigPin = 3; ○  
2 int echoPin = 2; ○  
3  
4 void setup() {  
5     Serial.begin(9600);  
6     pinMode(trigPin, OUTPUT);  
7     pinMode(echoPin, INPUT);  
8 }  
9  
10 void loop() {  
11     digitalWrite(trigPin, HIGH);  
12     delay(1000);  
13     digitalWrite(trigPin, LOW); ○  
14  
15     long cm = pulseIn(echoPin, HIGH)/29.1/2; ○  
16 }
```

Digitaler PIN02 ist der Trigger-Pin

Digitaler PIN02 ist der Echo-Pin

PIN02 wird auf auf HIGH gesetzt um den Start Puls zu senden

Die Pulsdauer wird mit Hilfe der Funktion "pulseIn()" gemessen und mit Hilfe der Schallgeschwindigkeit in cm umgerechnet

Listing 2.14: Testsketch "hc-sr04Test.ino" für den HC-SR04

## 2.8. Umwelt Sensoren

Es gibt Sensoren, die so komplex sind, dass sie durch ein spezielles Protokoll gesteuert werden müssen. Für diese Sensoren ist dieses Protokoll in einer sogenannten Bibliothek (eng. Library) schon programmiert. Um den Sensor zu Benutzen wird am Anfang des Sketches diese mit Hilfe des Befehls "#include "libName.h";" eingebunden. Dadurch wird der Befehlssatz des Arduino erweitert. Der Kombi-Sensor DH11 ist so ein Sensor. Er verbindet die Temperatur-Messung mit einer Feuchtigkeits-Messung der Luft. Mit Hilfe dieses Sensors kann der Taupunkt indirekt gemessen werden und so z.B. eine intelligente Raumlüftung realisiert werden, die effektiv das kondensieren von Wasser an Wänden verhindert und so der Schimmelbildung aktiv entgegenwirkt.

### 2.8.1. Beispiel Sketch

```
1 #include "DHT.h"
2
3 DHT dht(6, DHT11);
4
5 void setup() {
6     Serial.begin(9600);
7     dht.begin();
8 }
9
10 void loop() {
11     delay(2000);
12
13     float h = dht.readHumidity();
14     float t = dht.readTemperature();
15
16     if (isnan(h) || isnan(t)) {
17         Serial.println("Failed to read!");
18     } else {
19         Serial.print("H= " + h + " % ");
20         Serial.print("T= " + t + " *C ");
21     }
22 }
```

Zunächst wird die Bibliothek für den Sensor eingebunden

Ein Sensor "Objekt" wird erzeugt

Der Sensor wird gestartet, das dauert bis zu 2 Sekunden

Das Auslesen der Luftfeuchtigkeit und Temperatur dauert jeweils 0,25 Sekunden!

Ausgabe erfolgt nur, wenn das Lesen geklappt hat

Listing 2.15: Test-Sketch für den DHT 11 Sensor

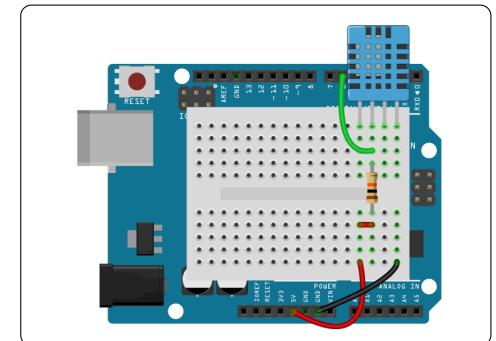


Abbildung 2.21.: Temperatur- und Feuchtigkeits-Sensor DH11

---

## 2.8.2. Aufgaben

Baue den DHT11 Sensor auf dem BreadBoard auf und teste deine Schaltung mit Hilfe des Beispielsketches.

### Aufgabe 1

Schreibe ein Funktion vom Typ float, die als Parameter die Temperatur t und Luftfeuchtigkeit h besitzt und die Taupunkttemperatur  $T_p$  berechnet.

$$T_p = \left( \frac{h}{100} \right)^{\frac{1}{8,02}} (109,8 + t) - 109,8 \quad (2.5)$$

```
1 ttp = pow(H/100, 1.0/8.02)*(109.8+t)+109.8;
```

**Listing 2.16:** Taupunkttemperatur “ttp”

Erweitere deinen Sketch, so dass die Taupunkttemperatur berechnet wird und wenn die aktuellen Temperatur kleiner als die Taupunkttemperatur wird eine Warnung über die serielle Schnittstelle versendet wird.

---

## 2.9. Daten Speichern und Auslesen

### 2.9.1. Die verschiedenen Speicher

Auf dem Arduino Uno der ATmega328 Microcontroller verbaut. Er besitzt drei Arten von Steuer:

- Flash ist der Speicher, in dem der Sketch gespeichert und ausgeführt wird.
- SRAM ist der Speicher, in dem Variablen gespeichert und manipuliert werden können.
- EEPROM ist der Speicher, der von Programmen genutzt werden kann um Daten dauerhaft zu speichern.

Flash Speicher und EEPROM Speicher sind nicht flüchtig, das heißt die Daten sind auch nach Stromunterbrechung noch vorhanden. SRAM Speicher ist flüchtig, das heißt die gespeicherten Daten sind nach einer Stromunterbrechung nicht mehr vorhanden. Der ATmega328 IC besitzt folgende Speichergrößen:

Speicher	Größe	Bemerkung
Flash	32kB	0.5kB enthält den Bootloader
SRAM	2kB	Speicherbereich für Variablen
EEPROM	1kB	Langzeit Speicher

Abbildung 2.22.: Speichergrößen des Arduino Uno

Es gibt nicht wirklich viel SRAM auf dem Arduino Uno! Bei größeren Datenmengen kann es passieren, dass er vollständig gefüllt wird. Zum Beispiel

```
1 char message[] = "I support the Cape Wind project.";
```

belegt 33 Bytes im SRAM, jedes Zeichen und das Endzeichen '\0' ein Byte. Dies scheint nicht viel zusein, aber es braucht nicht viel um 2048 Bytes zu füllen. Wenn du den ganzen SRAM aufgebracht hast, kann es zu komischen Fehlern kommen.

Du kannst so eine Sketch oft ohne Fehler auf den Arduino laden, da der SRAM während der Laufzeit gefüllt wird. Wenn der Sketch dann ausgeführt werden soll funktioniert er nicht richtig. Es kommt zu komischen Fehlern, die nur sehr schwer zu verstehen sind.

#### Info:

1kB = 1024 Byte  
B steht für Byte  
ein Byte besteht aus 8 Bit  
1 Bit ist die kleinste Speichergröße

---

## 2.9.2. EEPROM Speicher benutzen

```
1 #include <EEPROM.h>
2
3 void setup()
4 {
5     for (int i = 0; i < 255; i++)
6         EEPROM.write(i, i);
7 }
8
9 void loop()
10{
11}
```

Listing 2.17: EEPROM mit Daten füllen

```
1 #include <EEPROM.h>
2
3 int a = 0;
4 int value;
5
6 void setup() {
7     Serial.begin(9600);
8 }
9
10void loop() {
11    value = EEPROM.read(a);
12
13    Serial.print(a);
14    Serial.print(" = ");
15    Serial.println(value);
16
17    a = a + 1;
18    if (a == 256) {
19        a = 0;
20    }
21    delay(500);
22}
```

Listing 2.18: EEPROM mit Daten füllen

## 2.10. C Sprachstruktur: Schleifen

Eine Schleife ist eine weitere Kontrollstruktur. Sie dient zur Wiederholung eines Anweisungsblocks, den sogenannten Schleifenrumpf oder Schleifenkörper. Dieser wird solange die Schleifenbedingung als Laufbedingung gültig bleibt bzw. die Abbruchbedingung nicht eintritt wiederholt. Schleifen, die keine Schleifenbedingung haben, sind Endlosschleifen. Im Arduino-Sketch stellt die Loop-Methode eine solche Endlosschleife dar.

Prinzipiell können verschiedene Schleifentypen unterschieden werden:

- Die vorprüfende oder kopfgesteuerte Schleife.

Bei dieser Schleife wird eine Bedingung geprüft, mit der vorher entschieden wird, ob der Schleifenkörper ausgeführt wird, diese wird meist als while-Schleife bezeichnet.

- Die nachprüfende oder fußgesteuerte Schleife.

Bei dieser Schleife wird nach dem Durchlauf des Schleifenkörpers eine Bedingung überprüft, ob der Schleifenrumpf nochmal ausgeführt wird (meist als DO...WHILE = „ausführen...solange“ oder REPEAT...UNTIL = „wiederholen...bis“ Konstrukt).

- Die Zählschleife, eine Sonderform der vorprüfenden Schleife (meist als FOR = für -Schleife implementiert).

- Die Mengenschleife, eine Sonderform der Zählschleife (meist als FOREACH = „für jedes Element der Menge“ implementiert).

- Schleife mit Laufbedingung: Wertet die Bedingung zu „wahr“ aus, wird die Schleife fortgesetzt. Schleife mit Abbruchbedingung: Wertet die Bedingung zu „wahr“ aus, so wird die Schleife abgebrochen.

### 2.10.1. for-Schleife: diezählende Wiederholung

Eine for-Schleife ist eine Kontrollstruktur, mit der man eine Gruppe von Anweisungen, den Schleifenkörper mit einer bestimmten Anzahl von Wiederholungen ausführen kann. Die Anzahl der Wiederholungen steht schon beim Eintritt in die Schleife fest. Es gibt eine Schleifenvariable, die am Anfang auf den Startwert gesetzt wird und dann jeweils um die Schrittweite verändert wird, bis der Zielwert erreicht ist. Die Schleifenvariable, der Startwert, die Schrittweite und der Endwert müssen numerisch sein.

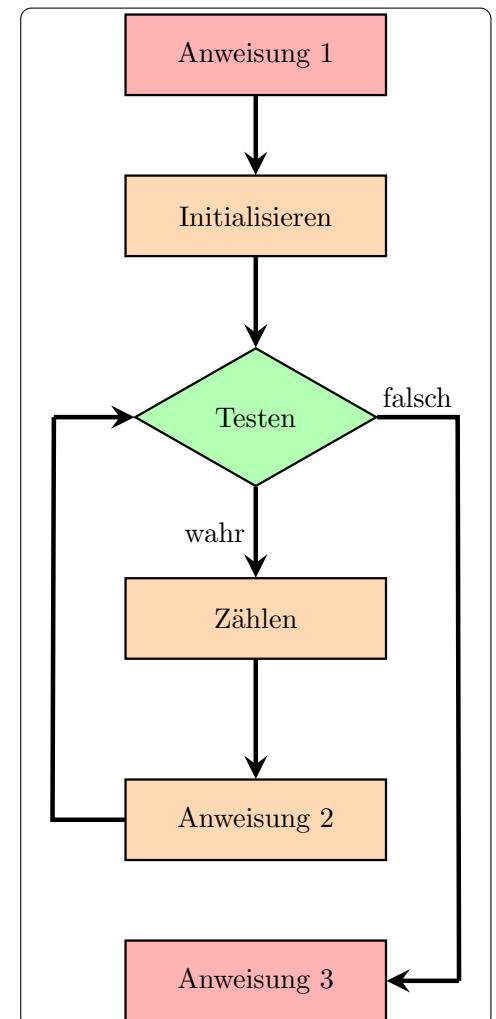


Abbildung 2.23.: for-Loop

---

```
1
2
3 for (Initialisieren; Testen; Zählen) {
4
5   // Anweisungen;
6
7
8 }
9 }
```

Initialisieren der Schleifenvariablen

Bedingung für die Schleifenvariablen überprüfen.

Schleifenvariablen erhöhen

Anweisungen, die wiederholt ausgeführt werden.

**Listing 2.19:** Die for-Schleife

In C ist die for-Schleife viel flexibler als for-Schleifen in vielen anderen Programmiersprachen, einschließlich BASIC. So kann auf alle der drei Kopfelemente verzichtet werden, nur die Strichpunkte sind nötig.

---

## Sensorwert in Array speichern und Mittelwert berechnen

```
1 const int senPin = A0;
2 const int numberOfValues = 10;
3 float senValues[numberOfValues];
4 float meanValue = 0.0;
5
6 void setup() {
7     Serial.begin(9600);
8 }
9
10 void loop() {
11     for (int i=0;i<numberOfValues;i++) {
12         senValues[i] = analogRead(senPin);
13         delay(20);
14     }
15     for (int i=0;i<numberOfValues;i++) {
16         meanValue += senValues[i];
17     }
18 }
19
20 meanValue /= numberOfValues;
21 Serial.println(meanValue);
22 delay(200);
23 }
```

Anzahl der Einzelmessungen  
Array zur Speicherung der Einzelmessungen  
Speicherplatz für gemittelten Sensorwert.  
Loop für die Einzelmessungen  
Loop für die Mittelwertbildung  
Mittelwert berechnen und an PC senden

**Listing 2.20:** Mittelwert berechnen

### 2.10.2. Aufgaben

#### Aufgabe 1

Schreibe einen Sketch, der die Summe der ganzen Zahlen von 1 bis 10 berechnet. Gib dein Ergebnis über die serielle Schnittstelle aus.

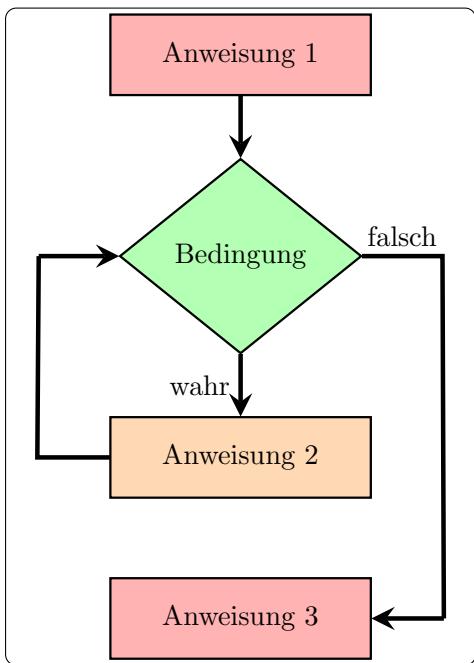


Abbildung 2.24.: while-Loop

### 2.10.3. while-Schleife: die bedingte Wiederholung

```

1
2
3 while (bedingung) {
4     // Anweisungen;
5 }
6
7

```

Bedingung auf Wahrheit überprüfen.

Anweisungen, die wiederholt ausgeführt werden.

Listing 2.21: Die for-Schleife

### 2.10.4. Sprungbefehle

Möchte man innerhalb einer Schleife (for, do oder while) auf ein Ereignis reagieren, indem die Schleife nicht weiter ausgeführt wird, steht der Befehl break zur Verfügung. Sprungbefehle sollte aber nur dann benutzt werden, wenn es keine andere Lösung gibt (, was praktisch nie der Fall ist).

```

1 for (x = 0; x < 255; x++)
2 {
3     analogWrite(pwmPin, x);
4     sens = analogRead(sensorPin);
5     if (sens > threshold){
6         x = 0;
7         break;
8     }
9     delay(50);
10 }
11

```

Wenn ein bestimmter Sensorwert überschritten ist, dann wird mit Hilfe des Befehls Break das weiter abarbeiten der for-Schleife unterbrochen

und der Sketch nach dem Ende des Schleifenkörpers sortgesetzt

Listing 2.22: MVerwendung von break

# 3 — Actoren

Die Maschine kann nur tun, was wir ihr zu befehlen wissen.

---

Ada Lovelace (1815-1852) eigentlich Augusta Ada King Byron, Countess of Lovelace, britische Mathematikerin

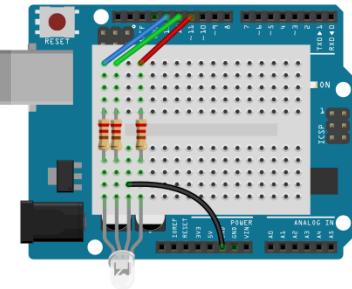


Abbildung 3.1.: RGB-LED

#### Info:

Es gibt auch Anoden RGB-LEDs, der gemeinsame Pin ist dann die Anode (plus Pol).

## 3.1. RGB-LEDs

RGB-LED bestehen aus 3 farbigen (rot, grün und blau) LEDs. Die eine gemeinsame Kathode (Minuspol) besitzen. Sie eignen sich dazu Mischfarben, die aus den drei Grundfarben aufgebaut sind darzustellen. Mit zwei Potentiometer und einem weißen Tischtennisball können schöne Farbeffekte erzeugt werden.

### 3.1.1. Farben mit Potentiometer mischen

An vielen elektronischen Geräten findet man Regler, an denen man drehen oder schieben kann. Dahinter stecken meistens Potentiometer, kurz „Poti“ genannt. Sie sind regelbare Widerstände und haben meistens drei Anschlüsse. Üblicherweise ist der linke Anschluss an einem Ende einer Kohleschicht befestigt, der rechte Anschluss am anderen Ende. Der mittlere Anschluss, genannt Mittelabgriff, ist mit einem Kontakt verbunden, der auf der Kohleschicht schleift und beim Drehen oder Schieben verschoben wird. Da die Kohlebahn relativ schlecht leitet, fließt ein Strom leichter durch ein kurzes dünnes Stück Kohlespur als durch ein langes ebenso dünnes Stück Kohle. Beim Drehen des Potentiometers wird also der elektrische Widerstand zwischen dem Mittelabgriff und jedem der anderen Anschlüsse verändert. Wenn man Potentiometer kauft, ist meistens der Maximalwiderstand angegeben. Der minimale Widerstand beträgt nahezu  $0\Omega$ .

### 3.1.2. Aufbau der Schaltung

RGB-LEDs sind empfindlich und relativ teuer, das heißt sie dürfen nur mit Vorwiderstand betrieben werden. Wenn du die Schaltung ohne die  $220\Omega$ -Widerstände aufbaust bist du schnell um ca. 1 Euro ärmer, aber um eine Kaputte RGB-LED reicher! Ansonsten erfolgt der Aufbau genau wie mit normalen LEDs. Der längste Anschluss ist die gemeinsame Kathode (Minus-Pol) und wird mit GND verbunden. Die anderen Pins werden mit PWM-Ports verbunden.

### 3.1.3. Aufgaben

#### Aufgabe 1

Baue zuerst die Schaltung entsprechend Abb. 3.1 auf. Öffne den Beispiel-Sketch Beispiele/01Basic/Fade und ändere den Sketch so ab, dass die rote LED über den digitalen PIN 11, die blaue auf dem digitalen PIN 12 und die grüne LED auf dem digitalen PIN 13 angesteuert werden können.

Ändere für die verschiedenen Ports den brightness-Wert und versuche so verschiedene Farben zu erzeugen.

## Aufgabe 2

Für dieses Aufgabe benötigst du ein größeres Bread-Board. Bau dann die Schaltung entsprechend der Abb. 3.2 auf. Lade anschließend den Beispiel-Sketch Beispiele/10StarterKit/p04\_ColorMixingLamp. Anstelle der Photo-Widerstände werden bei dieser Aufgabe die Potis verwendet. Mache dich mit der Funktionsweise dieses Sketches vertraut.

## Aufgabe 3 Projekt

Wenn du die Aufgabe 1 und 2 gut verstanden hast, dann solltest du jetzt in der Lage sein RGB-LED in verschiedenen Projekten zu verwenden. Vielleicht wäre es eine gute Übung für dich dein optisches Thermometer anstatt mit 3 LEDs (rot, gelb und grün) mit einer RGB-LED neu aufzubauen. Eine weitere Idee wäre es eine optische Abstandswarnung zu bauen. Im Abschnitt 3.1.4 ist noch Hintergrundwissen zur Farberzeugung mit Hilfe einer RGB-LED zu finden.

### 3.1.4. Der Farbraum

Ein RGB-Farbraum ist ein additiver Farbraum, der Farbwahrnehmungen durch das additive Mischen dreier Grundfarben (Rot, Grün und Blau) nachbildet. Das Farbsehen des Menschen ist von drei Zapfentypen geprägt.

Dieser Farbraum wird für selbstleuchtende (farbdarstellende) Systeme benutzt, die dem Prinzip der Additiven Farbmischung unterliegen, daher auch als Lichtmischung bezeichnet. Nach Graßmanns Gesetzen lassen sich Farben durch drei Angaben definieren, im RGB-Farbraum sind dies der Rot-, der Grün- und der Blauanteil. Die konkrete Form des Farbraums hängt vom jeweils konkreten technischen System ab, für das der jeweilige Farbraum bestimmt wurde.

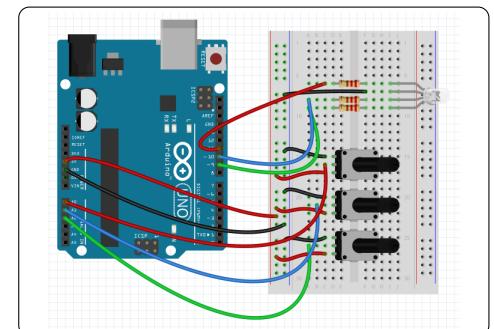


Abbildung 3.2.: RGB Mischer

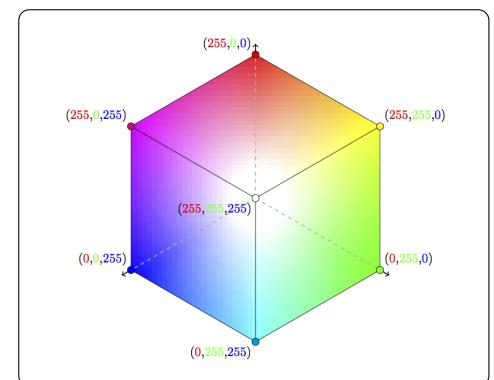


Abbildung 3.3.: RGB Farbenraum

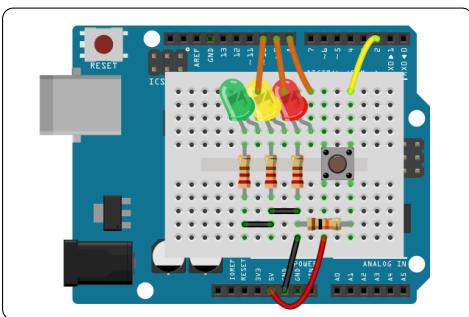


Abbildung 3.4.

## 3.2. C Sprachelemente: millis() versus delay()

Bisher hast du eine LED mit Hilfe der delay()-Funktion ein und ausgeschaltet. Das ist in Ordnung, solange das Arduino Board nur eine Aufgabe hat. Aber wie sollte mit dem delay()-Befehl folgende Aufgabe gelöst werden?

### Aufgabe 1

Bau die Schaltung aus Abbildung 3.4 auf. Die rote LED soll blinken (eine Sekunde an, eine Sekunde aus). Die grüne LED leuchten, wenn der Push-Button gedrückt ist und nicht leuchten wenn der Push-Button losgelassen wird.

### Lösungs Hinweis

Um diese Aufgabe zu lösen, muss man das An- und Ausschalten der roten LED so steuern, dass der Arduino nicht blockiert wird. Eine mögliche Lösung bietet der millis()-Befehl. Wenn millis() aufgerufen wird, dann gibt er die Zeit in Millisekunden zurück, seit der Arduino das aktuelle Programm ausführt.

```
1 const int blinkLed = 13;
2 long time = 0;          ○ Aktuelle Ausführzeit des
3 boolean stateLed = false; ○ Sketches wird gespeichert
4
5 void setup() {
6   pinMode(blinkLed, OUTPUT);
7   time = millis();          ○ Aktueller Zustand der
8 }
9
10 void loop() {           ○ blickLed wird gespeichert
11   if (millis()-time > 1000){  ○ Zustand der blickLed wird
12     stateLed = !stateLed;    ○ nach einer Sekunde geändert
13     time = millis();        ○ Zustand der blickLed wird übertragen
14     digitalWrite(blinkLed, stateLed);  ○
15   }
16 }
```

Listing 3.1: Aufbau einer C-Funktion

### Info:

! ist der Negations-Operator

Mit Hilfe des Listings 3.1 sollte es dir möglich sein Aufgabe 1 zu lösen.

### 3.3. Lautsprecher

In Kapitel 1 haben wir zur Visualisierung des digitalen Ausgänge LED's benutzt. Statt optischer Ausgaben über LEDs können aber auch akustische Ausgaben erzeugt werden. Dazu wird ein Lautsprecher an einen Ausgang des Mikrocontrollers angehängt und schnell ein und ausgeschaltet. Dies erzeugt Kräfte in der Spule des Lautsprechers. Diese Kräfte lassen die Membran im Lautsprecher hin und her schwingen und wir hören die entsprechende Frequenz der Membranschwingung als Ton.

#### 3.3.1. Lautstärkeregelung mit einem Potentiometer

Ihr sollt hier ein Potentiometer verwenden, um die Lautstärke des ohnehin recht leisen Tons noch leiser einzustellen, bis der Ton schließlich unhörbar leise wird. Dazu muss die Stromstärke im Lautsprecher verringert werden - der zusätzliche veränderbare Widerstand muss also zwischen den Port und den Lautsprecher oder - das ist genau so gut - zwischen den Lautsprecher und den 5 V des digitalen Ports geschaltet werden.

#### 3.3.2. Aufgaben

Baue die Schaltung mit dem Lautsprechen entsprechend Abb. 3.5 auf.

##### Aufgabe 1

Für deine ersten Töne genügt es den „Blink“ Sketch entsprechend anzupassen. Dazu muss du den Befehl Delay (Millisekunden) durch ein DelayMicroseconds (Microsekunden) ersetzen und auf den Arduino laden. Wenn dein Lautsprecher mit dem richtigen Port verbunden ist solltest du einen konstanten Ton hören.

##### Aufgabe 2

Für die folgende Aufgabe solltest du auf deine Mitschüler Rücksicht nehmen, indem du ein Potentiometer zur Lautstärke-Regelung einbaust. Bau dazu deine Schaltung entsprechend der Abb. 3.6 um. In der Tabelle 3.1 sind die Frequenzen der Töne einer C-Dur Tonleiter aufgeführt.

- Berechnen anhand der Werte aus der Tabelle die Perionendauer für die verschiedenen Töne. Um einen Ton jetzt zu erzeugen, muss der digitale Pin jeweils für die Hälfte der Periodendauer 'HIGH' bzw. 'LOW' sein.
- Schreibe einen Sketch, der das Lied Alle meine Endchen spielt.

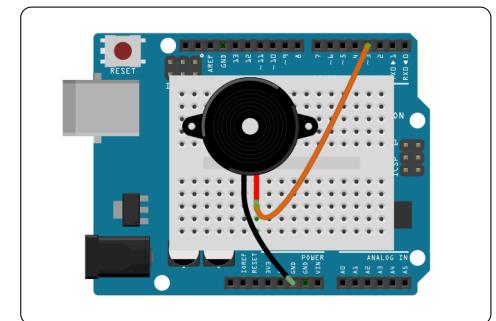


Abbildung 3.5.: Arduino mit Piezolautsprecher

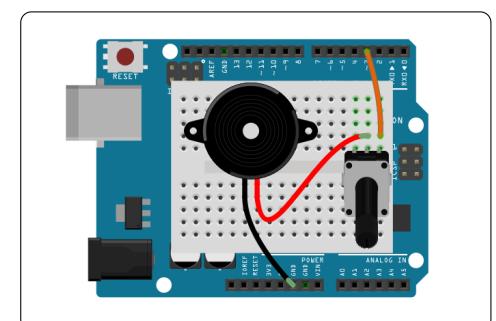


Abbildung 3.6.: Lautsprecher mit Poti

---

Ton	c	d	f	g	a
Frequenz (Hz)	294	329	349	392	440

**Tabelle 3.1.:** C-Dur Tonleiter mit Frequenzen

c) Schreibe eine C Funktion:

```
void playTone(int tone, int duration)
```

Die einen gegebenen Ton 'tone' für eine bestimmte Zeit 'duration' abspielt. Schreibe den Sketch aus Teil b) entsprechend um, dass jetzt diese Funktion verwendet wird.

### Aufgabe 3 Projekt

Der Beispiel-Sketch „Examples/Digital/Melody“ macht das ganze etwas eleganter. Lade den Sketch auf dein Board. Wenn das Abspielen klappt, kannst du versuchen die Melodie zu ändern. Dazu muss du allerdings verstanden haben, wie das Beispiel genau funktioniert.

Wichtig: Du kannst nur die Töne verwenden, die in der Datei pitches.h definiert sind.

#### 3.3.3. Zum weiterlesen:

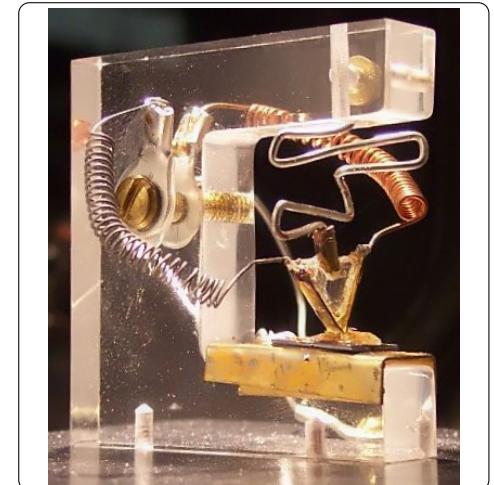
- <http://arduino.cc/en/Tutorial/Tone>

### 3.4. Der Transistor als Schalter oder zur Verstärkung

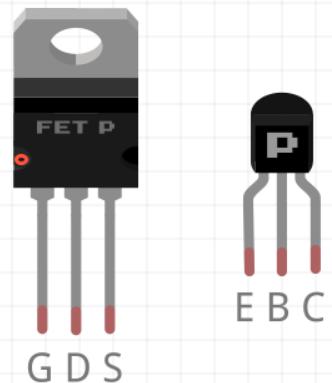
Ein Transistor ist ein elektronisches Bauelement zum Schalten und Verstärken von elektrischen Signalen, ohne dabei mechanische Bewegungen auszuführen. Transistoren sind die weitaus wichtigsten „aktiven“ Bestandteile elektronischer Schaltungen, welche beispielsweise in der Nachrichtentechnik, der Leistungselektronik und in Computersystemen eingesetzt werden. Besondere Bedeutung haben Transistoren in integrierten Schaltkreisen, was die derzeit weit verbreitete Mikroelektronik ermöglicht.

Es gibt zwei wichtige Gruppen von Transistoren, nämlich Bipolartransistoren und Feldeffekttransistoren (FET).

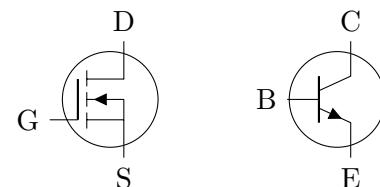
Der Metall-Oxid-Halbleiter-Feldeffekttransistor (englisch metal-oxide-semiconductor field-effect transistor, MOSFET auch MOS-FET, selten MOST) gehört zu den Feldeffekttransistoren mit isoliertem Gate. Er ist den Metall-Isolator-Halbleiter-Feldeffekttransistoren (MISFET) zuzurechnen. Obwohl heute dotiertes Polysilizium als Gate-Material vorherrscht, wurde die Bezeichnung MOSFET beibehalten.



**Abbildung 3.7.:** Nachbau des ersten Transistors (CC Attribution-Share Alike 3.0 Unported by commons.wikimedia.org)



(a) PinOut: MOSFET und bipolarer Transistor



(b) Schaltzeichen: MOSFET und bipolarer Transistor

**Abbildung 3.8.:** MOSFET (n-Kanal) und bipolaren Transistor (npn)

---

### 3.4.1. Einsatzmöglichkeiten für einen bipolaren Transistor

Grundsätzlich gibt es zwei verschiedene Möglichkeiten einen bipolaren Transistor zu betreiben:

- Proportionalbetrieb: Der Kollektor- bzw. Emitterstrom ist proportional zum Basisstrom. Je größer der Basisstrom desto größer der Kollektorstrom. Den Verstärkungsfaktor selbst kannst du im Datenblatt nachlesen. Da die Spannung  $U_{CE}$  recht groß ist erwärmt sich der Transistor (Verlustleistung!).
- Schaltbetrieb: Der Transistor leitet entweder voll oder sperrt komplett. Der Basisstrom muss groß genug sein, um den Kollektor- bzw. Emitterstrom nicht zu begrenzen. Ein Basiswiderstand ist für den Schutz des digitalen Ports nötig, nicht für den Betrieb des Transistors. Ohne Basiswiderstand würde der Ausgang des Arduino fast kurzgeschlossen, denn die Vorwärtsspannung  $U_{BE}$  ist in der Regel etwa 0,7 V (Achtung: für jeden Transistor muss dieser Wert aus dem Datenblatt entnommen werden). Du benötigst deshalb einen Vorwiderstand von etwa  $100\Omega$ , der den Strom durch den digitalen Port auf maximal 40 mA begrenzt.

#### Info:

Möchte man einen Transistor nur als Schalter verwenden, so ist ein MOSFET einem bipolaren Transistor vorzuziehen, da ein MOSFET praktisch keine Verlustleistung hat.

Um die Musik in vernünftiger Lautstärke über einen Lautsprecher auszugeben reicht die maximale zulässige Stromstärke der digitalen Pots nicht aus.

### 3.4.2. Aufbau der Schaltung

Beachte, dass du je nach Transistor im Datenblatt nachsehen musst, wo sich die Basis, der Collector und der Emitter befinden.

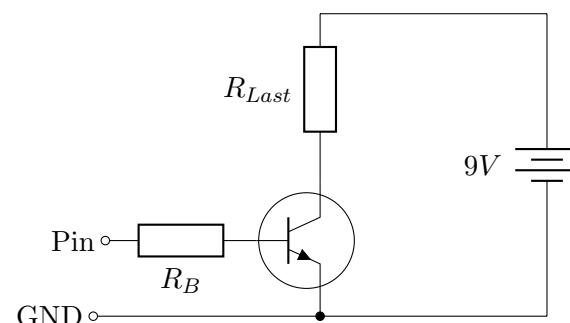


Abbildung 3.9.: Grundschaltung bipolarer Transistor (npn)

---

### 3.4.3. Aufgaben

Baue die Schaltung nach Abb. 3.10. Beachte, dass du für den verwendeten Transistor im Datenblatt nachsehen musst, wo sich die Basis, der Collector und der Emitter befinden.

#### Aufgabe 1

Wenn du dich überzeugt hast, dass die Schaltung richtig aufgebaut ist. Kannst du die Wirkung des bipolaren Transistor und des Pots mit Hilfe deines Melodie-Sketch testen.

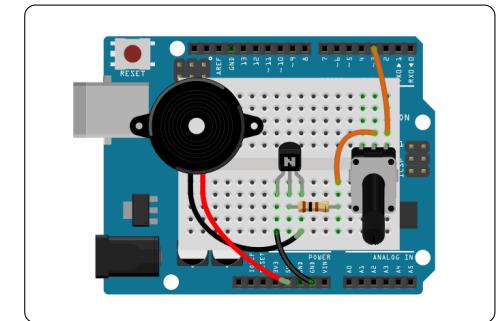


Abbildung 3.10.: npn-Transistor

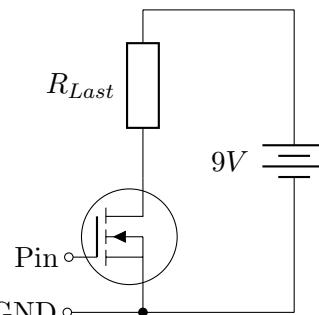
## Info:

Natürlich kann eine MOSFET auch als Verstärker geschaltet werden. Da es bei ihm keinen Gate-Source-Strom gibt, wird mit Hilfe der Gate-Source-Spannung der Collector-Source-Strom geschaltet. Da aber ohne Spannungsteiler immer 0V oder 5V Potentialunterschied anliegen, schaltet der MOSFET eben nur.

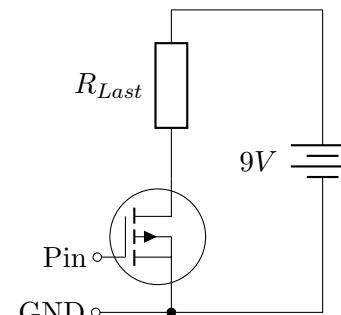
### 3.4.4. MOSFET als Schalter

Auch mit einem MOSFET ist es möglich den Lautsprecher lauter zu bekommen. MOSFET werden dann bevorzugt eingesetzt, wenn die Verlustleistung entscheidend wird. Da es keine Collector-Source-Strom gibt, ist die Verlustleistung praktisch Null. Der zweite Einsatz ist wenn sehr große Lasten geschaltet werden, so steuert man ein Relais üblicherweise mit einem MOSFET an.

### 3.4.5. Aufbau der Schaltung



(a) Direkte Methode: N-Kanal MOSFET (NFET)



(b) Direkte Methode: P-Kanal MOSFETs (PFET)

Abbildung 3.11.: Grundschaltung MOSFET

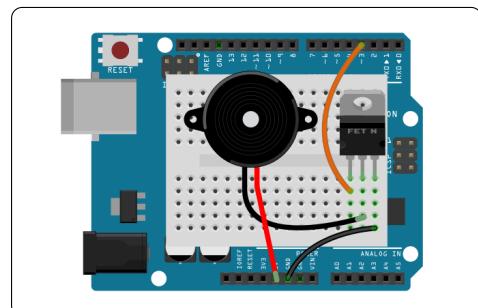


Abbildung 3.12.: n-MOSFET Verstärker

Zu beachten ist, dass zwischen Gate und Source und zwischen Gate und Drain eine Kapazität existiert, welche bei jedem Umschalt-Vorgang umgeladen werden muss. Besonders bei höheren Frequenzen (größer 10 kHz) ist zur Strombegrenzung zwischen Gate und dem digitalen Port des Arduinos ein Widerstand sinnvoll. Übliche Werte dafür sind  $50 - 100\Omega$ .

### 3.4.6. Aufgaben

Baue die Schaltung mit dem Lautsprecher und n-MOSFET Transistor entsprechend Abb. 3.12 auf. Anstatt der Batterie wird hier die 5 V des Arduino Uno verwendet.

---

### Aufgabe 1

Wenn du dich überzeugt hast, dass die Schaltung richtig aufgebaut ist. Kannst du die Wirkung des MOSFETs mit Hilfe deines Melodie-Sketch testen.

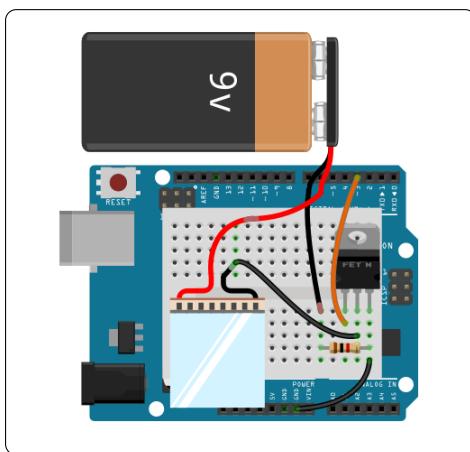


Abbildung 3.13.: Peltier-Element

## 3.5. Peltier-Element

Das Peltier- oder thermoelektrische Element, ist ein Bauteil, das Wärme (Entropie) von einer Seite auf eine andere übertragen kann, wenn ein Strom angelegt wird. Es kann zum heizen oder kühlen verwenden werden, dazu muss die Stromrichtung umgepolt werden. Das Peltier-Element kann mit zu einer Spannung von 15,4 V und einem maximalen Strom von 7A versorgt werden.

### 3.5.1. Ausbau der Schaltung

Du benötigst ein Peltier-Element, einen N-Mosfet, einen  $1000\Omega$  Widerstand und einen Batteriehalter mit zwei Akkus. Die Abb. 3.13 zeigt den Aufbau der Schaltung.

### 3.5.2. Der Beispiel-Sketch

```
1 int peltierPin = 3;
2 int power = 0;
3 int peltLevel = 0;
4
5 void setup(){
6     Serial.begin(9600);
7     pinMode(peltierPin, OUTPUT);
8 }
9
10 void loop(){
11     if(Serial.available() > 0) {
12         int deltaPower = Serial.read();
13         power += deltaPower
14         peltLevel = map(power, 0, 99, 0, 255);
15     }
16     Serial.print("Power= ");
17     Serial.print(power);
18     analogWrite(peltierPin, peltLevel);
19 }
```

This is a value from 0 to 255 that actually controls the MOSFET

Write this new value out to the port

Listing 3.2: Peltier Kühlung

---

Das besondere am Sketch aus Listing 3.2 ist, dass das Verhalten des Arduinos über die serielle Schnittstelle gesteuert wird. Man sollte die Schaltung nicht zu lange betreiben, das durch den MOSFET schon einiges an Energie fließt und sich dieser durch die Verlustleistung stark aufheizen kann.

### 3.5.3. Aufgaben

Bau die Schaltung entsprechend Abb. 3.13 auf. Achte darauf, dass der N-Mosfet richtig eingebaut ist.

**Info:**

Achtung: Der Mosfet kann bei lagen Betrieb heiß werden!

#### Aufgabe 1

Lade den Beispiel-Sketch 3.2 auf deinen Arduino und öffne anschließend das serielle Terminal. Mache dich mit der Funktionsweise des Sketch vertraut und dokumentier alle noch nicht dokumentierten Befehle.

#### Aufgabe 2

Erweitere deine Schaltung und deinen Sketch um einen Temperatur-Sensoren. Messe die Temperatur auf beiden Seiten des Peltier-Elements und gibt die gemessene Temperatur ebenfalls über die serielle Schnittstelle aus.

#### Aufgabe 3 Projekt

Erweitere deine Schaltung um ein zweites Peltier-Element das das Heizung geschaltet ist. Erweitere deinen Sketch, so dass eine vorgegebenen Temperatur in einem abgeschlossenen Behälter gehalten werden kann.

---

## 3.6. Servomotor

Soll ein Arduino etwas bewegen, dann gibt es mehrere Möglichkeiten dies zu tun. Es können normale Elektromotoren, Schrittmotoren oder Modellbauservos benutzt werden. Modellbauservos haben den Vorteil, dass sie leicht anzusteuern sind, und die notwendige Leistungselektronik bereits im Servo eingebaut ist. Auch benötigt man keine aufwändigen Rückmeldungen von der Mechanik um eine bestimmte Position anzufahren oder um die Schrittverluste eines Schrittmotors auszugleichen. Ein Servo findet ganz von alleine nach dem Einschalten seine Neutralposition.

Servos gibt es in allen möglichen Preiskategorien, wobei die Unterschiede in dem Drehmoment des Motors bzw. in der Qualität des integrierten Getriebes liegen.

Ein Servo besteht aus mehreren logischen Komponenten

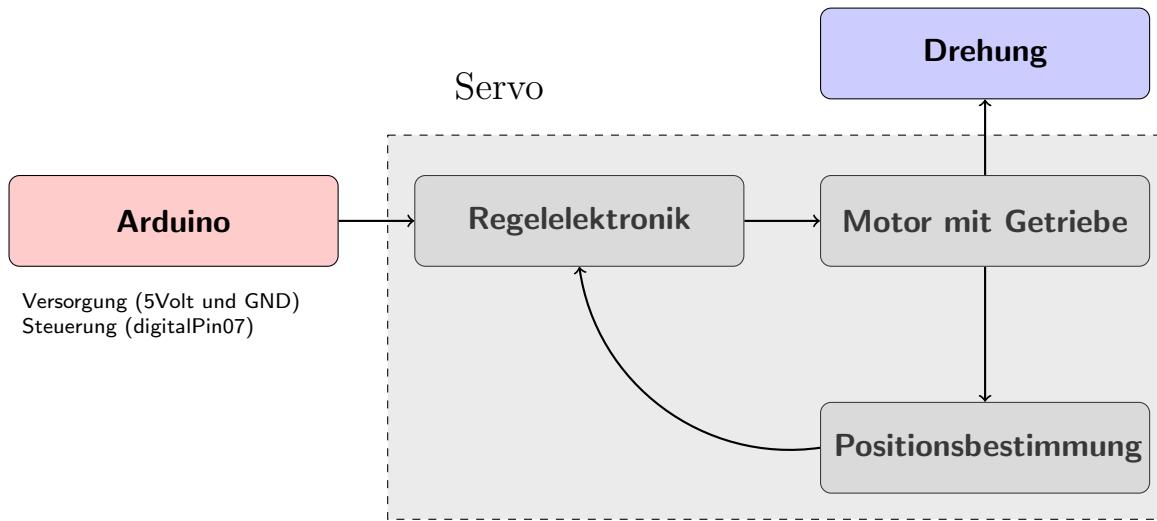
- Elektronik für die Pulsauswertung
- Elektronik für eine Regelschleife
- Leistungselektronik zur Ansteuerung eines Motors
- der Motor samt Getriebe
- Positionsauswertung

Aus diesen Komponenten wird eine Regelschleife gebildet, so dass der Motor einem Positionssignal nachgeführt wird. Am Motor ist ein Getriebe angeflanscht, welches wiederum die Abtriebsscheibe bewegt, an der die Bewegung mechanisch abgegriffen werden kann.

Der Positionsencoder, im Regelfall ein mechanisch mit dem Getriebe gekoppeltes Potentiometer, stellt die Positionsinformation wieder der Regelelektronik zur Verfügung, die bei Abweichungen entsprechende Motorbewegungen veranlasst. Man gibt also einem Servo nicht eine Bewegung vor, sondern eine Position die es ansteuern soll. Die Regelelektronik fährt dann diese Position an und hält sie in weiterer Folge.

### 3.6.1. Pulsweitenmodulation PWM

Ein Servo wird mit einem Pulsweiten modulierten Signal (PWM-Signal) angesteuert. Die Information über den Winkel ist in der Länge der Pulse enthalten. Der zeitliche Abstand zwischen zwei positiven Pulsflanken bleibt dabei unverändert. In der Abbildung 3.15 sieht man, wie die Pulsweite den Winkel beeinflusst.



**Abbildung 3.14.:** Blockdiagramm eines Servos, der von digitalen Pin07 gesteuert wird

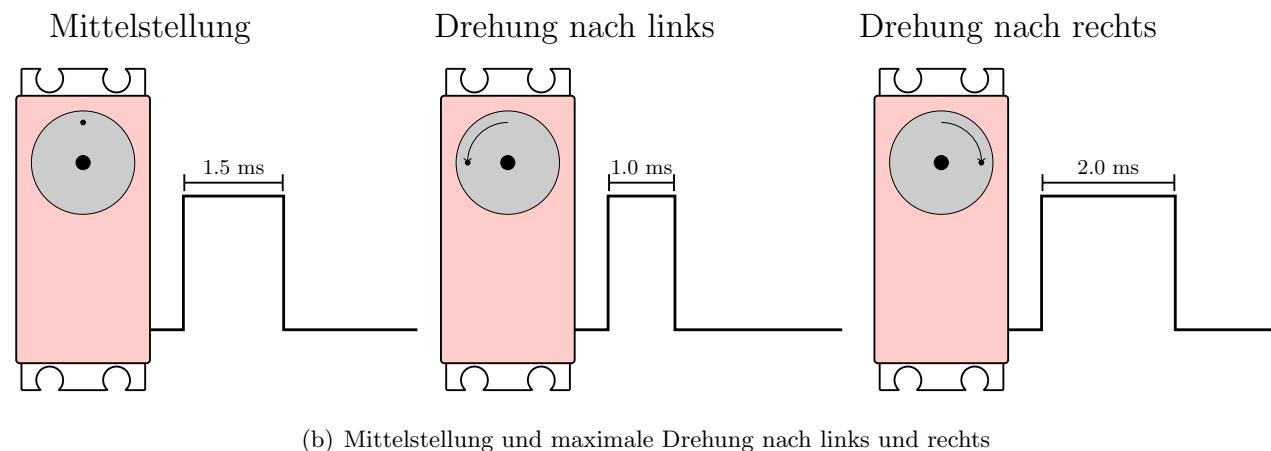
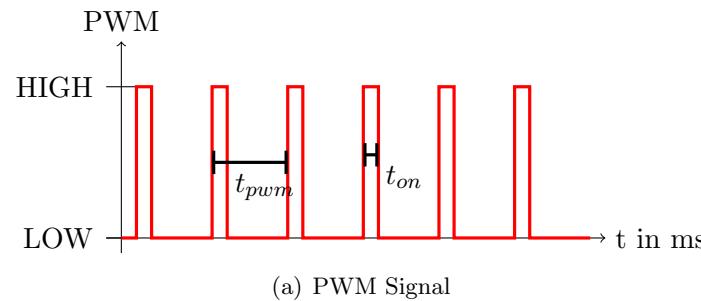
Die Zeiten ‘t periode’ und ‘t on’ bestimmen den Servowinkel. In der Tabelle 3.2 stehen die Parameter die von einem Standard-Servo verarbeitet werden. Mit diesen Parametern ist ein Servo aber noch nicht vollständig ausgesteuert. Generiert man die Signale selber kann man noch mehr Drehwinkel aus dem Servo herausholen. Die maximalen Werte müssen für jeden Servotyp experimentell bestimmt werden. Man tut dies, indem man ein PWM-Signal generiert das den Servo knapp an den mechanischen Endanschlag stossen lässt.

### 3.6.2. Aufbau der Schaltung

Gab es früher je nach Herstellerfirma unterschiedliche Stecksysteme, so hat sich im Laufe der Zeit der sog. Uni-Anschluss durchgesetzt. Elektrisch ist der Uni-Stecker so aufgebaut, dass er das in der Elektronik übliche 2.54mm Rastermaß benutzt. Er passt also problemlos auf unser Breadboard.

Dieser Stecker ist mit einem 3-poligen Flachband-Kabel mit der eigentlichen Servoelektronik verbunden. Gebräuchlich sind einige verschiedene Farbschemen bei diesen Kabeln:

- schwarz - rot - weiß



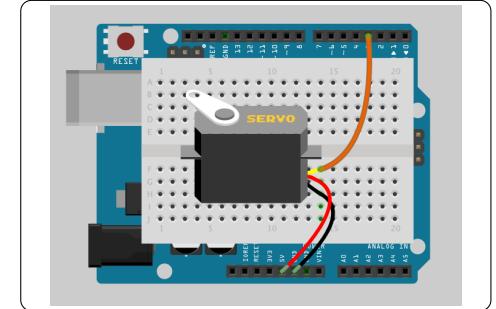
**Abbildung 3.15.:** Der Zusammenhang zwischen Pluslänge des PWM Signals und Drehwinkel des Servus

Servo Type	t periode	t on min	t on max
Standard-Servo	20ms	1ms	2ms
Servo Futaba S3003	20ms	0,58ms	2,4ms
Modellcraft VSD-5E-HS (digital)	10ms	0,5ms	2,3ms

**Tabelle 3.2.:** Parameter für verschiedene Servos

- schwarz - rot - gelb
- braun - rot - orange
- schwarz - rot - blau

Das schwarz Kabel ist meistens GND und das rot Kabel die Spannungsversorgung (5Volt). Die dritte Leitung (weiß, gelb, orange, blau, ...) ist die Signalleitung, über die das Servo mit Pulsen versorgt wird, welche ihm die anzufahrende Position mitteilen. Wenigstens in einem Punkt sind sich aber alle Hersteller einig: Die Versorgungsspannung wird immer über die mittlere der 3 Adern des Flachbandkabels geführt, die auch immer rot ausgeführt wird.



**Abbildung 3.16.:** Servomotor

### 3.6.3. Der Beispieldsketch

Mit dem folgenden Sketch (siehe Listing 3.3) kann ein Servo betrieben werden ohne Hinzunahme von Bibliotheken. Dies kann von Vorteil sein, wenn man Platz auf dem Controller sparen möchte. Es ist hier darauf zu achten, dass im folgenden Sketch der digitale Pin07 als Signal Pin für den Servo verwendet wird. Der Servo sollte in diesem Fall also auch entsprechend angeschlossen sein.

Beim Ausführen des Sketchs bewegt sich der Servo einmal bis zum Anschlag (180 Grad) und dreht wieder zum entgegengesetzten Anschlag zurück. Das Ganze wiederholt sich anschließend.

In der Zeile 17 wird die aktuelle Position von Grad in Millisekunden umgerechnet. Dabei entspricht ein Winkel von  $0^\circ$  einer PWM-Plusweite von 200ms. Ein Winkel von  $180^\circ$  einer PWM-Pulsweite von 20ms.

---

```

1 int servoPin = 3; /
2 int pwm;
3 int pos;
4
5 void setup() {
6     pinMode(servo, OUTPUT);
7 }
8
9 void loop() {
10    for(pos=0; pos<180; pos++) {
11        servoMove(servoPin, pos);
12    }
13 }
14
15 void servoMove(int servo, int pos){
16     // Winkel in Mikrosekunden umrechnen
17     pwm = (pos * 11) + 200;
18     digitalWrite(servo, HIGH);
19     delayMicroseconds(pwm); // Dauer des PWM-
        Pluses
20     digitalWrite(servo, LOW); // Servo Pin
        auf LOW Ende des PWM-Pluses
21     delay(20); // 20 ms warten
22 }
```

Mit dem Wert 0 bis 255 wird der Servowinkel gesteuert

Die Funktion servoMove() berechnet aus der Winkelangabe (pos) die Dauer des PWM-Signal in Millisekunden und erzeugt anschliessend das PWM-Signal.

Servo Pin auf HIGH Beginn des PWM-Pluses

**Listing 3.3:** PWM Singnal wird erzeugt

### 3.6.4. Aufgaben

#### Aufgabe 1

Baue die Schaltung in Abb. 3.16 nach und teste dann die zwei verschiedenen Arten (Listing 3.3 und 3.4) einen Servo zu steuern.

#### Aufgabe 2

Eine etwas einfachere Lösung zeigt Listing 3.4. Hier wird auf die in der Arduino IDE enthaltene Servo Library zurückgegriffen.

---

### 3.6.5. Die Servo Library

Das Beispiel findet sich auch in der IDE unter File –> Examples –> Servo. Achtung: bei Listing 3.4 wird der digitale Pin09 verwendet.

```
1 #include <Servo.h>
2
3 Servo myservo; // create servo object to control a servo
4 // a maximum of eight servo objects can be created
5
6 int pos = 0; // variable to store the servo position
7
8 void setup()
9 {
10    myservo.attach(9); // attaches the servo on pin 9 to the servo object
11 }
12
13
14 void loop()
15 {
16    for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
17    {
18        myservo.write(pos); // tell servo to go to position in variable 'pos'
19        delay(15); // waits 15ms for the servo to reach the position
20    }
21    for(pos = 180; pos>=1; pos--) // goes from 180 degrees to 0 degrees
22    {
23        myservo.write(pos); // tell servo to go to position in variable 'pos'
24        delay(15); // waits 15ms for the servo to reach the position
25    }
26 }
```

**Listing 3.4:** Testsketch für die Servo-Library

---

### **3.6.6. Aufgaben**

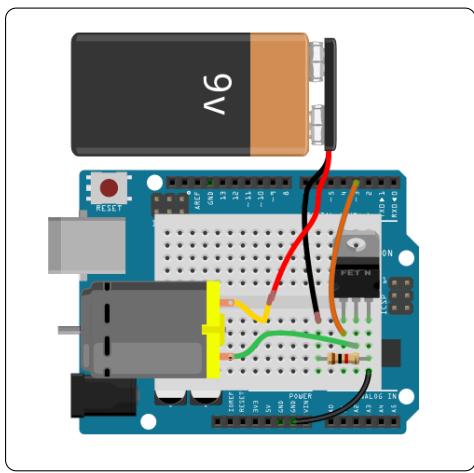
**Aufgabe 1**

**Aufgabe 2**

Verändere beide Beispiel-Sketche so, dass sich der Servo um  $90^\circ$  dreht. Verändere in Listing 3.3 die Zeile 17 entsprechend.

---

### 3.7. C Sprachstruktur: Library



### 3.8. DC-Motoren und externe Spannungsquelle

DC-Motoren bestehen auf Dauermagneten und Spulen. In der Spule wird beim durchfließen des elektrischen Stromes ein Magnetfeld induziert (erzeugt). Dieses induzierte Magnetfeld und das Magnetfeld des Dauermagneten stoßen einander ab. Durch Schreibkontakte wird der Stromfluss durch die Spule an- bzw. aufgeschaltet. Dadurch entsteht die Drehung des DC-Motors. Je stärker der Stromfluss ist, desto schneller dreht sich der DC-Motor. Ein normaler DC-Motor benötigt mehr als 100 mA. Deshalb reicht die max. mögliche Strom der ein digitaler Port (40mA) nicht aus um einen DC-Motoren zu betreiben. Mit zwei Motoren kommt man auch schnell an die Grenzen des gesamten Arduino-Boards. Das Arduino-Board (Steuer-Stromkreis) wird deshalb nur als Schalter für einen Externen Stromkreis (Arbeit-Stromkreis) benutzt. Die Steuerung erfolgt mit Hilfe eines n-FET Transistors.

Abbildung 3.17.: DC-Motor mit externer Spannungsquelle

---

### 3.9. Integrierte Schaltkreise, IC

Viele Schaltungen oder Schaltungsteile kommen in der praktischen Elektronik immer wieder vor. Um diese, teilweise komplexen, Schaltungen nicht immer wieder neu aufbauen oder erfinden zu müssen, werden sie in integrierte Schaltungen (IS = Integrierter Schaltkreis) zusammengefasst und in einem Gehäuse vergossen.

Integrierter Schaltungen sind preisgünstig, da als Massenprodukt gefertigt werden. Durch ihren kompakten Aufbau sind die zudem platzsparend und betriebssicher.

Vor der Entwicklung integrierter Schaltungen Ende der 1950er wurden elektrische Schaltungen mit diskreten Bauteilen aufgebaut, d. h. mit einzelnen Transistoren, Diode, etc., welche auf einer Leiterplatte zu einer Schaltung zusammengefügt wurden. Dies war in Größe und Lebensdauer bereits ein wesentlicher Durchbruch gegenüber den damals konkurrierenden Elektronenröhren. Die Vorteile durch den Einsatz von Transistoren und Leiterplatten (Platinen), wie Verkleinerung und geringere Leistungsaufnahme, verdrängten die Systeme aus Elektronenröhren zunehmend. Dieser Trend verstärkte sich mit der Entwicklung und dem massiven Einsatz von integrierten Schaltungen ab den 1960ern.

Der erste integrierte Schaltkreis (ein Flipflop) wurde im September 1958 von Jack Kilby entwickelt. Er bestand aus zwei Bipolartransistoren, welche auf einem Germanium-Substrat befestigt und durch Golddrähte verbunden wurden. Dieser Hybrid-Schaltkreis ist somit ein erstes Beispiel der Umsetzung der schon bekannten Transistor-Transistor-Logik (TTL) auf einen Schaltkreis. Sie war eine Vorstufe zur Weiterentwicklung der TTL-Schaltungen hin zu kleineren Bauformen. Schaltkreise aufgebaut aus solchen einfachen integrierten Schaltkreisen steuerten die Apollo Raumschiffe (siehe Abb. 3.18(c)).

Der erste „monolithische“, d. h. aus bzw. in einem einzigen einkristallinen Substrat gefertigte, integrierte Schaltkreis wurde von Robert Noyce im Juli 1959 zum Patent angemeldet. Das Entscheidende an Kilbys Erfindung war die komplette Fertigung der Bauelemente und Verdrahtung auf einem Substrat. Für die Herstellung wurden bereits fotolithografische Verfahren und Diffusionsprozesse genutzt.

Die ersten integrierten Schaltkreise in Serienproduktion entstanden bereits Anfang der 1960er und bestanden lediglich aus bis zu wenigen Dutzend Transistoren (siehe Abb. 3.18(b)). Mit den Jahren wurden die Bauelemente jedoch immer weiter verkleinert und auch passive Bauelemente wie Widerstände integriert sowie die Komplexität der integrierten Schaltkreise weiter erhöht. Damit erhöhte sich auch die Anzahl der Transistoren pro Chip beziehungsweise pro Chip-Fläche, dabei war die Anzahl der Transistoren die wichtigste Kenngröße von ICs.

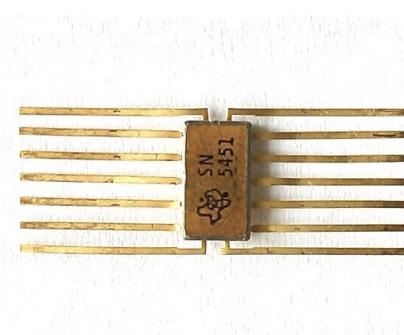
---

<sup>2</sup>Dual In-Line (DIL), Gehäuse mit Anschlüssen an zwei Seiten, meist im Raster 2,54 mm (=100 mil), die „Urform“ der Chipgehäuse

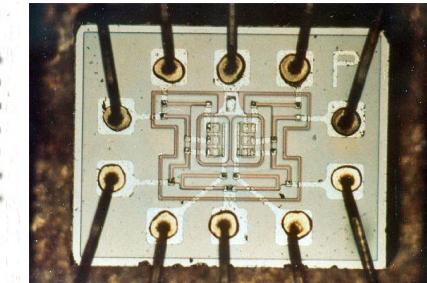
<sup>3</sup>Quelle: <http://www.ti.com/corp/docs/kilbyctr/downloadphotos.shtml>



(a) IC in DIL/DIP<sup>1</sup>



(b) TI SN5451, der erste kommerzielle IC



(c) Logisches NOR: IC aus dem PC, der das Apollo Raumschiff steuerte<sup>3</sup>

Abbildung 3.18.: IC im Gehäuse, der erste kommerzielle IC und der erste funktionsfähige IC

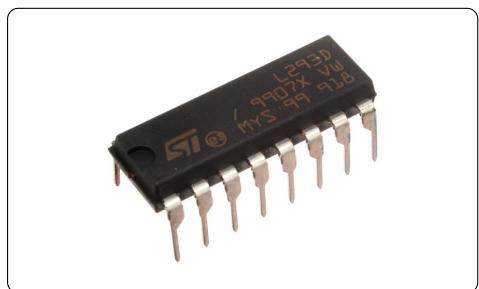


Abbildung 3.19.: IC L293D

## 3.10. Der IC L293D zur Motorsteuerung

### 3.10.1. Die Brückenschaltung

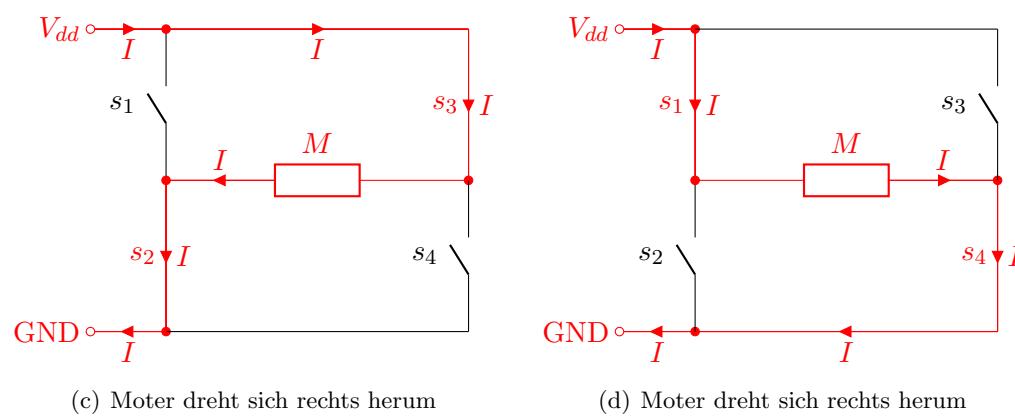
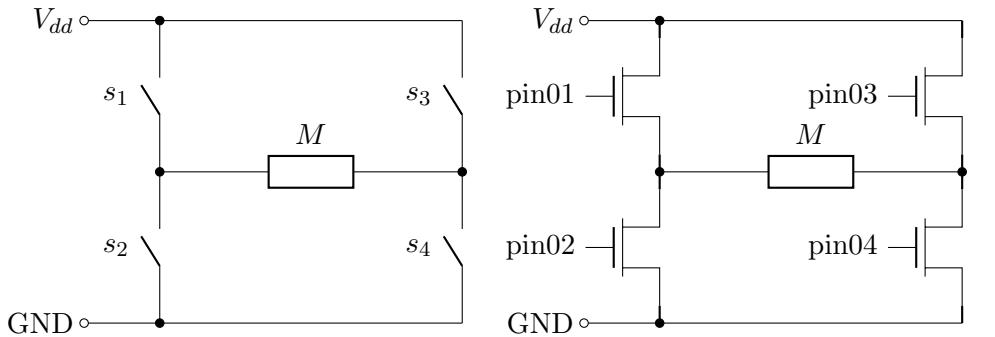
Wenn bei einem DC-Motor die Laufrichtung umgekehrt werden soll, so muss die Richtung des elektrischen Stroms der den Motor durchfließt umgekehrt werden. Der einfachste Weg dies zu erreichen ist eine sogenannte H-Brückenschaltung. In der Abb. 3.20 ist schematisch eine H-Brückenschaltung mit vier Schaltern dargestellt.

In der Tabelle 3.4 sind die möglichen Schalterstellungen und das entsprechende Verhalten des Motors aufgelistet. Wenn nun diese vier Schalter durch FET-Transistoren ersetzt werden, ist es möglich mit Hilfe von vier digitalen Pins, die jeweils über einen Spannungsteiler mit dem Gate-Eingang der FETs verbunden werden eine H-Brückenschaltung zu realisieren, bei der mit Hilfe der digitalen Pins mit Motordrehrichtung gesteuert werden kann (vergleiche Abb. 3.20(b)).

---

$s_1$	$s_2$	$s_3$	$s_4$	Verhalten
auf	auf	auf	auf	stop
auf	zu	zu	auf	links
zu	auf	auf	zu	rechts
zu	zu	auf	auf	Kurzschluss
auf	auf	zu	zu	Kurzschluss

**Tabelle 3.3.:** Mögliche Schalttereinstellungen bei der Brückenschaltung



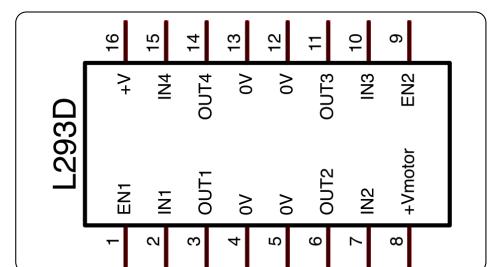
**Abbildung 3.20.:** Brückenschaltung mit vier Schaltern bzw. vier FET-Transistoren

“IN1”	“IN2”	“EN1”	“OUT1”	“OUT2”	entspricht
0V	0V	5V	-	-	Motor kurzschließen (Bremsen)
0V	5V	5V	-	+	Motor dreht vorwärts
5V	0V	5V	+	-	Motor dreht rückwärts
5V	5V	5V	+	+	Motor kurzschließen (Bremsen)
egal	egal	0V	hochohmig	hochohmig	Motor aus (Verbindung trennen, nicht kurzschließen)

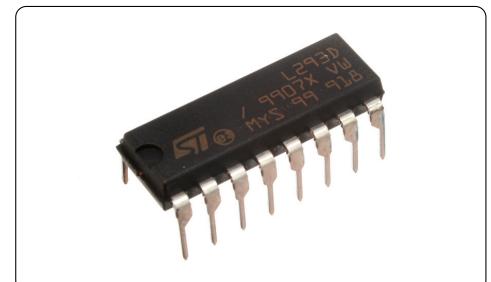
**Tabelle 3.4.:** Übersicht der Pinbelegung für einen Motor (1,2) analog für Motor (3,4).

### 3.10.2. Pinbelegung des IC L293D

Zum Glück müssen wir eine Brückenschaltung wie sie Abb. 3.20(b) ohne Spannungsteiler für jeden digitalen Pin zeigen nicht selber aufbauen. Diese Schaltung werden in ICs realisiert. Es gibt viele verschiedene Modelle und Marken von H-Bridge-ICs. Wir verwenden den IC L293D. Der L293D ist relativ billig und ist ohne zusätzliche Kurzschluss-Dioden verwendbar. Deshalb ist er sehr einfach zu benutzen. In der Abb. 3.21 ist die Pinbelegung dargestellt.



**Abbildung 3.21.:** Pin Belegung



**Abbildung 3.22.:** IC L293D

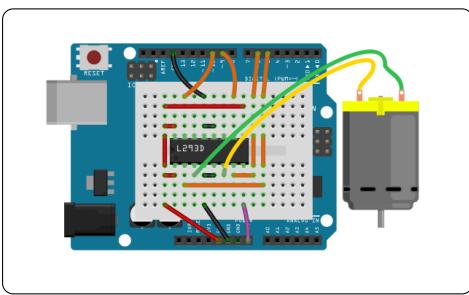


Abbildung 3.23.: L293D als Treiber für zwei DC-Motoren

### 3.10.3. Die Schaltung auf dem BreadBoard

Das schwierige bei der Schaltung für den IC L293D ist, dass sie übersichtlich aufgebaut werden muss. In der Abb. 3.23 ist ein möglicher Aufbau dieser für zwei Motoren abgebildet.

### 3.10.4. Der Beispielsketch

```
1 const int motorPin1 = 5; // IN1
2 const int motorPin2 = 6; // IN2
3
4 void setup() {
5     pinMode(motorPin1, OUTPUT);
6     pinMode(motorPin2, OUTPUT);
7 }
8
9 void loop() {
10    digitalWrite(motorPin1, HIGH);
11    digitalWrite(motorPin2, LOW);
12    delay(3000);
13
14    digitalWrite(motorPin1, LOW);
15    digitalWrite(motorPin2, HIGH);
16    delay(3000);
17
18    digitalWrite(motorPin1, HIGH);
19    digitalWrite(motorPin2, LOW);
20    delay(3000);
21
22    digitalWrite(motorPin1, LOW);
23    digitalWrite(motorPin2, LOW);
24 }
```

Die Motor PINs, wichtig ist, dass PWM-fähige PINs verwendet werden.

Motor dreht sich rückwärts

Motor stoppt

Motor dreht sich vorwärts

Motor stoppt

Listing 3.5: Testsketch für einen DC-Motor

### 3.10.5. Aufgaben

Bau die Schaltung entsprechend Abb. 3.23 auf. Lade anschließend den Beispiel-Sketch 3.6 auf deinen Arduino.

---

## Aufgabe 1

Verringere nun die Drehzahl deines Motors mit Hilfe eines PWM-Signals. Dazu musst du die digitalWrite-Befehle durch analogWrite ersetzen.

## Aufgabe 2

Entwickle für die drei verschiedene Zustände des Motor Routinen motorBackward, motorStop und motorForward. Übergabe Parameter sollen die PINs mit denen der Motor verbunden ist und ein PWM-Signal sein.

```
1 void motorBackward(pin1, pin2, pwm) {  
2     // Code  
3 }
```

### Info:

#### Syntax:

analogWrite(pin, value)

#### Parameters:

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

## Aufgabe 3 (Projekt)

Im Beispiel-Sketch habe dich den Befehl delay() benutzt um den Motor eine definierte Zeit zu bewegen. Bau auf deinem Breadboard noch einen Push-Button hinzu und verwende die Methode aus Kapitel 3.2. Wenn der Push-Button gedrückt wird, soll der Motor stoppen.

## 3.11. Schrittmotoren

Bisher haben wir sog. Gleichstrom-Motoren verwendet. Wird ein Gleichstrom-Motor mit einer passenden Betriebsspannung versorgt, so fängt er sich sofort an zu drehen. Die Drehzahl hängt dabei von der Leistung der Spannungsversorgung ab. Wenn eine Batterie zum Betrieb verwendet wird dreht sich der Motor langsamer, wenn die Batterie langsam leer wird. Bei spielerischen Einsatzbereichen ist aber eine genaue Kontrolle der Bewegung nötig.

Um einen Schrittmotor zu steuern benötigst du ebenfalls einen Treiber-Chip. Hier kann auch der L293D verwendet werden.

### 3.11.1. Funktionsweise eines Schrittmotors

#### Beispiel-Sketch

```
1 #include <Stepper.h>
2
3 #define STEPS 100
4
5 Stepper stepper(STEPS, 8, 9, 10, 11);
6
7 int previous = 0;
8
9 void setup(){
10   stepper.setSpeed(30);
11 }
12
13 void loop(){
14   int val = analogRead(0);
15   stepper.step(val - previous);
16   previous = val;
17 }
```

Die Stepper Bibliothek wird eingebunden

Anzahl der Schritte pro Umdrehung wird definiert

Ein Objekt Stepper wird erzeugt und mit den PINs verbunden

Setzt die Rotationsgeschwindigkeit auf Umdrehungen pro Minute

Sensorwert zur Drehung des Motor wird gemessen (Potentiometer)

Der Stepper wird je nach Messwertwert gedreht.

**Listing 3.6:** Testsketch für einen DC-Motor

---

### 3.11.2. Schrittmotor mit dem Motortreiber A4988

Wesentlich Komfortabler geht es mit Hilfe des Schrittmotor Treibers A4988 und der zugehörigen Library.

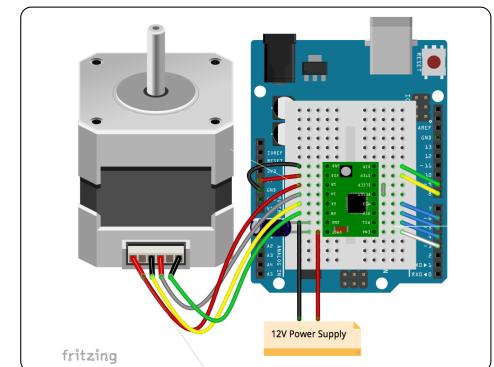


Abbildung 3.24.: nema 17

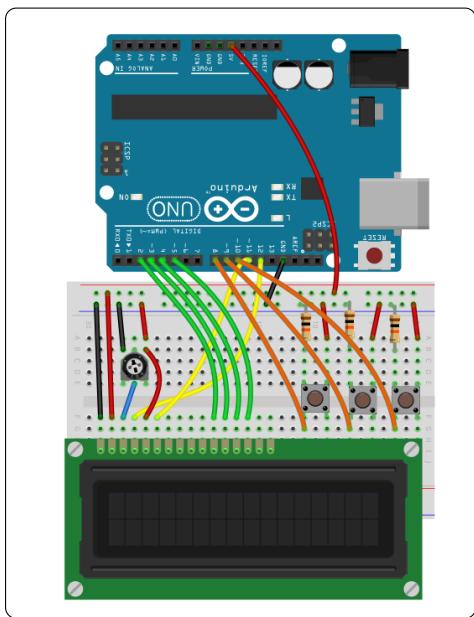


Abbildung 3.25.: LCD

## 3.12. Display

### 3.12.1. Der Beispiel-Sketch

```
1 #include <LiquidCrystal.h>
2
3 LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
4
5 void setup() {
6     lcd.begin(16, 2);
7     // :
8     lcd.print("hello, world!");
9 }
10
11 void loop() {
12     lcd.setCursor(0, 1);
13     // :
14     lcd.print(millis()/1000);
15 }
```

Die LiquidCrystal Bibliothek wird eingebunden

PINs für das LCD-Display

Setup des LCD's mit Anzahl der Spalten und Zeilen.

Schreibt eine Nachricht auf das LCD.

Setzt den Cursor auf die 0. Spalten in der 1. Zeile. (Beachte: Zeilen und Spalten werden von 0 an gezählt).

Schreibt die Anzahl der Sekunden seit dem letzten 'reset' auf das Display.

Listing 3.7: LCD-Display

## 4 — Projekte mit dem Arduino

Design is not just what it looks like and feels like.  
Design is how it works

---

Steven „Steve“ Paul Jobs (24. Februar 1955 - 5. Oktober 2011), US-amerikanischer Unternehmer

Don't undertake a project unless it is manifestly important and nearly impossible

---

Edwin Herbert Land, (May 7, 1909 – March 1, 1991), American scientist and inventor

---

## **4.1. Projektmanagement**

---

## 4.2. Das optische Tee-Tassen-Thermometer ( $oT^3$ )

### 4.2.1. Anforderungen

Anforderungen	leicht	mittel	schwer
Schaltung			
Programm			
Aufbau			

Tabelle 4.1.

---

## **4.3. Das eigene Motor Arduino Shield**

### **4.3.1. Anforderungen**

Anforderungen	leicht	mittel	schwer
Schaltung			
Programm			
Aufbau			

**Tabelle 4.2.**

### **4.3.2. Projektbeschreibung**

---

## 4.4. SchulBot ein autonomes Fahrzeug

### 4.4.1. Anforderungen

Anforderungen	leicht	mittel	schwer
Schaltung			
Programm			
Aufbau			

Tabelle 4.3.

### 4.4.2. Projektbeschreibung

Teilautonom oder autonom gesteuerte Fahrzeuge spielen eine immer größer werdende Rolle in unserer modernen Gesellschaft. In moderne Autos steuern Mikrocontroller Fahrerassistenzsysteme wie z.B. ABS. Diese Technik wird zur Zeit weiterentwickelt. Im Mai 2012 erhielt Google die erste Zulassung eines autonomen Fahrzeugs in den USA. So erlaubte der US-Bundesstaat Nevada den Test des selbstfahrenden Autos auf öffentlichen Straßen. Bedingung ist jedoch, dass sich eine Person hinter dem Steuer befindet, die notfalls eingreifen kann.

Im Bereich der Transportfahrzeuge gibt es schon seit längerem automatisierte fahrerlose Fahrzeuge. Fahrerlose Transportsysteme (FTS) sind innerbetriebliche, flurgebundene Fördersysteme mit automatisch gesteuerten Fahrzeugen, deren primäre Aufgabe der Materialtransport, nicht aber der Personentransport ist. Sie werden innerhalb und außerhalb von Gebäuden eingesetzt und bestehen im Wesentlichen aus folgenden Komponenten:

- einem oder mehreren Fahrerlosen Transportfahrzeugen
- einer Leitsteuerung
- Einrichtungen zur Standortbestimmung und Lage erfassung
- Einrichtungen zur Datenübertragung



Abbildung 4.1.: Google autonomous car



**Abbildung 4.2.:** Fahrerlose  
Transportsystem als  
Unterfahrschlepper

- Infrastruktur und peripheren Einrichtungen.

#### 4.4.3. Aufgabenstellung

Ihr sollte ein autonomes Roboterfahrzeug planen, bauen und programmieren. Das Fahrzeug soll folgende Aufgaben lösen.

##### Aufgabe 1

Das Fahrzeug soll mindestens 10 Meter gerade aus fahren und dabei maximal 1 Meter nach links oder rechts abweichen.

##### Aufgabe 2

Euer Fahrzeug soll mit Hilfe zweier Schalter Wände erkennen und dieses Ausweichen.

##### Aufgabe 3

Euer Fahrzeug soll mit Hilfe eines Optokopplers eine schwarze Linie erkennen und dieser folgen.

#### 4.4.4. Material

Vorgefertigter Bauplan, zwei Getriebemotoren, Arduino Uno mit ProjektShield, Batteriehalter, Grundplatte

#### Fertigungstechniken

CNC Fräsen, Löte,

#### 4.4.5. Erstellung des Bauplans

Mit Hilfe der CAD/CAM Software nccad plant und entwerft ihr auf Grundlage der Vorlage mcgbot.cad die Bodenplatte eures Fahrzeuges.

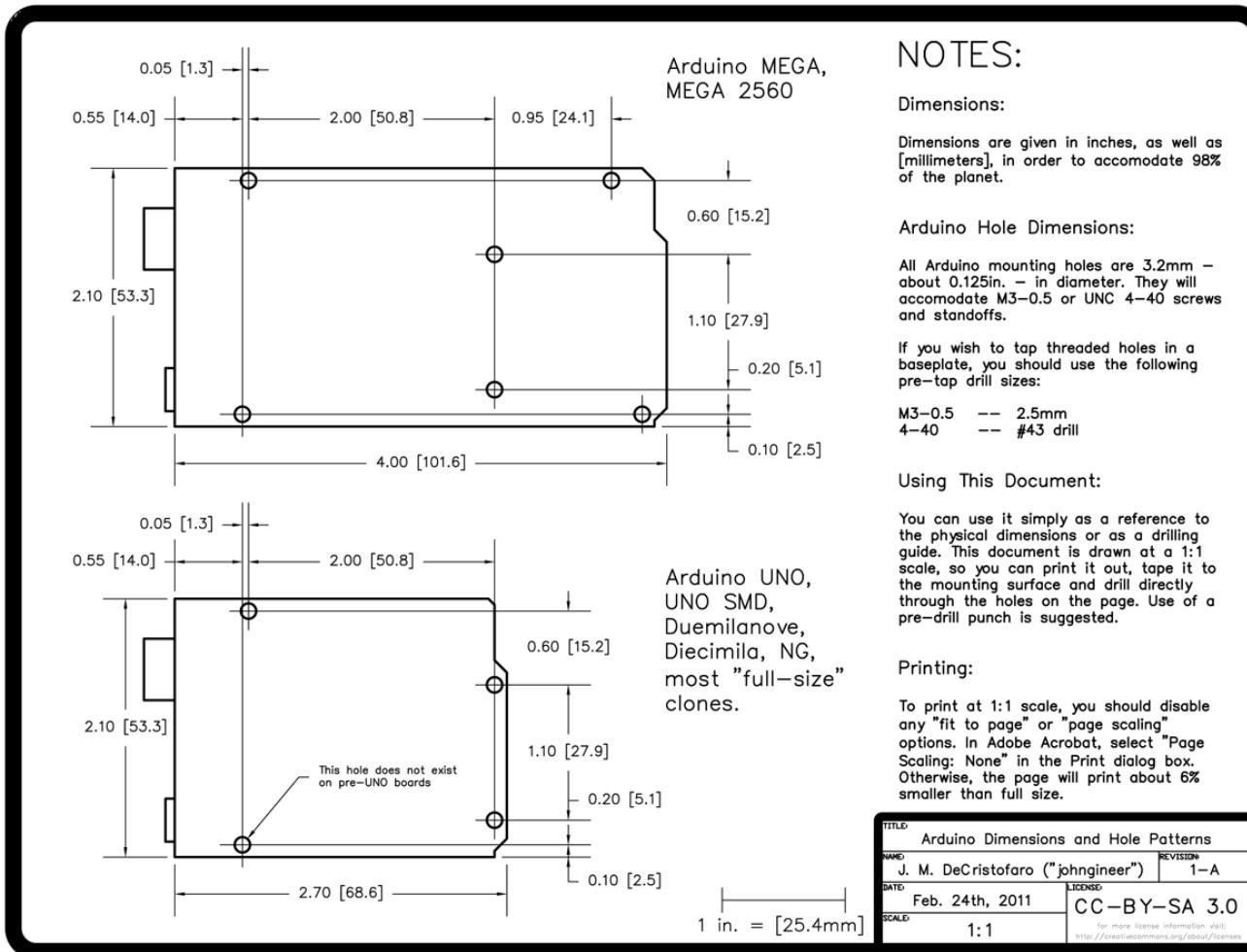


Abbildung 4.3.: default

---

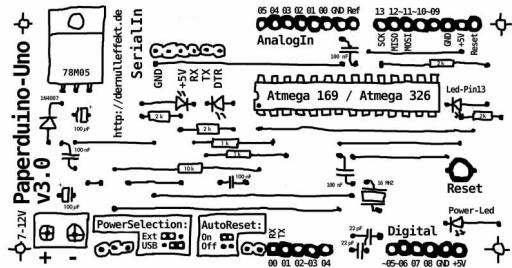
## **4.5. Paperduino-UNO, das selbstgebaute Arduino-Board**

### **4.5.1. Anforderungen**

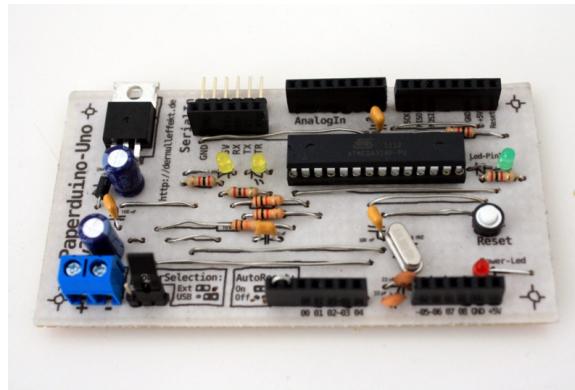
Anforderungen	leicht	mittel	schwer
Schaltung			
Programm			
Aufbau			

**Tabelle 4.4.**

### **4.5.2. Projektbeschreibung**



(a) Schaltplan



(b) Fertige Platine

**Abbildung 4.4.: default**

## 4.6. Roboter Arm

### 4.6.1. Anforderungen

### 4.6.2. Projektbeschreibung

---

Anforderungen	leicht	mittel	schwer
Schaltung			
Programm			
Aufbau			

**Tabelle 4.5.**

## **5 — Die Stationskarten**

---

---

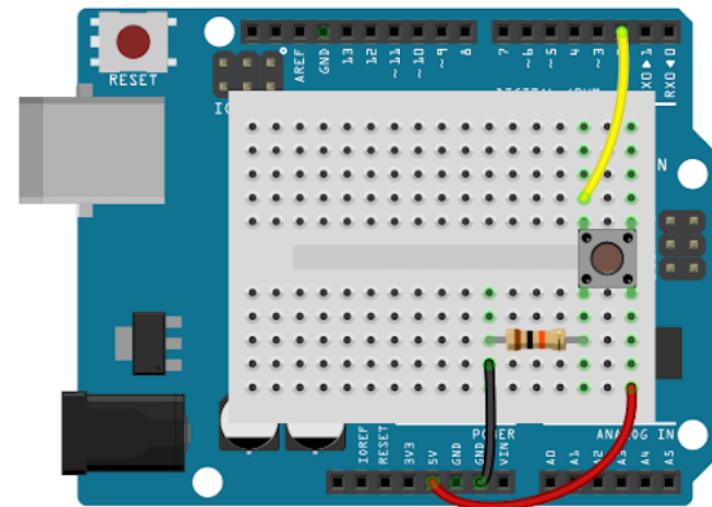
## Stationskarte: 1.6 Mit dem PC kommunizieren

Info:

### Material-Liste

- Arduino Uno mit Breadboard
- Pushbottum
- Widerstand ( $10\text{k}\Omega$ )

### Versuchsaufbau

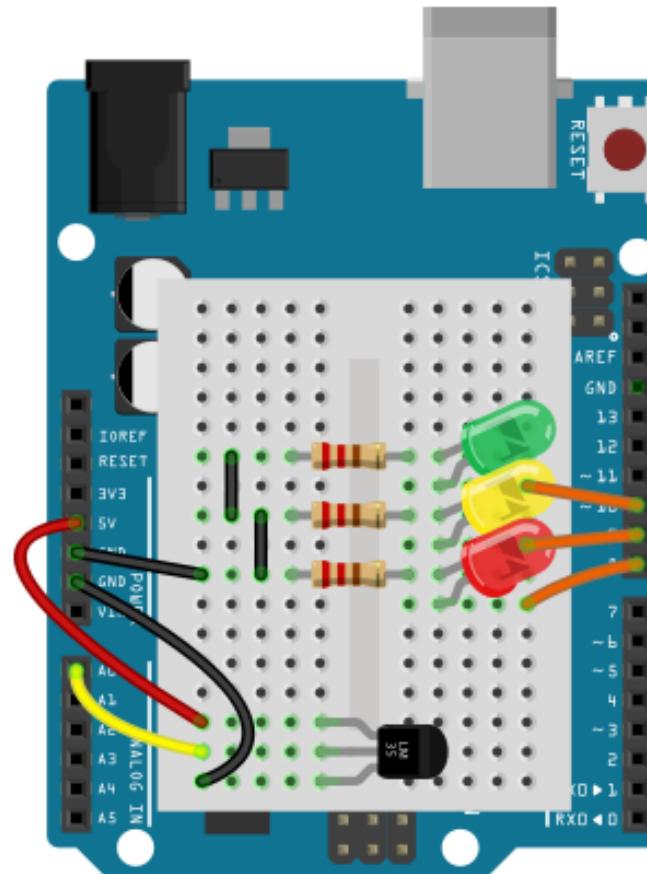


## Stationskarte: 2.1 Analoger Sensor: Der Temperatur-Sensor LM35 und LM36

### Material-Liste

- Arduino Uno mit Breadboard
- Temperatur Sensor LM35
- 3 Projekt-Kabel

### Versuchsaufbau



### Info:

Achte darauf, dass du den LM35 richtig einbaust. Sollte er sich erwärmen, musst du die Stromversorgung zum Arduino sofort unterbrechen, und den Aufbau deiner Schaltung überprüfen!

---

## Stationskarte: 2.4 Lichtsensoren

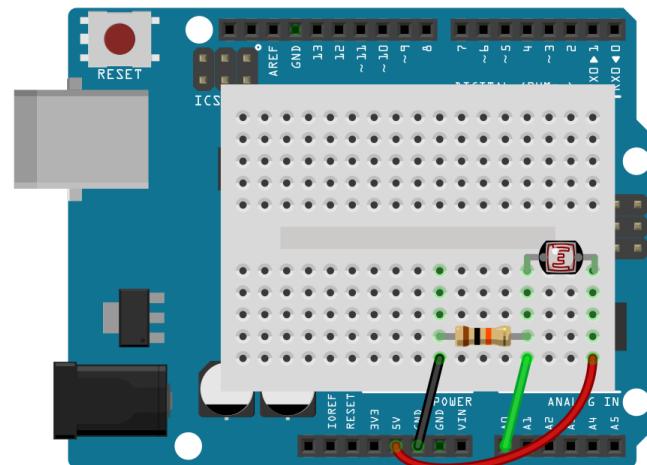
Info:

### Material-Liste

- Arduino Uno mit Breadboard
- LDR (Photozelle)
- Widerstand (muss bestimmt werden)
- Multimeter
- 3 Projekt-Kabel

### Übersicht Material

### Versuchsaufbau



## Stationskarte: 2.6 Reflexoptokoppler

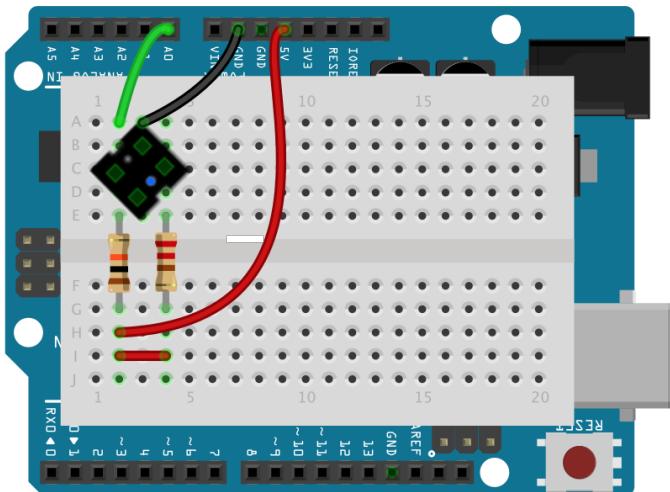
Info:

### Material-Liste

- Arduino Uno mit Breadboard
- cny70 (Reflexoptokoppler)
- Widerstände ( $220\Omega$  und  $10k\Omega$ )
- 3 Projekt-Kabel

### Übersicht Material

### Versuchsaufbau



---

## Stationskarte: 2.7 Digitaler Sensor: Der Ultraschallsensor

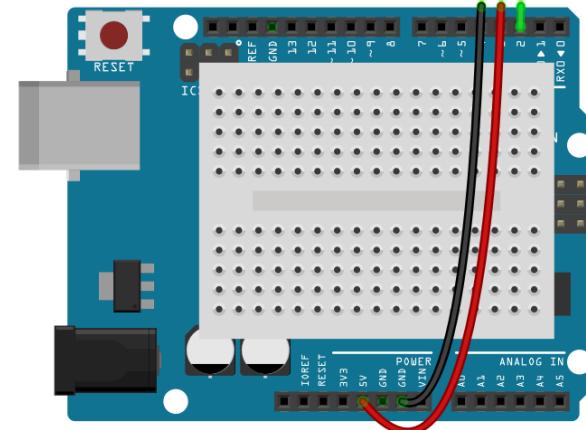
Info:

### Material-Liste

- Arduino Uno mit Breadboard
- Untraschall Sensor
- 3 Projekt-Kabel

### Übersicht Material

### Versuchsaufbau



---

## Stationskarte: 2.8 Umwelt Sensoren

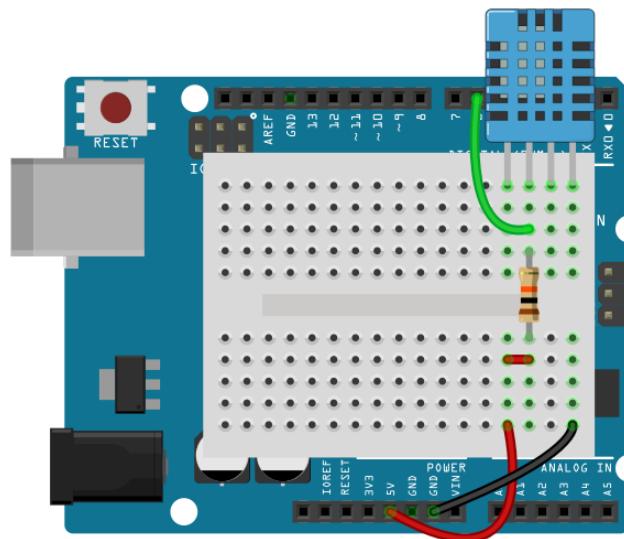
Info:

### Material-Liste

- Arduino Uno mit Breadboard
- DH11 (Temperatur und Feuchtigkeit)
- Widerstand ( $10k\Omega$ )
- 4 Projekt-Kabel

### Übersicht Material

### Versuchsaufbau



# Appendices

## A — Hintergrundwissen Arduino

---

## **A.1. Fehlersuche**

## A.2. Analoge und digitale Signale

Ein Analogsignal ist im Rahmen der Signaltheorie eine Form eines Signals mit stufenlosem und unterbrechungsfreiem Verlauf. Ein Analogsignal wird als glatte Funktion beschrieben und es lässt sich damit beispielsweise der zeitlich kontinuierliche Verlauf einer physikalischen Größe wie der Schalldruck in Form eines analogen Audiosignals beschreiben. Der Wertebereich eines Analogsignals wird als Dynamikumfang bezeichnet.

Ein Digitalsignal ist eine spezielle Form eines Signals, welches einerseits einen abgegrenzten und gestuften Wertvorrat und zudem in der zeitlichen Abfolge nur zu bestimmten periodischen Zeitpunkten definiert ist bzw. eine Veränderung im Signalwert aufweist. Es kann aus einem Analogsignal, welches den zeitlich kontinuierlichen Verlauf einer physikalischen Größe beschreibt, durch die Quantisierung und eine Abtastung, welche zu definierten Zeitpunkten erfolgt, gebildet werden. Die digitalen Werte sind üblicherweise als Binärzahlen kodiert, so dass ihre Quantisierung in Bits angegeben wird.

### A.2.1. Digital Analog Wandler

#### Aufbau der Schaltung

Der Aufbau der Schaltung ist in Abb. und Abb. A.1 zu sehen.

In der Abb. A.1 sind neben dem eigentlichen DAC-Wandler bestehend aus den neuen  $20k\Omega$  und sieben  $10k\Omega$  Widerständen ein Potentiometer, das zur Verändern der Frequenz dient und ein Summer, dessen Aufgabe es ist das Signal akustischen darzustellen.

#### Info:

digital von lat. digitus = Finger;  
mit Fingern wird gezählt

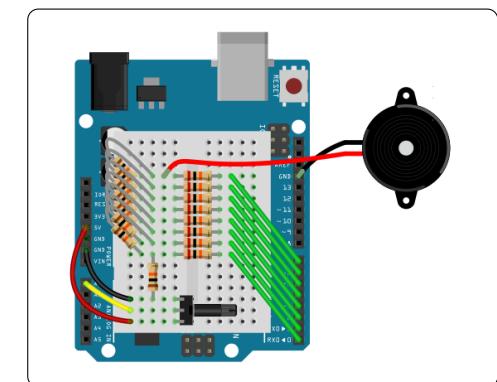


Abbildung A.1.: 8 BIT Analog Digital Wandler

---

### Beispiel-Sketch zum erzeugen eines sinusförmigen Spannungsverlaufes

```
1 int mPoti;
2
3 void setup()
4 {
5     for (int i=0;i<8;i++) {
6         pinMode(i, OUTPUT);
7     }
8 }
9
10 void loop() {
11     mPoti = analogRead(A0);
12
13     for (int i=0; i<8;i++) {
14         digitalWrite(i, HIGH);
15         delayMicroseconds(mPoti);
16         digitalWrite(i, LOW);
17     }
18
19     for (int i=6;i>0;i--) {
20         digitalWrite(i, HIGH);
21         delayMicroseconds(mPoti);
22         digitalWrite(i, LOW);
23     }
24 }
```

Digital PINs 0 bis 7 werden auf OUTPUT gesetzt

**Listing A.1:** Beispielsketch für eine sinusförmigen Spannungsverlauf

## A.3. Arduino's Verbindung zur Aussenwelt: PINs

Wie du schon weisst gibt es (neben anderen Anschlüssen) 14 digitale und 6 analoge PINs. In diesem Abschnitt erfährst du wichtiges Hintergrundwissen. Es ist wichtig, dass du diesen Abschnitt aufmerksam liest, da der Inhalt sehr wichtig für dein Verständnis für die Funktionsweise des Arduino-Boards ist. Wenn du mit dem Inhalt noch überfordert bist, kannst du auch zuerst mit Kapitel 2 anfangen, aber du solltest diesen Abschnitt unbedingt lesen und durcharbeiten. Ich habe versucht den Inhalt anhand von Beispielen und Aufgaben aufzulockern. Die Aufgaben sind der Schlüssel zum Verständnis.

### A.3.1. Digitale Ein- und Ausgänge

Die digitalen PINs des Arduino's können entweder als Ein- oder Ausgänge konfiguriert werden. Du hast beide Arten schon kennengelernt. Im Abschnitt 1.4 hast du den digitalen PIN 13 als OUTPUT-PIN und im Abschnitt 1.6 den PIN 02 als INPUT-PIN benutzt.

### A.3.2. Der INPUT-Mode

Die digitalen PINs des Arduino's sind standardmäßig als Eingänge definiert. Es ist also nicht unbedingt nötig (aber trotzdem sinnvoll) sie als Eingänge mit dem Befehl `pinMode(pinNummer,INPUT);` zu definieren. Sinnvoll deswegen, damit deinen Sketch verständlicher wird. Die digitalen PINs haben einen sehr hohe Impedanz. Dies bedeutet, dass ein kleiner elektrischer Strom ausreicht, damit der PIN seinen Zustand wechselt. Dies kann zum Beispiel für einen kapazitiven Touch-Sensor benutzt werden.

Wenn ein PIN im INPUT-Mode nicht mit einem Sensor verbunden ist, wirkt sich die Hohe Impedanz negativ aus. Wenn du diesen PIN ausliest stellen sich zufällige Ergebnisse für den Zustand an diesem PIN ein! Es findet ein Kopplung mit der in seiner Umgebung vorhandenen Elektrizität statt. Das kann dein Pulli sein, der sich elektrostatisch aufgeladen hat, oder das Laptop. In diesem Fall spricht man davon, dass der Zustand des PINs nicht definiert ist.

#### Info:

Impedanz: Die Fähigkeit einen elektrische Ladung lange zu speichern

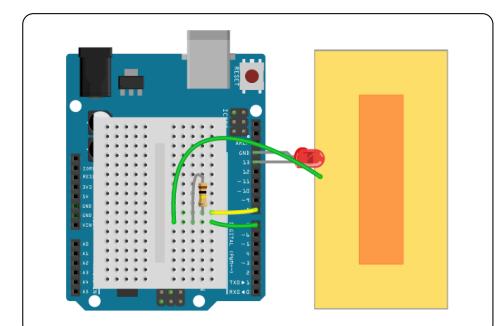


Abbildung A.2.: Touchsensor

#### Aufgabe 1: Ein Touch-Sensor

Für diese Schaltung benötigt du einen großen Widerstand ( $100k\Omega - 1M\Omega$ ), ein Krokodilklemmen-Kabel und ein Stück Metallfolie. Baue die Schaltung nach Abb. A.2 auf und lade den Beispiel-Sketch aus Listing A.2 auf deinen Arduino. Da die auf dem Arduino-Board am PIN 13 eingebaute LED von dem Projekt-Shield verdeckt wird, kannst du am PIN 13 und GND eine LED einstecken. Wenn du die Schaltung richtig aufgebaut hast, dann sollte die LED an PIN 13 leuchten. Berühre kurz die Metallfolie, dann erlischt die LED für eine Sekunde.

---

```

1 const int capPin1 = 7;          ○
2 const int capPin2 = 8;          ○
3 const int ledPin = 13;
4 int capState = 0;              ○
5
6 void setup() {
7     pinMode(ledPin, OUTPUT);
8     pinMode(capPin1, INPUT);
9     pinMode(capPin2, OUTPUT);
10 }
11
12 void loop() {
13     capState = digitalRead(capPin1); ○
14     digitalWrite(capPin2, HIGH);
15     if (capState == HIGH) { ○
16         digitalWrite(ledPin, HIGH);
17     }
18     else {
19         digitalWrite(ledPin, LOW);
20         delay(1000);
21     }
22 }
```

**Listing A.2:** Beispiel-Sketch für einen Touch-Senor

### A.3.3. INPUT-Mode und Pullup-Widerstände

Im Atmega-Chip sind  $20\text{k}\Omega$  Pullup-Widerstände eingebaut, auf die mit Hilfe von Software zugegriffen werden kann. Diese integrierten Pullup-Widerstände werden durch den Arduino-Befehl `pinMode(pinNummer,INPUT)` aktiviert. Durch die internen Pullup-Widerstände liegt an einem INPUT-PIN der HIGH-Zustand an.

Wenn nun ein Sensor mit dem auf INPUT gesetzten digitalen PIN verbunden wird, sollte der andere Ausgang des Sensors mit GND verbunden werden. Im Fall eines einfachen Schalters bewirkt dies, dass wenn der Schalter geschlossen ist der Wert LOW und wenn er geöffnet ist der Wert HIGH am digitalen PIN anliegt.

Oft ist es nützlich, einen digitalen Eingang in einen bekannten Zustand zu steuern, wenn keine Eingabe vorhanden ist. Dies kann mit Hilfe eines Pullup-Widerstand (+5 V) oder eines Pulldown-Widerstand (GND) erreicht werden. Pull-Widerstand sollte um die  $10\text{k}\Omega$  groß sein.

---

## Aufgabe 2

Ziel dieser Aufgabe ist es das Verhalten eines Pullup- und Pulldown-Widerstandes zu testen. Baue die Schaltung aus Abb. A.3 auf. Der digitale PIN 03 wird mit einem  $10k\Omega$ -Widerstand mit +5 V verbunden. Der digitale PIN 05 wird mit einem  $10k\Omega$ -Widerstand mit GND verbunden. Mit Hilfe des Sketches A.4 kannst du den jeweiligen Zustand auslesen.

```
1 int pullDown = 3; ○
2 int pullUp = 5;
3
4 int valuePullDown = 0; ○
5 int valuePullUp = 0;
6
7 void setup() {
8
9     Serial.begin(9600); ○
10
11    pinMode(pullDown, OUTPUT);
12    pinMode(pullUp, OUTPUT);
13 }
14
15 void loop() {
16
17     valuePullDown = digitalRead(pullDown); ○
18     valuePullUp = digitalRead(pullUp);
19
20     Serial.print("Wert am PIN 3: ");
21     Serial.println(valuePullDown);
22     Serial.print("Wert am PIN 5: "); ○
23     Serial.println(valuePullUp);
24
25     delay(200);
26 }
```

Definition der Pull-PINs

Speichern der aktuellen Werte

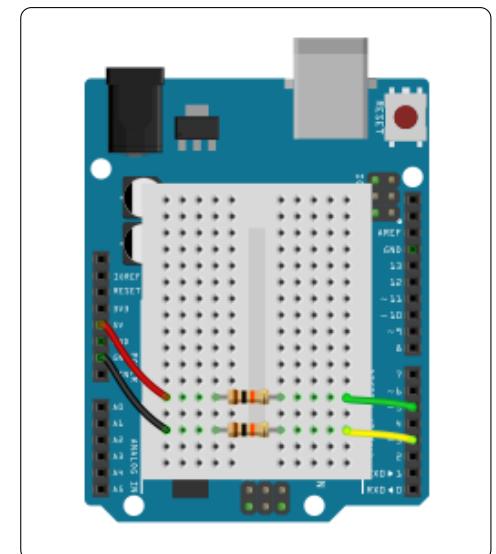
Starten der seriellen Verbindung

Deklaration der PINs als OUT-  
PUT, so dass der interne Pulldown-  
Widerstand nicht verbunden ist.

Auslesen und speichern der  
Zustände am PIN 3 und 5.

Ergebnis an PC übertragen.

**Listing A.3:** Pullup- und Pulldown-Widerstände



**Abbildung A.3.:** Pullup- und Pulldown-Widerstände

### A.3.4. PIN 13

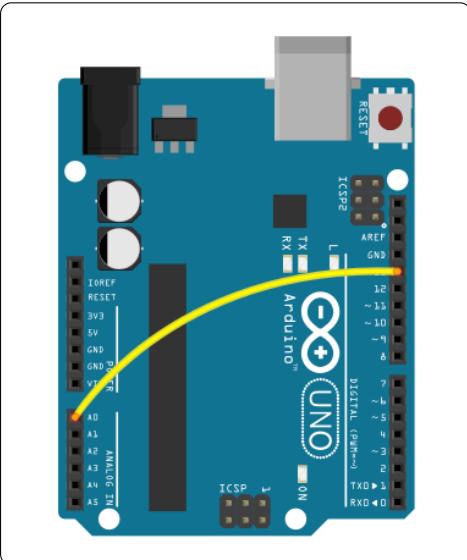


Abbildung A.4.: Potenzial am PIN 13 messen

Der digitale PIN 13 ist etwas besonders, da er mit einer LED mit Vorwiderstand verbunden ist. Wenn du diesen PIN auf INPUT setzt, d.h. seinen internen  $20k\Omega$  Pull-up-Widerstand aktivierst, wird das Potenzial bei etwa 1,7 V anstatt der erwarteten 5 V liegen. Schuld an diesem Umstand ist die LED mit ihrem Vorwiderstand, die die Spannung nach unten ziehen. Das bedeutet, dass dieser PIN immer im Zustand LOW ist! Wenn du den PIN 13 als Eingang verwenden möchtest, musst du den PIN auf INPUT setzen und einen externen Pullup-Widerstand verwenden.

#### Aufgabe 3

Für diese Aufgabe benötigst du nur das Arduino-Board. Verbinde den analogen PIN A0 mit dem digitalen PIN 13 (siehe Abb. A.4). Mit Hilfe des Sketches A.4 kannst du das Potenzial am PIN 13 messen.

```
1 int pin13 = 13;          ○ Definition der Pull-PINs
2
3 int pinPot = 0;          ○ Speichern der aktuellen Werte
4
5 void setup() {
6
7   Serial.begin(9600);    ○ Starten der seriellen Verbindung
8
9   pinMode(pin13, INPUT); ○ Deklaration der PINs
10
11 }
12
13 void loop() {
14
15   pinPot = analogRead(A0); ○ Auslesen und speichern der
16                           Zustände am PIN 3 und 5.
17
18   Serial.print("Potenzial am PIN 13: ");
19   Serial.println(pinPot/256); ○ Ergebnis an PC übertragen.
20
21 }
```

Listing A.4: Verhalten des PINs 13

### A.3.5. Digitale PINs im OUTPUT Mode

Wenn ein digitaler PIN in den Modus OUTPUT gesetzt wird, ist er in einem niederohmigen Zustand. Dies bedeutet, dass der Arduino sehr leicht kurzgeschlossen werden kann, indem man den PIN direkt mit GND verbindet. Ein digitaler PIN kann maximal bis zu 40mA an Strom für angeschlossene Geräte liefern. Die maximale Leistung ist groß genug um eine LED die meisten Sensoren zu versorgen, aber nicht groß genug um die Relais, Motoren, Magnetspulen zu betreiben. Diese Bauteile müssen dann mit Hilfe eines Transistor betrieben werden.

Die digitalen PINs sind sehr empfindlich. Es ist immer eine gute Idee, PINs im OUTPUT Modus mit einem Widerstand von mind.  $220\Omega$  mit GND zu verbinden. Gerade bei LEDs sinkt der Widerstand, wenn sie leiten extrem ab. Bei langem Betrieb kann sowohl die LED als auch der digitale PIN verstört werden.

#### Aufgabe 4

Digitale PINs im INPUT Mode liefern sehr wenig Strom, so dass eine LED nur schwach leuchtet. Sollte eine LED nur sehr schwach leuchten dann ist es wahrscheinlich, dass der Mode des PINs falsch gesetzt ist.

Versuche es selbst: Setzte dazu eine rote LED in PIN 13 und GND (schnelle Methode 1.12(a)) und lade den Sketch aus Listing A.5 auf deinen Arduino.

```
1 int ledPin = 13; 
2 void setup() { 
3 } 
4 
5 void loop() { 
6   pinMode(ledPin, INPUT); 
7   digitalWrite(ledPin,HIGH); 
8   delay(2000); 
9 
10  pinMode(ledPin, OUTPUT); 
11  digitalWrite(ledPin,HIGH); 
12  delay(2000); 
13 }
```

Definition des LED-PINs

INPUT-Mode

OUTPUT-Mode

Listing A.5: PIN in input und output Mode

### A.3.6. Analoge Input PINs

Das Arduino Uno Board hat 6 analoge PINs die mit A0 bis A5 bezeichnet sind. Die Hauptfunktion der analogen PINs ist natürlich das Auslesen von analogen Sensoren. Zusätzlich haben die analogen PINs alle Funktionen der so genannten GPIO-PINs (General Purpose Input/Output), das heißt sie können als zusätzliche digitale PINs verwendet werden.

#### Funktionsweise der analogen PINs: A/D-Wandler

Der ATmega-IC besitzt einen so genannten integrierten 6-Kanal-Analog-Digital-Wandler (kurz A/D-Wandler). Der A/D-Wandler hat einen Spannungsbereich von 0V bis 5 V, dieser 5 V Potenzialunterschied wird in 1024 Teile aufgeteilt was einer Auflösung von 10-Bit ( $2^{10} = 1024$ ) entspricht.

$U_{in}$	0	0.005	0.010	...	4.995	5.000
10-Bit Wert	0	1	2	...	1022	1023

Tabelle A.1.: Spannung und 10-Bit Wert

Um den Wert der gemessene Spannung auszugeben, muss der gemessene 10-Bit-Wert durch 256 geteilt werden. Dies kann mit Hilfe des Arduino-Befehles `map()` geschehen:

#### Aufgabe 5

Widerstandsmessung eines unbekannten Widerstandes. Mit Hilfe eines  $1k\Omega$ -Widerstandes soll der Wert eines unbekannten Widerstandes bestimmt werden. Dazu müssen die Widerstände in Reihe geschaltet werden. Der unbekannte Widerstand kann dann mit Hilfe der Formel

$$R_2 = \frac{U_2}{U_g - U_2} R_1$$

berechnet werden. Mit  $R_1 = 1k\Omega$  und  $U_g = 5 \text{ V}$  folgt:

$$R_2 = \frac{U_2}{5 \text{ V} - U_2} \cdot 1k\Omega$$

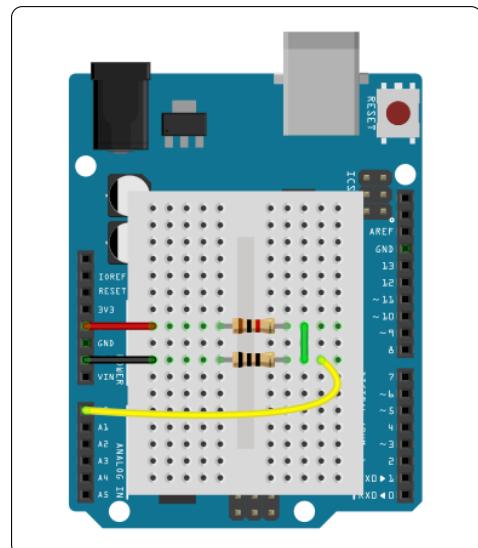


Abbildung A.5.: Widerstandsmessung

---

```

1 float senU = 0.0;   ○ Speichern der aktuellen Werte
2 float gemR = 0.0;
3
4 void setup() {
5     Serial.begin(9600); ○ Starten der seriellen Verbindung
6 }
7
8 void loop() {
9     senU = map(analogRead(A0), 0, 1024, 0, 5); ○ Messen und umrechnen auf Spannungswert.
10
11    gemR = senU/(5-senU)*1000; ○ Widerstand berechnen
12
13    Serial.print("R gemessen: ");
14    Serial.println(gemR); ○ Ergebnis an den PC senden
15
16    delay(200);
17 }
```

**Listing A.6:** Messen eines unbekannten Widerstandes

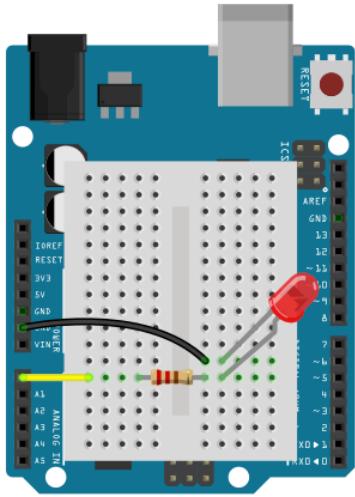
## Das PIN Mapping

Analoge PINs können auch als digitale PINs verwendet werden. Um sie von den digitalen PINs unterscheiden zu können, werden die Alias Namen A0 bis A5 verwendet. Um den analogen PIN A0 als digitalen Ausgang mit dem Wert HIGH zu definieren sind folgende Befehle nötig:

```

1 pinMode(A0, OUTPUT); ○ PIN A0 wird zu einem digitalen PIN
2 digital(A0, HIGH); ○ A0 wird auf HIGH gesetzt
```

**Listing A.7:** Analoger PIN wird zum digitalen PIN



## Aufgabe 6

Ändere den Blink Sketch so ab, dass eine LED, die mit dem analogen PIN A0 verbunden ist mit einer Frequenz von einer Sekunde blinkt.

### pullup-Widerstände

Die analogen PINs besitzen ebenfalls Pullup-Widerstände. diese Pullup-Widerstände können mit den folgender Befehl aktiviert werden:

```
1 digital(A0, HIGH); // Set Pullup auf analogen PIN 0
```

Der PIN sollte natürlich dann im INPUT-Mode sein.

Das Einschalten des Pullup-Widerstandes verfälscht natürlich die Werte von analogRead()!

### Für Experten!

Wenn du die analogen PINs verwenden möchtest, kann es vorkommen, dass sie sich manchmal komisch verhalten.

- Wenn ein analoger PIN als Ausgang definiert wurde, wird der Befehl analogRead() nicht korrekt funktionieren. Wenn dies der Fall ist, muss der PIN zuvor als Eingang definiert werden, erst dann kann der Befehl analogRead() korrekt ausgeführt werden.
- Wenn zwischen verschiedenen analogen Eingang zu schnell gewechselt wird, kann es passieren, dass A/D-Messwerte durch elektrische Störsignale (genannt Noise) verfälscht werden. Deshalb ist es sinnvoll beim Wechseln von analogen Eingängen eine kurze Pause einzubauen.

Abbildung A.6.: Analoger PIN A0 als digitaler PIN verwendet

---

## A.4. Interruptsteuerung

Der Mikrocontroller des Arduino ist mit einer Interrupt-Steuerung (IRC) ausgestattet. Die Aufgabe einer IRC ist es auf stattfindende Ereignisse sofort zu reagieren. Dazu wird die aktuelle Aufgabe unterbrochen und sofort auf das Ereignis zu reagieren. Man könnten Sie sich fragen, warum ein Interrupt notwendig ist, um auf externe Ereignisse zu reagieren? Schließlich können man den Zustand externer Pins jederzeit überprüfen.

Das Problem ist, dass wenn die Aufgaben komplexer werden, es sicherlich nicht sinnvoll ist alle Abläufe in der Loop-Routine abzuarbeiten. Das auf bestimmte Signale unterbrechen der Loop-Routine ist ein entscheidenden Vorteil - die Unterbrechungen sind asynchron.

Ein asynchrones Ereignis ist etwas, das außerhalb des regulären Ablauf der Loop-Routine auftritt – es kann jederzeit passieren, egal, was Ihr Sketch im Moment gerade ausführt. Dies bedeutet, dass anstatt manuell prüfen, ob ein bestimmtes Ereignis eintritt, übernimmt diese Aufgabe der Mikrocontroller selbst.

Wenn ein Projekt ein genaues Timing verlangt oder schnell auf Eingaben (z.B. von Sensoren) reagieren soll, dann sind Interrupts das Mittel der Wahl. Im folgenden Abschnitt erkläre ich die Funktion und Verwendung von Interrupts. ?

### A.4.1. Was sind Interrupts?

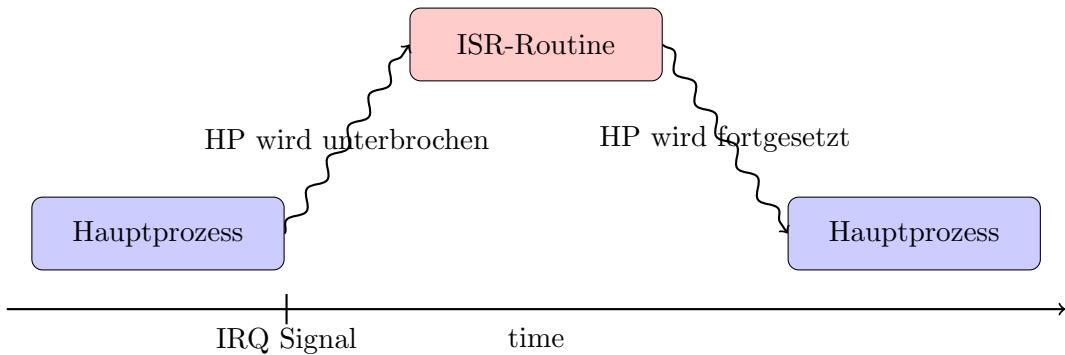
Im Grunde ist ein Interrupt ein Signal, das den ablaufenden Prozess unterbricht. Dieses Signal kann entweder durch ein externes Ereignis (z.B. die Änderung eines digitalen Pin Zustandes) oder durch ein internes Ereignis (z.B. ein Zeitsignal eines Softwaresetimers) ausgelöst werden.

Einmal ausgelöste (triggered), wird ein Interrupt den zur Zeit ablaufenden Prozess anhalten (den Zustand dieses Prozesses speichern) und eine so genannte “interrupt service routine” (ISR) ausführen. Die ISR reagiert auf den Interrupt, wenn die Abgeschlossen ist, wird er zuvor aufgeführte Prozess fortgeführt.

### Die verschiedenen Arten von Interrupts

Es gibt im Wesentlichen zwei Arten von Interrupts:

- Hardware-Interrupts, die auf ein externes Ereignis reagieren, wie z. B. wenn bei einem Eingangspin der logische Zustand von HIGH nach LOW wechselt.
- Software-Interrupts, die auf eine Software Ereignis reagieren.



**Abbildung A.7.:** Funktionsweise eines IRQs, das Hauptprogramm wird unterbrochen um die IRQ-Routine auszuführen.

Die 8-bit AVR-Prozessoren, die meistens in Arduino-Boards verbaut sind, sind nicht in der Lage auf Software-Interrupts zu reagieren.

In der Arduino-Sprache gibt es deshalb nur eine einzige Art von Interrupts, die `attachInterrupt()`-Funktion.

Für einfache Anwendungen ist diese Funktion ausreichend, aber es gibt noch eine ganz andere Art von Hardware-Interrupts.

#### A.4.2. Die Interrupt Service Routines

Jeder Prozessor AVR hat eine Liste von Interrupt-Quellen, oder Vektoren, die die Art der Ereignisse, die einen Alarm auslösen können, umfassen. Die Interrupt-Vektor - wenn Interrupts aktiviert sind und eines dieser Ereignisse eintritt, wird der Code an einer bestimmten Stelle im Programmspeicher springen. Durch das Schreiben eines ISR und dann die Anbringung eines Links die es bei der Interrupt-Vektors Speicherplatz, können wir sagen unseren Code, um etwas Bestimmtes, wenn ein Alarm ausgelöst wird tun.

Implementieren wir dies mit einem einfachen Beispiel: Erkennen, wann eine Taste gedrückt wurde, und eine Aktion auf diesem Pressefreiheit. In diesem Tutorial verwenden wir ein Standard Arduino-Board, um die Dinge konsequent, wenn wir uns auf die Pins und unser Programm Setup beziehen. Sie können diese Techniken mit einem einfachen AVR zu nutzen, brauchen Sie nur auf das Datenblatt überprüfen, um sicherzustellen, dass Sie wissen, welche Pins die Sie benötigen. Hier ist unser Beispiel Schaltung:

---

## **Die Arduino Interrupt-Funktion**

### **Implementierung eines Interrupt in einem Sketch**

## B — Data Kommunikations Systeme

## B.1. Serielle Schnittstelle - die Verbindung mit dem PC

Die verschiedenen Komponenten eines Computersystems müssen Informationen austauschen, dieser Austausch vollzieht sich mit Hilfe von Schnittstellen. Es macht Sinn, für diese Schnittstellen bestimmte Konventionen festzulegen und einzuhalten, damit die Datenkommunikation reibungslos ablaufen kann. Eine mögliche Schnittstelle ist die sog. serielle Schnittstelle.

### B.1.1. Die TTY-Schnittstelle

Die TTY-Schnittstelle ist die älteste serielle Schnittstelle und wurde ursprünglich zur Verbindung von Fernschreibern entwickelt. Da im Ruhezustand ein konstanter Gleichstrom von 20 mA fließt, wird die Schnittstelle oft auch als 20-mA-Stromschnittstelle bezeichnet. Die Übertragung von Daten geschieht nun einfach so, dass der Sender den Schleifenstrom

#### Info:

Der Optokoppler dient zur galvanischen Trennung von Sender- und Empfänger-Stromkreis.

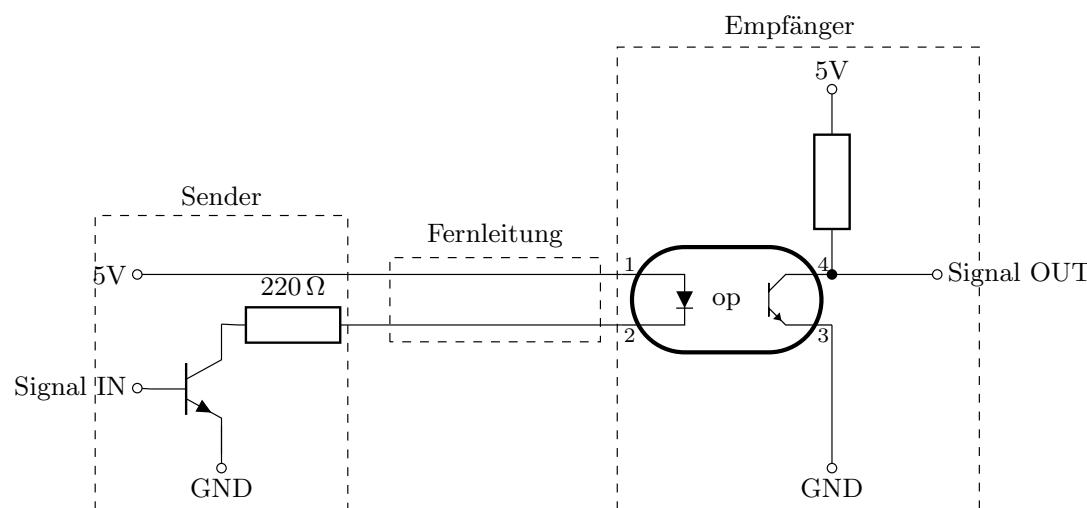


Abbildung B.1.: TTY Schaltung (nur eine Richtung)

in einem bestimmten Rhythmus unterbricht. Der Empfänger erkennt die rhythmischen Unterbrechungen. Für diese Art

---

der Datenübertragung genügt eine 2-adrigen Leitung ohne. Die Datenübertragung funktioniert aber immer nur eine Richtung. Für den gleichzeitigen Datenversand benötigt man getrennte Stromschleifen für Senden und Empfang. Heute wird die TTY-Schnittstelle mit moderne Halbleiterbauelemente realisiert. Der Strom wird mit Hilfe eines Schalttransistor modelliert, der Empfang der Daten durch einen Optokoppler (siehe Abb. B.1).

### B.1.2. Aufbau eines seriellen Datenpakets

Auf der Leitung liegt zu jedem Zeitpunkt nur ein BIT (siehe Abb. ??). Die zu übertragende Daten werden in einzelnen Datenpakete aufgeteilt. Ein Datenpaket besteht aus 7-Daten-BITs und 3 Steuerung-BITs.

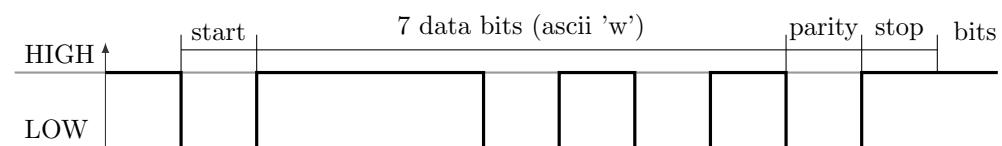


Abbildung B.2.: Serielles Datenpaket (das 7-BIT ASCII-Zeichen 'w' wird übertragen)

Jedes Datenpaket beginnt mit dem start-BIT (log. LOW). Danach folgen die chrakter-BITs die zu sendenden Daten (z.B. sieben Bit des ASCII-Codes). Als Abschluss folgt das paritäts-BIT und ein oder zwei stop-BITS (log. HIGH), worauf die Leitung wieder im Ruhezustand (log. HIGH) ist.

Jedes BIT liegt für die gleiche, feste Zeitspanne auf der Leitung. Die Übertragungsgeschwindigkeit wird in Boud angegeben. Denn zwischen einzelnen Datenpaketen die Leitung beliebig lange im Ruhezustand verharren kann, ist die maximale Übertragungsgeschwindigkeit erreicht, wenn die Datenpakete, direkt aufeinanderfolgen. Wenn zum Beispiel einzelne 7-Bit-ASCII-Zeichen übertragen werden, so ergibt sich bei 9600 Baud eine maximale Datenrate von  $9600/(7 + 3) = 960$  Zeichen pro Sekunde.

Damit der Empfänger ein Datenpaket empfangen kann muss er zum einen den Anfang und das Ende des Datenpakets erkennen. Zusätzlich zu den Datenbits kann ein sog. Paritäts-BIT übertragen werden mit dessen Hilfe der Empfänger Übertragungsfehler erkennen kann.

#### Info:

Die Firma Baudot, eine Pionier der Fernschreibertechnik war Namensgeber für die Einheit Boud 1 Baud = 1 Bit pro Sekunde.

## start-BIT

Jeden Datenpaket wird durch das Start BIT eingeleitet. Werden keine Daten übertragen, befindet sich die Leitung im Ruhezustand (log. HIGH). Das Start BIT setzt die Leitung für die Dauer eines BITS auf log. LOW. Der Empfänger weiss nun, dass die folgenden 7 BITS die Daten enthalten.

## parity-BIT

Das Parität BIT ermöglicht eine primitive Fehlerkontrolle. Wird mit gerader Parität gearbeitet, so setzt der Sender das Paritätsbit auf log. HIGH, falls die Daten-BITS eine ungerade Anzahl von gesetzten (log. HIGH) BITS enthält. Bei einer geraden Anzahl wird das Paritätsbit auf log. LOW gesetzt. Der Empfänger prüft nun nach der gleichen Vorschrift, ob das Paritätsbit zu den Datenbits 'passt'. Falls bei der Übertragung eines der Datenbits verfälscht worden ist, so ist dies also vom Empfänger erkennbar. Wenn zwei Übertragungsfehler vorliegen, dann kann der Empfänger das nicht mehr erkennen.

## stop-BIT

Das stop-BIT (log. LOW) beendet das Datenpaket. Danach geht die Datenleitung in den Ruhezustand (log. HIGH) über. Dieser Ruhezustand besteht bis zum nächsten Start BIT

### B.1.3. RS-232: eine moderne serielle Schnittstelle

Die RS-232-Schnittstelle wurde ursprünglich bei Großcomputern verwendet, um Terminals an einen Zentralrechner anzuschließen. Sie wurde bis zur Einführung der USB-Schnittstelle auch zum Anschluß von Peripheriegeräten (zum Drucker, Modem etc) an einen Rechner benutzt. Bei technischen Anwendungen, wie zum Beispiel CNC-Maschinen wird die serielle Schnittstelle heute immer noch verwendet. Physikalisch ist die RS-232-Schnittstelle eine Spannungsschnittstelle für jede Richtung ist eine Signalader erforderlich, dazu kommt eine gemeinsame GND-Leitung.

### B.1.4. UART serieller Interface-IC

Prinzipiell kann man das serielle Datenpaket softwaretechnisch erzeugen, indem man eine digitalen PIN zu den richtigen Zeitpunkten auf LOW oder HIGH setzt. Der Datenempfang per Software ist schon etwas schwieriger zu bewältigen. In der Praxis verwendet man meist einen seriellen Interface-IC, UART genannt. Das Arduino Board besitzt einen UART-Port, dieser kann mit Hilfe der System-Bibliothek Serial gesteuert werden.

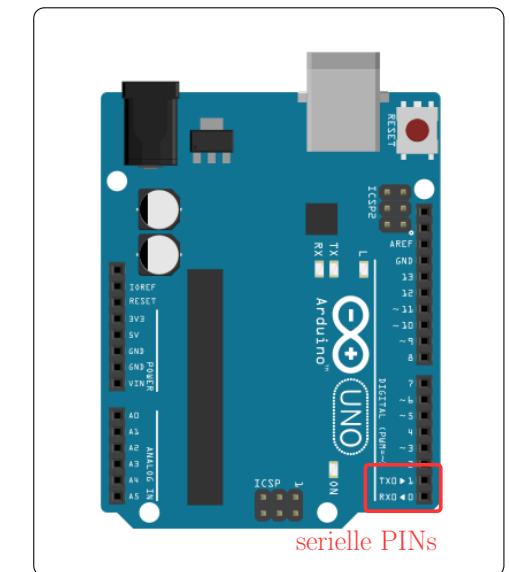


Abbildung B.3.: tx und rx PINs auf dem Arduino Board

#### Info:

Der PIN zum Senden von Daten wird tx (transmit) und zum empfangen von Daten rx (receive) genannt.

#### Info:

UART steht für Universal Asynchronous Receiver and Transmitter

---

### B.1.5. Softwaretechnisch Daten von Arduino zu Arduino senden

Obwohl es für das senden von Daten über eine serielle Schnittstelle die Systembibliothek Serial gibt, macht es Sinn sich die ganze Sache genauer anzuschauen und selber eine Softwarelösung zu programmieren.

#### Aufbau der Schaltung

Für den Aufbau einer seriellen Verbindung unter zwei Arduinos müssen nur zwei digitale PINs und zwei GND PINs miteinander verbunden werden.

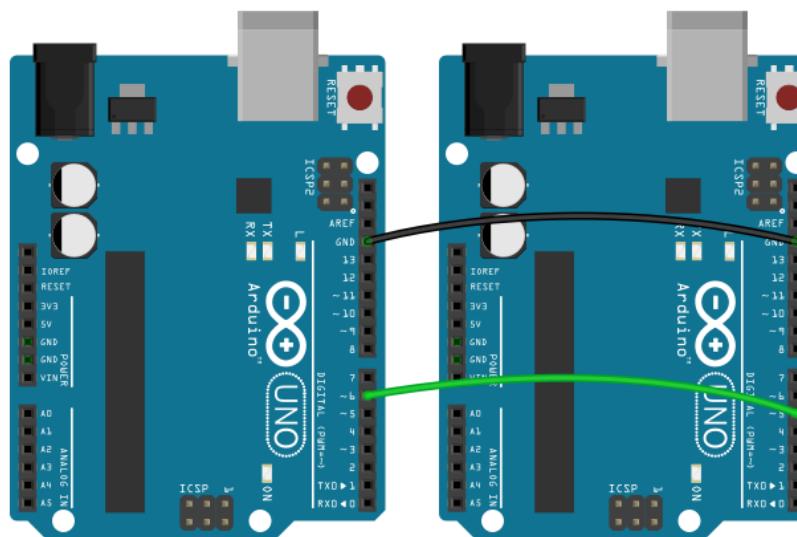


Abbildung B.4.: Arduino zu Arduino Verbindung mit einer seriellen Schnittstelle

#### Das Senden von Daten

```
1 #include <ctype.h>
2 #define bit9600Delay 84
3
4 byte tx = 6;
5
6 void setup() {
7     pinMode(tx,OUTPUT);
8     digitalWrite(tx,HIGH);
9 }
10
11 void SWprint(int data)
12 {
13     byte mask;
14     digitalWrite(tx,LOW);
15     delayMicroseconds(bit9600Delay);
16     for (mask = 0x01; mask>0; mask <= 1) {
17         if (data & mask){ // choose bit
18             digitalWrite(tx,HIGH);
19         }
20         else{
21             digitalWrite(tx,LOW);
22         }
23         delayMicroseconds(bit9600Delay);
24     }
25     digitalWrite(tx, HIGH);
26     delayMicroseconds(bit9600Delay);
27 }
28
29
30 void loop()
31 {
32     SWprint(toupper('w'));
33 }
```

Initialisieren der Schleifenvariablen

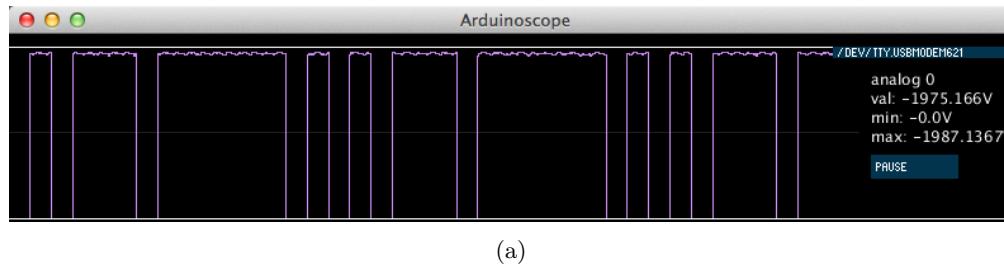
## Das Empfangen von Daten

---

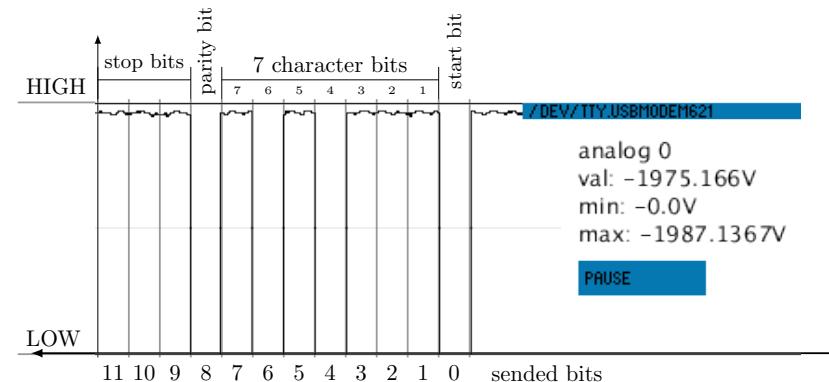
```
1 #include <ctype.h>
2 #define bit9600Delay 84
3
4 byte rx = 5;
5 byte SWval;
6
7 void setup() {
8     pinMode(rx, INPUT);
9 }
10
11 int SWread() {
12     byte val = 0;
13     while (digitalRead(rx));
14     if (digitalRead(rx) == LOW) {
15         delayMicroseconds(halfBit9600Delay);
16         for (int offset = 0; offset < 8; offset
17            ++) {
18             delayMicroseconds(bit9600Delay);
19             val |= digitalRead(rx) << offset;
20         }
21         delayMicroseconds(bit9600Delay);
22         delayMicroseconds(bit9600Delay);
23         return val;
24     }
25
26 void loop()
27 {
28     SWval = SWread();
29 }
```

Initialisieren der Schleifenvariablen

## Ergebnis



(a)



(b)

**Abbildung B.5.:** Serielles Signal

---

## B.2. I2C Schnittstelle

Der I2C Bus ist ein synchroner serieller 2-Draht Bus, der in den 80er Jahren von Philips entwickelt wurde. I2C gesprochen 'I Quadrat C' kommt von der Abkürzung IIC und bedeutet Inter-Integrated Circuit. Er wird hauptsächlich dazu benutzt, zwischen Schaltkreisen, die sich auf einer Platine verbinden, Daten auszutauschen. Die beiden Leitungen, die den I2C Bus bilden heißen SCL und SDA. SCL steht für Signal Clock und ist die Takteleitung für den Bus. Deshalb spricht man auch von einem synchronen Bus. SDA steht für Signal Data und ist die Datenleitung. Die Datenübertragungsrate des I2C Busses beträgt 100kHz im Standard Mode, bzw. 400kHz im Fast Mode. Aus Lizenzgründen nennt man das I2C Interface bei Atmel TWI (Two Wire Interface).

Der I2C Bus ist ein Multi Master/Slave Bus. Das bedeutet, es gibt mindestens einen I2C Master und ebenso mindestens einen I2C Slave. Der Master selektiert einen Slave durch seine Slave Adresse, die innerhalb eines Busses eindeutig sein muss. Eine Datenübertragung kann nur durch einen I2C Master initiiert werden. Der Slave bleibt immer passiv und lauscht nur auf die Slave Adresse und vergleicht diese mit seiner eigenen Slave Adresse. Erst wenn er seine Slave Adresse erkennt, greift der Slave auch aktiv in das Busgeschehen ein.

Aus Sicht des I2C Masters unterscheidet man zwischen Read und Write Sequenzen. Bei einer Read Sequenz liest der I2C Master Daten vom I2C Slave. Bei einer Write Sequenz sendet der I2C Master Daten zum Slave.

## C — Hintergrundwissen: Programmieren

---

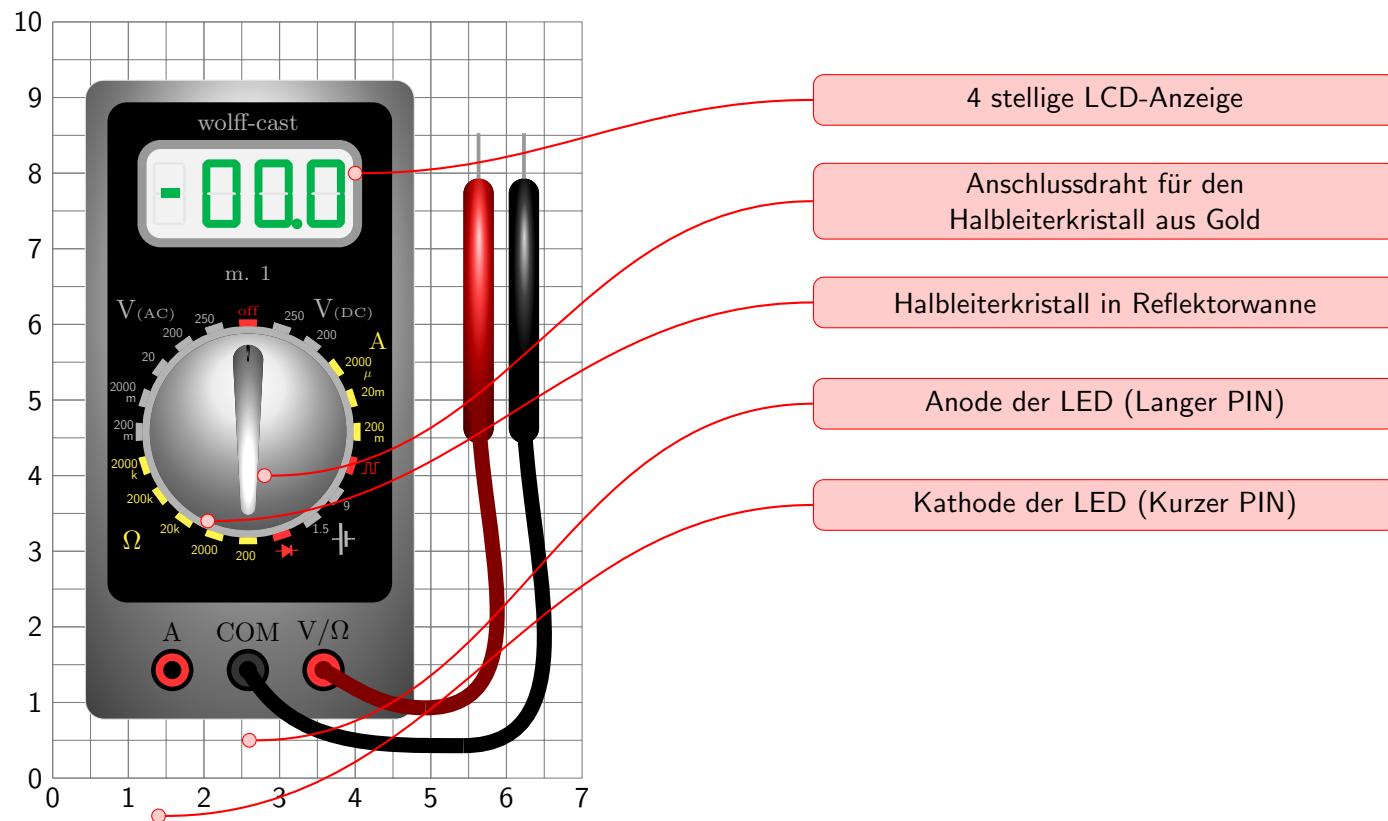
## **C.1. Rechnen im dem Mikrocontroller**

---

## **C.2. Programmablaufplan**

## D — Hintergrundwissen: Elektronik

## D.1. Multimeter



---

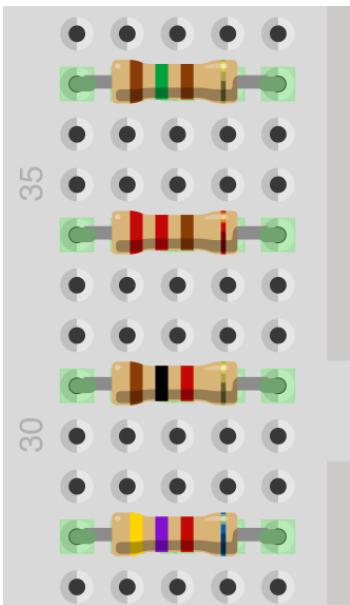
## D.2. Farbcodes von Widerständen

Auf Schichtwiderstände werden die Widerstandswerte mit Hilfe von farbigen Ringen aufgemalt. Dies hat gegenüber den Vorteil, dass die Kennzeichnung eines in eine Schaltung eingetragenen Widerstandes auf jeden Fall zu erkennen ist. Um den Widerstandswert bestimmen zu können, braucht man den Farbcode. Bei einem Widerstand mit vier Ringen geben die ersten beiden Ringe die Zahlen vor der Zehnerpotenz an. Die Farbe des dritten Rings gibt den Exponenten der Zehnerpotenz an. Der vierte, etwas abgesetzte Ring macht eine Angabe über die Toleranz des Widerstandswertes. Fehlt der Toleranzring, so kann man davon ausgehen, dass der Widerstandswert nur auf  $\pm 20\%$  genau ist.

Ringfarbe	Ring Nr. 1 – 4			
Silber	–	–	$10^{-2}$	$\pm 10\%$
Gold	–	–	$10^{-1}$	$\pm 5\%$
Schwarz	–	0	$10^0$	–
Braun	1	1	10	$\pm 1\%$
Rot	2	2	$10^2$	$\pm 2\%$
Orange	3	3	$10^3$	–
Gelb	4	4	$10^4$	–
Grün	5	5	$10^5$	$\pm 0.5\%$
Blau	6	6	$10^6$	$\pm 0.25\%$
Lila	7	7	$10^7$	$\pm 0.1\%$
Grau	8	8	–	$\pm 0.05\%$
Weiß	9	9	–	–

**Tabelle D.1.:** Farbkodes von Kohleschichtwiderständen

**Aufgabe:** Bestimme die Widerstandswerte folgender Widerstände.



a)

	Ring Nr.	1	2	3	4	Ergebnis
Farbe	braun	rot	braun	gold	-	
Wert	1	5	·10	$\pm 20\%$	$150 \pm 20\% \Omega$	

b)

	Ring Nr.	1	2	3	4	Ergebnis
Farbe						-
Wert						

c)

	Ring Nr.	1	2	3	4	Ergebnis
Farbe						-
Wert						

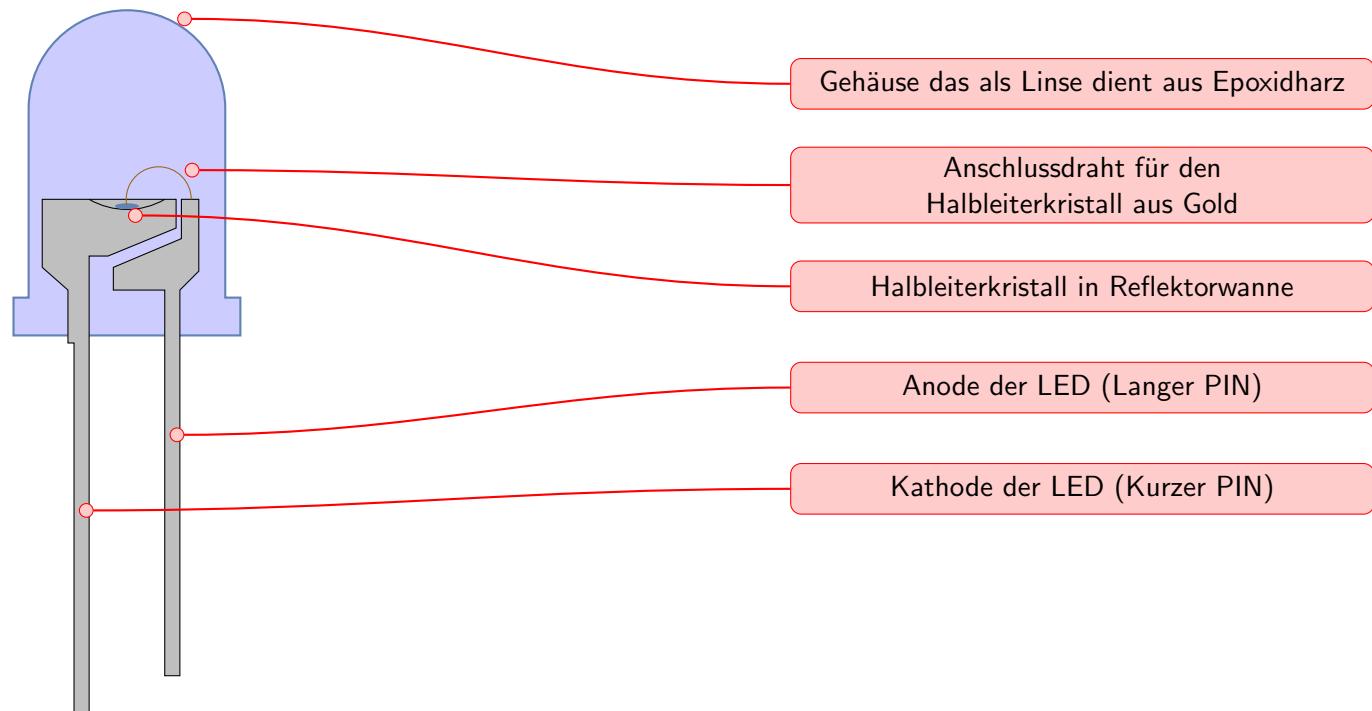
d)

	Ring Nr.	1	2	3	4	Ergebnis
Farbe						-
Wert						

---

## D.3. Hintergrundwissen LED

### D.3.1. Funktionsweise



### D.3.2. Vorwiderstand einer LED berechnen

---

#### **D.4. Schaltungsentwicklung mit Fritzing**

## **E — Hintergrundwissen: Free Software Foundation and Creative Commons**

---

## **E.1. Open Source und Creative Commons**

# **Literaturverzeichnis**

# **Index**

Arduino Uno, 16

IDE, 17

