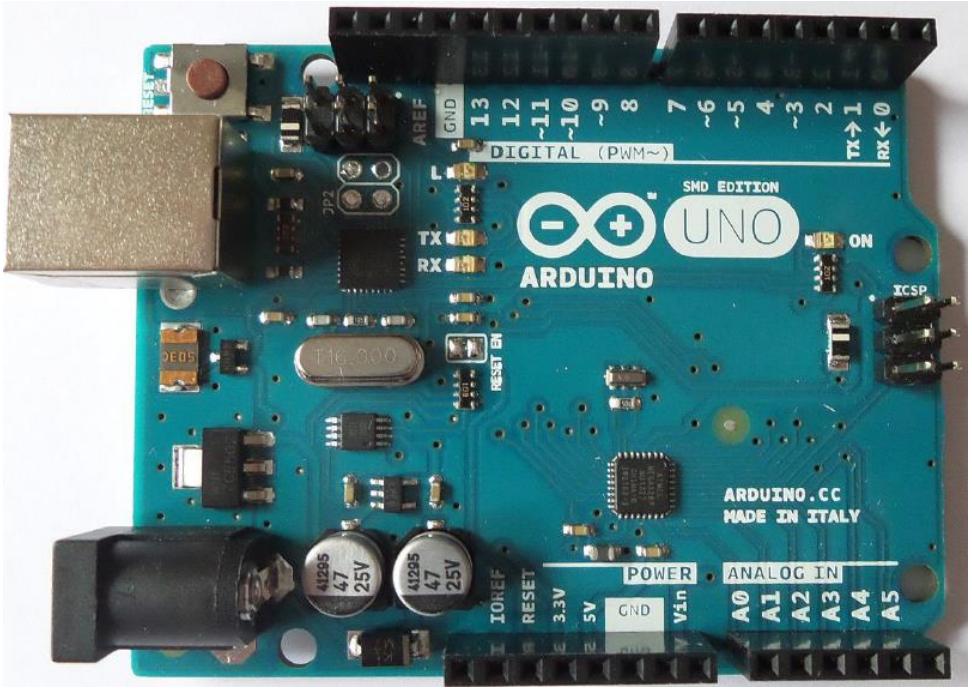
Einführung in Mikrocontroller
–
Der Arduino als Steuerzentrale



Arbeitsheft für Schülerinnen und Schüler

Tobias Frick Reinhard Fritsch Martin Trick

Stand: 11.09.2017

Schülerforschungszentrum Bad Saulgau

Inhalt

1. Was ist ein Mikrocontroller? 1
2. Hardware: Der Arduino 2
3. Software: Die Arduino-Entwicklungsumgebung 3
4. Jetzt geht's los: Dein erstes Programm 5
5. Verwendung einer Steckplatine: RGB-LED und 7-Segment-Anzeige 8
6. Vorwiderstände von LEDs 10
7. Farbcodes von Widerständen 11
8. Lautsprecher am Arduino betreiben 12
9. Rückmeldungen an den Nutzer – der serielle Monitor 14
10. Variablen 15
11. LCD 18
12. for-Schleife 19
13. Programmablaufpläne (PAP) 21
14. while-Schleife und random-Befehl 22
15. Dimmen einer LED - Pulsweitenmodulation 24
16. Der Arduino bekommt Sinne 26
17. Hell oder dunkel? Der Arduino wird zum Messgerät 27
18. Potentiometer 29
19. Spannungsteiler 31
20. Der Arduino als Thermometer 32
21. Taster – Start per Knopfdruck 33
22. if-Bedingung Der Arduino reagiert auf seine Umwelt 35
23. Fernbedienung 39
24. Ultraschall-Entfernungsmesser 40
25. Motoren mit dem Arduino steuern 42
26. Servo-Motor für genaue Drehwinkel 43
27. Motortreiber 44
28. Bewegungsmelder Der Arduino als „Wachhund" 46
29. Projekt – Fußgängerampel 47
30. Anhang A Grundlagen zum elektrischen Stromkreis 48
31. Anhang B Aufbau-Hilfen ..... **Fehler! Textmarke nicht definiert.**
32. Anhang C Kleiner Lötkurs 49
33. Anhang D Einbinden einer Bibliothek 51
34. Anhang E Trouble-Shooting 52
35. Anhang F Verwendete Bauteile 54

1. Was ist ein Mikrocontroller?

Ein Airbag wird ausgelöst, wenn ein Auto in einen Unfall verwickelt ist. Rasenmäherroboter mähen in immer mehr Gärten den Rasen und zwar exakt bis zur Grundstücksgrenze. In vielen Geschäften öffnet sich am Eingang eine elektrische Schiebetür, sobald sich ein Kunde nähert. Neue Spülmaschinen erkennen, wie schmutzig das Geschirr darin ist, und passen ihre die Spülzeit daran an. Viele Herzpatienten bekommen nach einem Herzinfarkt einen kleinen Defibrillator eingepflanzt, der einen zukünftigen drohenden Herzinfarkt erkennt und den Patienten mit einem Stromstoß zurück ins Leben holt.

Unser Alltag ist voll von solchen elektrischen Geräten, die uns das Leben erleichtern. Manchmal sind sie sogar ware Lebensretter.

Vielleicht hast du dich bei dem ein oder anderen Beispiel von oben schon einmal gefragt, wie diese Dinge eigentlich genau funktionieren.

Sicherlich steckt in ihnen eine ganze Menge Technik bzw. Elektrontik. Doch ein normaler Computer hätte in den meisten Geräten keinen Platz und wäre auch viel zu teuer und zu „gut“ für die Anwendungsbeispiele von oben.

Wenn man einen Blick in die Technik hinter Airbags, Rasenmäherrobotern, Schiebetüren und Spülmaschinen wirft, stößt man immer auf ein gemeinsames Bauteil: **Mikrocontroller**.

Mikrocontroller funktionieren im Prinzip wie kompakte Computer, jedoch mit viel geringerer Leistungsfähigkeit. Sie vereinen verschiedene Bauteile wie Recheneinheit (CPU), Speicher, Schnittstellen wie USB, Display-Controller und Analog-Digital-Wandler auf einem einzigen Chip. Der hier verwendete Mikrocontroller hat den Name „ATmega328“. Der Vorteil von Mikrocontrollern ist, dass sie speziell für ihre Aufgabe konzipiert werden können und relativ günstig sind.

Dieses Arbeitsheft ermöglicht es dir weitestgehend eigenständig das Programmieren eines Mikrocontrollers zu erlernen. Der Mikrocontroller, den wir verwenden, heißt Arduino Uno bzw. nur **Arduino**.

Für das Wort „Mikrocontroller“ verwendet man oft die Abkürzung **µC**.
„µ“ (sprich: mü) ist ein griechischer Buchstabe und wird als Abkürzung für „Mikro“ verwendet.
„Arduino“ ist der Name einer Kneipe, in der sich die Erfinder des Arduino 2005 oft trafen.

2. Hardware: Der Arduino

Der Arduino stellt sich vor:

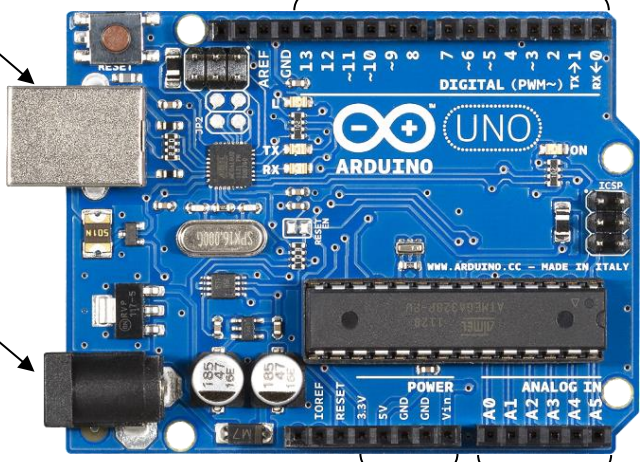
Das Herz unseres Arduino: Der eigentliche Mikrocontroller heißt „Atmega328“. Seine Anschlüsse werden **Ports** genannt. ATmega328 hat 28 Ports. Manche dieser Ports dienen als Ein- und Ausgänge für elektrische Signale.

Die Ports 0 bis 13 sind sogenannte **digitale Ports**. Das heißt, sie können nur zwei Zustände unterscheiden: „ein“ oder „aus“ bzw. „high“ oder „low“. In der Computersprache entspricht das den Zahlen 0 oder 1.

Reset-Knopf: Startet ein Programm von vorne.

USB-Schnittstelle für die Verbindung zum PC.
Er dient als Datenleitung und Stromversorgung für den Arduino.

Anschluss für eine externe Stromversorgung (bis 12V), falls der Arduino ohne PC betrieben wird.



Die Anschlüsse dienen zur Stromversorgung der **Aktoren** (z.B. Motoren, Lautsprecher, …) und liefern unterschiedliche Spannungswerte.

Die Ports A0 bis A5 sind **analoge Eingänge** für **Sensoren**. An ihnen können Spannungen zwischen 0 V und 5 V gemessen werden.

Als **Hardware** bezeichnet man allgemein die mechanischen und elektronischen Bauteile eines Mikrocontrollers bzw. PCs (im Prinzip alles, was man „in die Hand nehmen“ kann). Der Begriff „hardware“ kommt aus dem Englischen und bedeutet übersetzt ursprünglich „Eisenwaren“. Diese Bedeutung hat der Begriff im englischsprachigen Raum auch heute noch. Zusätzlich hat er sich aber weltweit auch als allgemeiner Überbegriff für die Bauteile eines Computersystems verbreitet.

Auf der Platine unseres **Arduino** sind neben dem Mikrocontroller noch viele kleine Bauteile (Widerstände, Spannungswandler, Strombegrenzer, …) und viele verschiedene Anschlüsse angebracht.

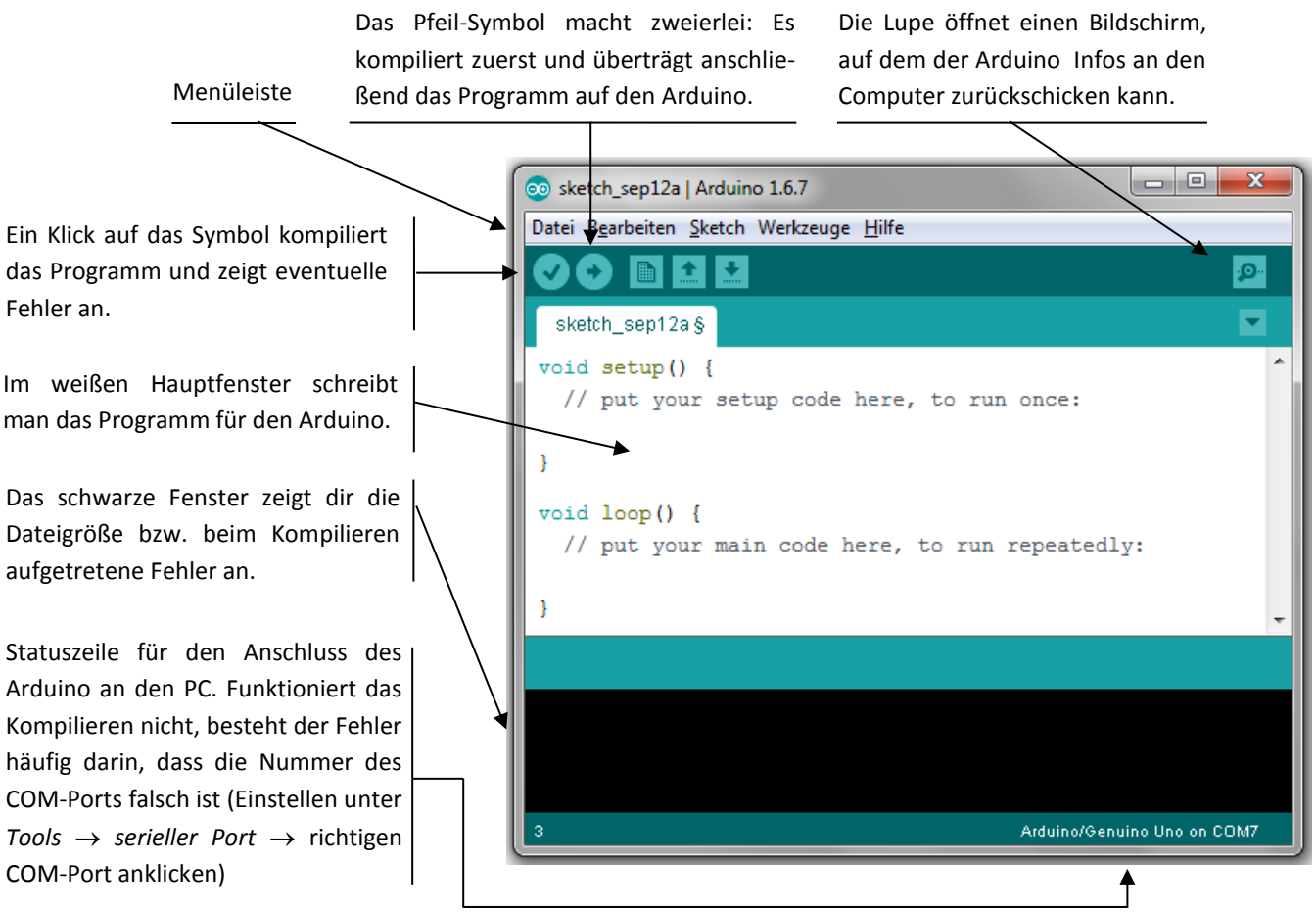
Arduino vs. Smartphone

Bei Computern oder Smartphones wird oft die Taktfrequenz des Prozessors angegeben. Damit ist die Geschwindigkeit gemeint, mit der Daten verarbeitet werden können. Der Atmega328 unseres Arduino besitzt eine Taktfrequenz von 16 MHz und 32 Kilobyte Speicherplatz. Ein Vergleich mit dem iPhone 8 lässt den Arduino dagegen alt aussehen: Es hat einen Prozessor mit 2,4 GHz (das sind 2400 MHz) Taktfrequenz und 3 GB (das sind 3.000.000 Kilobyte) Arbeitsspeicher. Umso erstaunlicher: Beim Flug zum Mond half Neil Armstrong ein Computer mit 4 Kilobyte Arbeitsspeicher und einem 1 MHz-Prozessor.

3. Software: Die Arduino-Entwicklungsumgebung

Vor der Arbeit mit dem Arduino muss dieser mit dem PC verbunden und programmiert werden. Der Arduino ist dabei immer nur so „klug“ wie der, der ihn bedient. Er kann nicht eigenständig denken, stattdessen arbeitet er Befehle der Reihe nach ab. Eine Abfolge von solchen Befehlen heißt **Programm** oder **Sketch**.

Um den Arduino zum Leben zu erwecken, schreibt man auf dem PC in der sogenannten Entwicklungsumgebung ein Programm und überträgt es dann über die USB-Schnittstelle auf den Arduino. Wie das funktioniert, zeigt die folgende Abbildung.



<u>Aufgabe 3.1:</u> Starte die Entwicklungsumgebung, indem du das Programm *arduino.exe* öffnest.

Die Programme werden in einer Programmiersprache geschrieben, die der berühmten Sprache „C“ ähnelt, programmiert. Bevor das geschriebene **Programm** (dieses wird oft auch „**Sketch**“ genannt) über das Datenkabel auf den Arduino übertragen wird, übersetzt es ein elektronischer Dolmetscher in Maschinensprache. Dieser Vorgang heißt **Kompilieren**. Es ist übrigens ganz hilfreich für uns, dass wir nicht die eigentliche Sprache des Arduino lernen müssen, um mit ihm zu kommunizieren. Diese ist nämlich sehr kompliziert.
Zum Beispiel würde die Rechnung c = a + b in Maschinensprache so aussehen: *8B 45 F8 8B 55 FC 01 D0*

Bevor du eine Schaltung nachbaust und vom Arduino steuern lässt, musst du sicherstellen, dass dein Mikrocontroller richtig mit dem PC kommuniziert. Dazu übertragen wir zunächst ein „leeres“ Programm auf den PC. Alle Programme haben dabei den gleichen Aufbau.

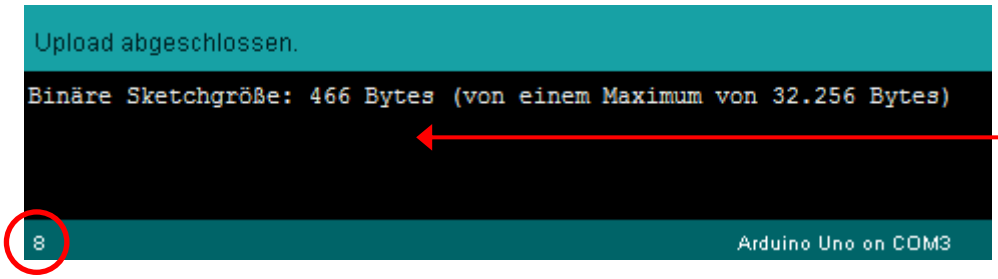
```
void setup() {
  // Grundeinstellungen;
}

void loop() {
  // Anweisungen;
}
```

Dieser Programmteil wird **einmal** durchlaufen. Im setup()-Teil werden die Ein- und Ausgänge (und andere Grundeinstellungen) festgelegt.

Dieser Programmteil wird **unendlich oft** wiederholt (engl.: loop = Schleife). Hier befindet sich das eigentliche Programm.

An den Stellen `// Grundeinstellungen;` und `// Anweisungen;` steht im echten (d.h. nicht leeren) Programm später dein Programmcode.
Es kommt übrigens später oft vor, dass man in sein Programm etwas hineinschreibt, häufig als Erklärung für jemanden, der bei der Entstehung des Programms nicht dabei war. Dieser „eigene“ Text soll und kann später beim Kompilieren des Programms nicht übersetzt werden. Dafür gibt es sogenannte Kommentarstellen: Alles, was in einer Zeile nach zwei Schrägstrichen `//` geschrieben wird, wird beim Kompilieren nicht beachtet. Anschaulich erkennt man Kommentare daran, dass sie vom Programm grau geschrieben werden.

Aufgabe 3.2
Schreibe und kompiliere zunächst das leere Programm von oben und übertrage es anschließend auf den Arduino. Nach erfolgreichem Übertragen steht in der Statuszeile „Upload abgeschlossen“:



Achtung: An dieser Stelle werden später (Schreib-)Fehler im Programm in orange angezeigt. Meist steht dort auch, was falsch gemacht wurde.

Findest du heraus, was die Zahl „8“ (bei dir kann auch eine andere Zahl stehen) unten links bedeutet?

Fehlermeldungen
Es kommt später öfters vor, dass ein von dir erstelltes Programm Fehler (Schreibfehler, logische Fehler, …) aufweist. Manchmal tritt auch ein Fehler bei der Übertragung des Programms an den Arduino auf. Der Arduino erkennt viele dieser Fehler und zeigt sie – in oranger Farbe geschrieben – im Ereignis-Fenster an. Das heißt also: Immer wenn etwas orange Geschriebenes auftaucht, ist irgendwo ein Fehler aufgetreten.

Was bedeutet eigentlich das Wort „void“?
void kann mit *Leerstelle* oder *Loch* übersetzt werden. Der Begriff *void* wird immer den Programmteilen „setup“ und „loop“ vorangestellt. Später kannst du selbst noch weitere Programmteile (sog. Unterprogramme) erstellen, denen du einen eigenen Namen geben kannst und die auch jeweils mit *void* beginnen.

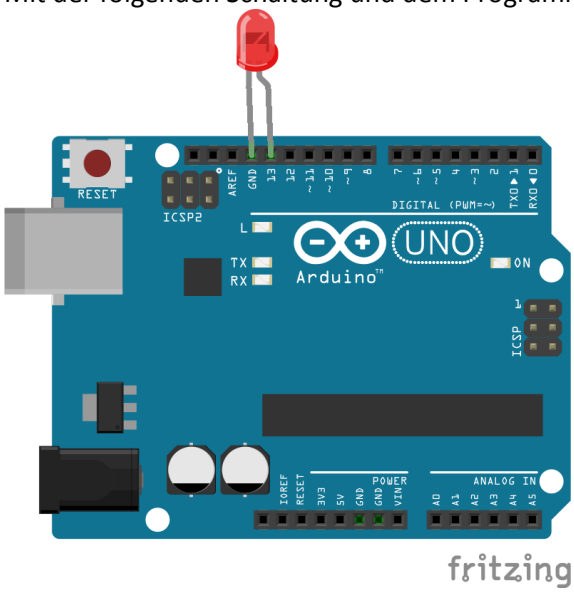
4. Jetzt geht's los: Dein erstes Programm

In diesem Kapitel lernst du, wie man eine mit Hilfe des Arduino eine LED zu Blinken bringst. Am Ende des Kapitels baust du ein Testgerät, mit dem du die Reaktionszeit deines Auges bestimmen kannst.

LEDs lassen sich mit dem Arduino gezielt steuern.
Dazu kann man die 20 Anschlussports des Arduino nutzen (0 – 13 und A0 bis A5). Sie sind wie kleine anschaltbare Steckdosen. Bei jeder Steckdose gibt es immer einen Anschluss mit dem Potenzial 0 V. Das ist beim Arduino der Anschluss GND (Ground). Für den zweiten Anschluss mit dem hohen Potenzial lassen sich beim Arduino die Ports zwischen 5 V und 0 V an- und ausschalten.

Erinnerung an die Physik: Das **elektrische Potenzial** ist eine Art „elektrischer Druck" auf einem Kabel. Ganz ähnlich wie auch in einer Wasserleitung ein hoher oder tiefer Wasserdruck herrschen kann. Verbindet man eine Leitung hohen Drucks mit einem Bereich, in dem ein tiefer Druck herrscht, dann fließt Wasser. Genauso beim Strom: Nur wenn es einen Potenzial**unterschied** gibt, werden Elektronen fließen. Der Potenzialunterschied ist die bekannte Spannung. Achtung: Weil sowohl das Potenzial als auch der Potenzialunterschied die Einheit Volt haben, werden die Begriffe Spannung und Potenzial oft verwechselt!!!

Mit der folgenden Schaltung und dem Programmtext wollen wir eine LED zum Leuchten bringen:



Merke: GND = Ground = 0 V = LOW

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13,HIGH);
  delay(5000);
  digitalWrite(13,LOW);
  delay(5000);
}
```

Aufgabe 4.1
Baue die Schaltung auf und schreibe den Programmtext ab. Beim Anschluss der LED ist es wichtig, diese richtig herum einzustecken (das kurze Bein der LED kommt ans kleinere Potenzial, also an GND)

Was tun, wenn's nicht klappt?

- Sind die einzelnen Befehle richtig geschrieben? (Groß- und Kleinschreibung beachten!)
- Endet jede Befehlszeile mit einem Semikolon (Strichpunkt)?
- Meist hilft dir bei einem Fehler im Programmcode das schwarze Fenster. Dort steht (leider nur auf Englisch), welches Problem beim Kompilieren aufgetreten ist.

Aufgabe 4.2

Erkläre deinem Nachbarn welche Funktionen die einzelnen Programmzeilen jeweils erfüllen.

```
1  void setup() {
2   pinMode(13, OUTPUT);
3  }
4
5  void loop() {
6   digitalWrite(13,HIGH);
7   delay(5000);
8   digitalWrite(13,LOW);
9   delay(5000);
10 }
```

Aufgabe 4.3

Verändere deinen Programmtext so, dass deine LED schneller bzw. langsamer blinkt.

Aufgabe 4.4

Mit diesem Programm blinkt die angeschlossene LED nicht. Erkläre warum!

(Geh davon aus, dass der `setup`-Teil korrekt ist)

```
void loop() {
 digitalWrite(13,HIGH);
 delay(500);
 digitalWrite(13,LOW);
}
```

Aufgabe 4.5

Wie andere hohe Masten müssen auch Windkraftanlagen ein rotes Blinklicht an ihrer Spitze haben, um Flugzeuge zu warnen. Da die Spitze des Mastes bei einem Windrad aber nicht wirklich die Spitze der Gefahrenzone ist, blinkt ihr rotes Licht in einem besonderen Rhythmus: 1 Sekunde an, dann 0,5 Sekunden aus, dann wieder 1 Sekunde an, dann aber 1,5 Sekunden aus.

Programmiere so ein Dauerblinklicht und bertrage das Programm. Läuft es?

Aufgabe 4.6

Die Anweisungen „anweisung1“ bis „anweisung5“ im folgenden Beispiel sind ja keine echten Anweisungen.

Dennoch die Frage: In welcher Reihenfolge werden sie ausgeführt?

```
void setup() {
 anweisung 1;
 anweisung 2;
}

void loop() {
 anweisung 3;
 anweisung 4;
 anweisung 5;
}
```

<u>Aufgabe 4.7</u> „Augen-Testgerät“

Verringere die delay-Zeit einer blinkenden LED so lange, bis du das Blinken nicht mehr mit deinem Auge erkennen kannst.

Bestimme daraus die Blinkfrequenz (wie oft pro Sekunde deine LED angeht) deines Auges. Bei aktuellen Kinofilmen wird das Bild in einer Sekunde ca. 30 mal geändert. Ab dieser Bildfrequenz nimmt unser Auge die Einzelbilder als flüssige Bewegung wahr

Vergleiche deine eigene Bildfrequenz mit deinem Nachbarn und mit der Bildfrequenz eines Kinofilms.

<u>Hausaufgabe 4.8</u> „Spickzettel“

Lege dir auf einem karierten DIN A4-Blatt eine Liste an, in die du alle etwa 40 einzelnen Befehle und Funktionen des Arduino mit einem Beispiel einträgst (3 Spalten: linke Spalte → Befehl; mittlere Spalte → Beispielcode, in dem der Befehl vorkommt; rechte Spalte → Kommentare und Erklärungen).

Bislang sind es genau 5 Befehle. Kannst du sie und ihre exakte Schreibweise schon auswendig?

Wie findet man Fehler?

Der Arduino ist dir gegenüber ziemlich streng seinen Umgang mit Fehlern im Programmcode angeht. Begriffe, die er nicht kennt, oder die nicht seiner „Grammatik“ entsprechen, akzeptiert er nicht und meldet sie als Fehler.

Die Entwicklungsumgebung zeigt dir teilweise auf deutsch und teilweise auf englisch in einem orangen Feld unter dem eigentlichen Programmcode an, wo sie einen Fehler vermutet. Zusätzlich springt der Arduino in die Zeile, die nach der fehlerhaften Zeile kommt und markiert diese rosa.

```
  digitalWrite(13, LOW);
  delay(1000)
}
```

```
expected ';' before '}' token
```

← **fehlerhafte Zeile: Strichpunkt fehlt**

← **Nachfolgende Zeile wird rosa markiert**

← **Fehlermeldung mi t Angabe des Fehlers**

Die häufigsten Fehler sind hier aufgeführt:

`expected ';' before…`

Auf deutsch: Ich erwarte einen Strichpunkt vor der Zeile, die markiert ist. Diese Strichpunkte vergisst man gerne am Anfang.

`expected ')' before…`

Es fehlt eine Klammer oder du hast einen Strichpunkt an Stelle eines Kommas gesetzt.

`expected initializer before…`

Es fehlt eine geschweifte Klammer vor der markierten Zeile.

`…was not declared in this scope`

Der Arduino ist auf etwas Unbekanntes gestoßen. Vermutlich hast du dich bei einer Anweisung vertippt.

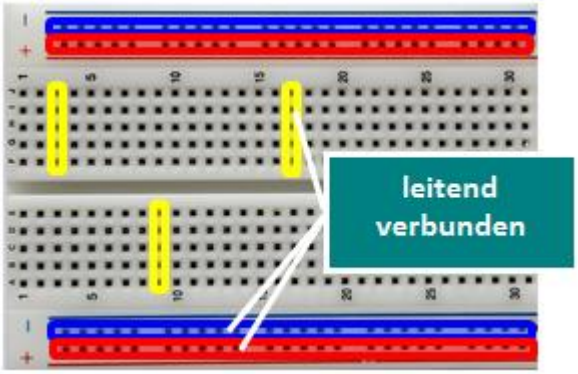
`Prombleme beim Hochladen`

Meistens hast du nicht die richtige COM-Schnittstelle ausgewählt. Schaue unter *Werkzeuge* → *Port* nach.

5. Verwendung einer Steckplatine: RGB-LED und 7-Segment-Anzeige

Eine **RGB-LED** ist eine LED, die im Inneren aus einer **r**oten, **g**rünen und **b**lauen LED besteht. So können wir über die drei kurzen Beinchen der RGB-LED die verschiedenen Farben ein- und ausschalten; das vierte Beinchen dient als gemeinsamer GND-Anschluss.

Die RGB-LED soll nicht direkt auf den Arduino aufgesteckt werden, sondern auf eine sog. **Steckplatine** (auch **Breadboar**d genannt) verschaltet werden.
Breadboards dienen dazu, Kabel ohne Löten elektrisch miteinander zu verbinden.



Dabei gilt:

- Die seitlichen Kontakte (blau bzw.rot umrandet) sind über die ganze Länge leitend miteinander verbunden. Sie dienen der „Spannungversorgung“ der Schaltung.
- Zusätzlich sind die 5er-Reihen (gelb umrandet) leitend miteinander verbunden.

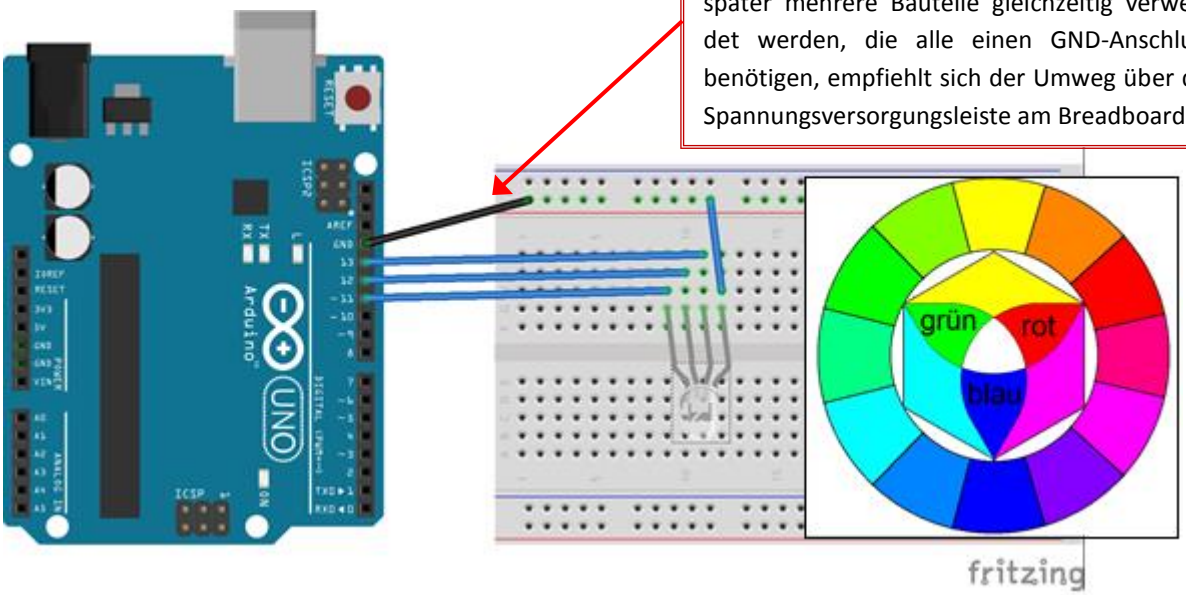
Aufgabe 5.1

a) Baue die Schaltung von unten nach.
Stülpe einen Tischtennisball über die RGB-LED, um die Farben besser erkennen zu können.
Lass die RGB-LED nacheinander in den Grundfarben rot, grün und blau leuchten.

b) Mit der RGB-LED lassen sich auch andere Farbeindrücke erzeugen. Lasse die RGB-LED gelb leuchten und notiere alle Farbeindrücke, die auch noch möglich sind.
Tipp: Additive Farbmischung (der Farbkreis unten rechts hilft dir weiter).

c) ✍ Stell dir vor, du hättest eine LED mit vier verschiedenen Farben.
Wie viele verschiedene Farbeindrücke sind hier möglich?

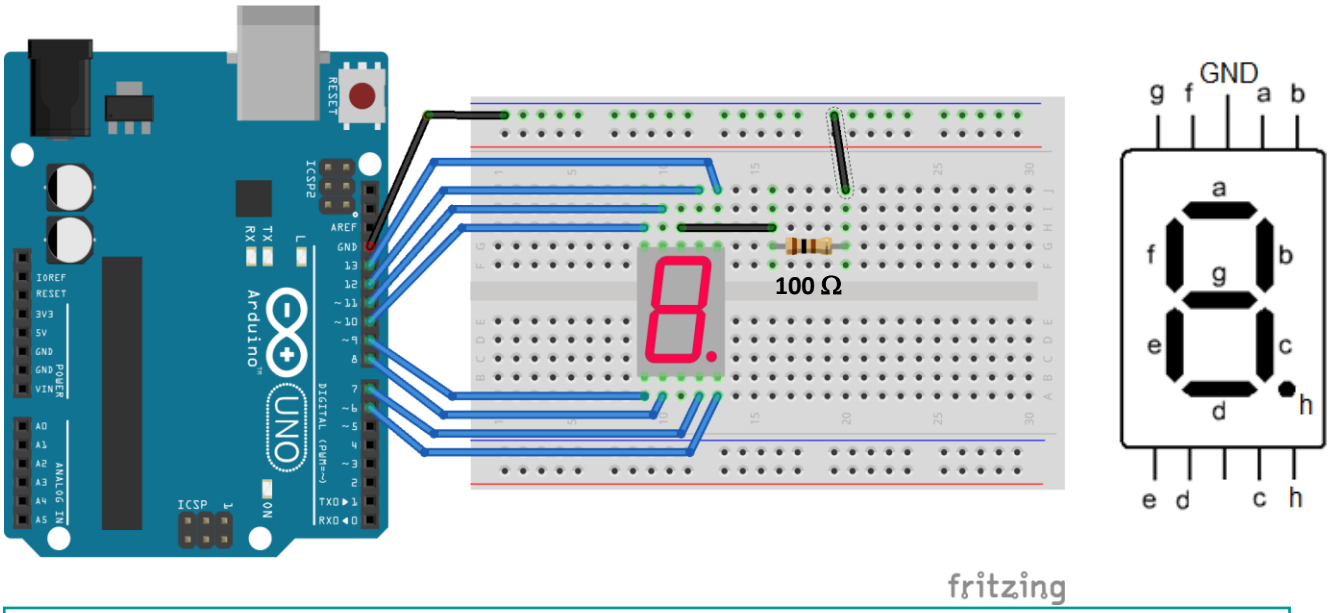
Das schwarze GND-Kabel könnten wir auch direkt an die LED anschließen. Wenn jedoch später mehrere Bauteile gleichzeitig verwendet werden, die alle einen GND-Anschluss benötigen, empfiehlt sich der Umweg über die Spannungsversorgungsleiste am Breadboard.



Die RGB-LED würde man sicherlich auch ohne Steckplatine noch an den Arduino anschließen können. Es gibt aber viele Bauteile mit noch mehr Anschlüssen, bei denen die Verwendung einer Steckplatine unumgänglich ist. So besitzt eine **7-Segment-Anzeige** (sie heißt so, weil sie aus sieben einzelnen LEDs besteht, die in einem Gehäuse verbaut sind) 10 Anschlüsse, die auf einer Steckplatine übersichtlich verschaltet werden können. 7-Segment-Anzeigen werden z.B. in Fahrstühlen zur Anzeige der Stockwerke benutzt. Die sieben LEDs besitzen einen gemeinsamen GND-Anschluss.

Wir werden mit Hilfe einer 7-Segment-Anzeige nun einen Countdown für einen Raketenstart realisieren.

Schaltbild:



<u>Aufgabe 5.2</u> „Countdown“

a) Baue die Schaltung von oben nach. Lass dir die Ziffer „4“ anzeigen.

b) Programmiere einen Countdown-Zähler. Er soll in Sekundenschritten von 9 auf 0 zählen.

(Tipp: Notiere dir zu jedem Buchstaben (a bis h) den entsprechenden Anschluss-Port des Arduino.)

Du hast sicher schon gemerkt, dass ein Programmtext schnell sehr unübersichtlich wird. Um ihn übersichtlicher zu gestalten kann man einzelne Programmbausteine in sog. **Unterprogrammen** auslagern. Zum Beispiel kann die Anzeige der Ziffer „1“ in einem Unterprogramm mit dem Namen „Eins“ erfolgen.
Aufgerufen werden die Unterprogramme mit Hilfe ihres Namens dann im Hauptprogramm an der Stelle, an der sie ausgeführt werden sollen.

```
void loop() {
 Eins();
}

void Eins() {
 digitalWrite(13,HIGH);
 digitalWrite(7,HIGH);
}
```

<u>Aufgabe 5.3</u>

Lass die Ziffern deiner Telefonnummer nacheinander auf der 7-Segment-Anzeige aufleuchten.
Verwende für die einzelnen Ziffern Unterprogramme.
(Tipp: Verwende ein Unterprogramm um alle LEDs auszuschalten.)
<u>Für Schnelle:</u> Lass dir die einzelnen Buchstaben des Wortes „HALLO“ darstellen.

6. Vorwiderstände von LEDs

Die LEDs, die wir verwenden, betreibt man idealerweise mit einer Stromstärke von ca. 20 mA. Dieser Strom fließt in etwa, wenn an der LED eine Spannung von ca. 2 V (sog. „Vorwärtsspannung“) anliegt.

Wenn wir die LED direkt an den 5 V des Arduino anschließen würden, würde ein größerer Strom als die 20 mA fließen. Als Folge davon würde sich die Lebenszeit der LED (unter idealen Bedingungen ca. 10 Jahre Dauerbetrieb) deutlich verringern, d.h. sie würde früher kaputt gehen. Um das zu vermeiden, muss man den Strom, der durch die LED fließt, verringern. Das macht man, indem man die Spannung, die an ihr anliegt, von 5 V auf 2 V verringert.

Dazu baut man zusätzlich zur LED in Reihe einen sog. **Vorwiderstand** oder **Schutzwiderstand** in den Stromkreis ein. Dieser wird so dimensioniert, dass er die Versorgungsspannung von 5V auf 2 V, also um 3 V (sog. „Spannungsabfall“), reduziert. Zusätzlich wollen wir den Maximalstrom von 20 mA durch die LED fließen lassen. Da LED und Widerstand in Reihe geschalten sind, fließen die 20 mA auch durch den Widerstand.

Den idealen Vorwiderstand berechnet man mit Hilfe des Ohmschen Gesetzes:

$$\text{Aus} \quad U = R \cdot I \quad \text{folgt} \quad R = \frac{U}{I} = \frac{5\,\text{V} - 2\,\text{V}}{20\,mA} = \frac{3\,V}{0{,}02\,A} = 150\,\Omega$$

Wird zur LED also ein $150\,\Omega$–Widerstand in Reihe geschaltet, dann fließen durch sie ein Strom von 20 mA. Unsere LEDs betreiben wir deshalb mit Vorwiderständen in der Größenordnung von $100\,\Omega$ bis $200\,\Omega$.

Die allgemeine Formel zur Berechnung des Vorwiderstands einer LED lautet:

$$R_{Vorwiderstand} = \frac{U_{Versorgung} - U_{Vorwärtsspannung}}{I_{LED}}$$

Aufgabe 6.1: ✍ Berechne den Vorwiderstand, falls die LED wie folgt angeschlossen werden soll:

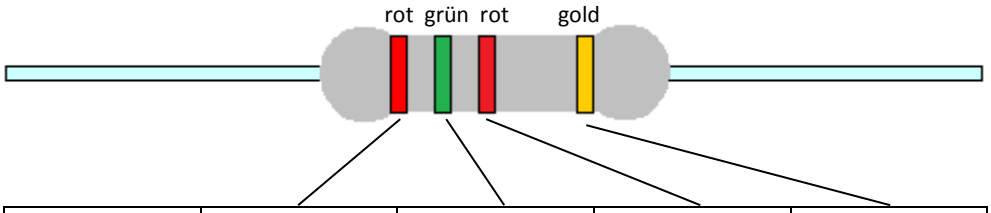
a) an einer Spannungsquelle mit 9 V,
b) an einer normale Steckdose (Achtung **LEBENSGEFAHR**: Nicht zu Hause ausprobieren!),
c) an einer Hochspannungsleitung (110 kV) .

Aufgabe 6.2: ✍ Drei gleiche LEDs sollen an einer Spannungsquelle mit 9 V
a) in Reihe geschaltet b) parallel geschaltet betrieben werden.
Zeichne jeweils einen Schaltplan und berechne den passenden Vorwiderstand.
Recherchiere wie das Schaltsymbol einer LED aussieht und verwende es korrekt in deinem Schaltplan.

Hausaufgabe 6.3
✍ Skizziere die U-I-Kennlinie einer roten und einer grünen LED in dein Heft.
Die beiden Kennlinien unterscheiden sich offensichtlich voneinander. Beschreibe den Unterschied und erläutere mit seiner Hilfe, warum grüne und rote LEDs eigentlich versch. Vorwiderstände benötigen.
(Internetquellen: https://de.wikibooks.org/wiki/Arbeiten_mit_LEDs/_Grundlagen oder http://www.leifiphysik.de/themenbereiche/halbleiterdiode/leuchtdioden-ubersicht → Menüpunkt: Funktionsweise und Kennlinie oder eigene Recherche.)

7. Farbcodes von Widerständen

Der Farbcode auf Widerständen besteht meistens aus vier Farbringen. Damit man weiß, auf welcher Seite man mit dem Ablesen beginnen muss, hat der letzte Ring einen etwas größeren Abstand von den anderen.



Farbe	1. Ring	2. Ring	3. Ring	4. Ring
schwarz	-	0	-	-
braun	1	1	0	± 1%
rot	2	2	00	± 2%
orange	3	3	000	-
gelb	4	4	0.000	-
grün	5	5	00.000	± 0,5%
blau	6	6	000.000	-
lila	7	7	-	-
grau	8	8	-	-
weiß	9	9	-	-
gold	-	-	× 0,1	± 5%
silber	-	-	× 0,01	± 10%

Tipp: Notiere auf Zetteln die Größe der Widerstände und lege ihn in ein Fach deiner Box

Die ersten zwei Ringe geben jeweils Ziffern an. Jeder Ring steht für eine Ziffer der Zahl gemäß diesem Farbcode. Beim oben abgebildeten Widerstand also: 25. Der Ring vor dem etwas größeren Abstand gibt an, *wie viele Nullen angefügt werden müssen*. Beim Widerstand oben müssen (roter Ring = 00) zwei Nullen angehängt werden; so ergibt sich der Wert des Widerstands zu $2500\ \Omega\ =\ 2{,}5\ \mathrm{k}\Omega$.

Der Ring ganz rechts (meist in silber oder gold) gibt an, wie genau die Angabe des Widerstands ist. Hier (gold = ± 5%) bedeutet das, dass der tatsächliche Wert unseres Wiederstand im Bereich von $2375\ \Omega$ bis $2625\ \Omega$ liegen kann. Den exakten Widerstandswert kann man z.B. mit einem Multimeter bestimmen.

Aufgabe 7.1

a) ✎ Welchen Farbcode hätte ein Widerstand mit 450 Ω und einer Toleranz von 10%?

b) ✎ Bestimme den Widerstandswert bei folgenden Farbkombinationen:

grün – orange – braun – gold und silber – rot – blau – lila.

Aufgabe 7.2 „Verkehrsampel“

Programmiere (ohne Unterprogramme) mit einer grünen, einer gelben und einer roten LED eine Ampel. Verwende für die LEDs einen gemeinsamen Vorwiderstand. (Probleme beim Aufbau? → s. Anhang B)

Für Schnelle: Füge zusätzlich noch eine Fußgängerampel hinzu.

8. Lautsprecher am Arduino betreiben

Mit dem Arduino kann man auch Töne und sogar Melodien an einem kleinen Lautsprecher abspielen. Der Lausprecher wird dabei mit einem digitalen Port und GND verbunden. Der zugehörige Befehl beim Arduino heißt `tone`:

`tone(`*Port, Frequenz (Tonhöhe), Dauer des Tones in ms*`);`

Der Port, an den der Lautsprecher angeschlossen wird, muss nicht extra als OUTPUT definiert werden. Auch dies erledigt der `tone`-Befehl.

```
void loop() {
 tone(11,440);
 delay(1000);
 noTone(11);
 delay(1000);
}
```

Ein Lautsprecher an Port 11 spielt einen Ton mit der Tonhöhe 440 Hz für 1000 Millisekunden (1 Sekunde). Die Zeit des delay-Befehls bestimmt die Länge des Tons.

Der `noTone`-Befehl schaltet den Lautsprecher am Port 11 aus. Nach einer Pause von 1000 ms beginnt das Programm von vorne.

Aufgabe 8.1:

Schließe einen Lautsprecher an Port 11 und GND an.

a) Der Arduino soll ohne Pause nacheinander zwei Töne abspielen.

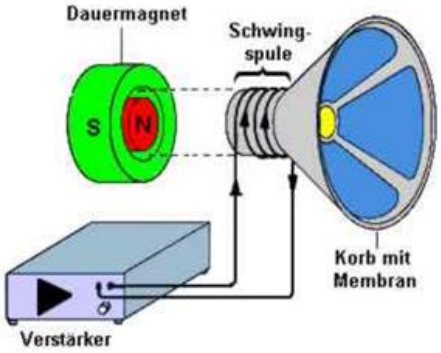
b) Der Arduino soll jeweils eine Sekunde Pause zwischen den Tönen machen.

Aufgabe 8.2:

Bestimme die Frequenz des höchsten Tones, den du noch hören kannst.

Wie funktioniert ein Lautsprecher?

Viele Leute meinen, dass ein Lautsprecher sofort einen Ton von sich gibt, wenn er an eine Stromversorgung angeschlossen wird. Das ist aber falsch: Ein Lautsprecher besteht nämlich nur aus einer elastisch aufgehängten Membran, einem Elektromagnet und einem Dauermagnet. Fließt Strom durch den Elektromagnet, bildet sich ein Magnetfeld aus und der Dauermagnet zieht den Elektromagnet und die mit ihm verbundene Membran an. Fließt kein Strom, fällt die Membran wieder zurück. Ein Ton entsteht, indem die Membran schnell abwechselnd angezogen und wieder losgelassen wird, also die Stromversorgung abwechselnd ein- und ausgeschaltet wird.



www.leifiphysik.de/themenbereiche/akustische-phaenomene/ausblick#Schallquellen

Daieses schnelle Ein- und Ausschalten könnte man prinzipiell „von Hand“ programmieren. Weil man aber Töne häufig braucht, gibt es den tone-Befehl, der sich um das schnelle Ein- und Ausschalten kümmert.

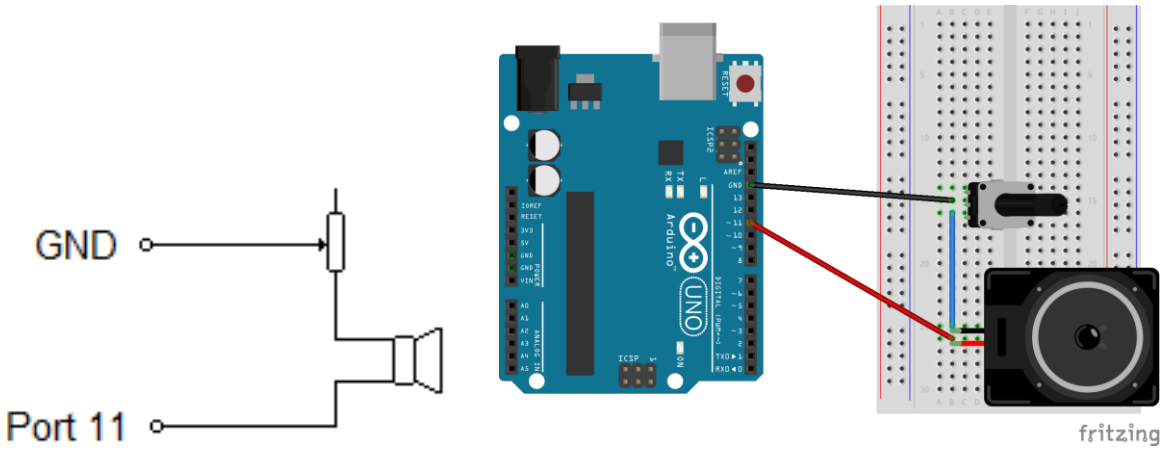
Weißt du übrigens, dass wir die Schwingfrequenzen der Membran nur zwischen etwa 20 Hertz und 20.000 Hertz mit Hilfe unserer Ohren als Ton wahrnehmen können?

Aufgabe 8.3

a) Programmiere eine Polizeisirene.

b) Schalte nacheinander verschiedene Widerstände (100 Ω, 220 Ω, 1 kΩ, 10 kΩ) in Reihe zum Lautsprecher. Was beobachtest du? Notiere im Heft

c) Baue in deine Sirene aus einen Lautstärkergler ein (siehe Abbildung unten)
Info: Unser Lautstärkeregler ist ein regelbarer Widerstand. Er heißt Potenziometer (s. Kapitel 17).



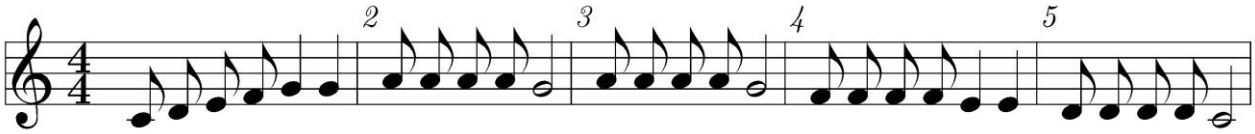
Aufgabe 8.4 (*Die erforderlichen Tonfrequenzen findest du bei der Klaviatur unten)*

Entscheide selbst:

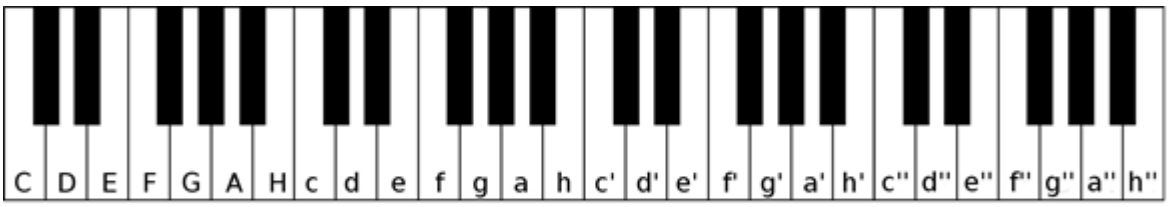
Variante a): Programmiere das Kinderlied „Alle meine Entchen“. Achte auf die richtigen Tonlängen!

Variante b): ❀ Spiel die Titelmusik eines bekannten Kinofilms (Noten erhältst du von deinem Lehrer).

Melodie von „Alle meine Entchen“:



Klaviatur mit den Frequenzen der einzelnen Töne:



Ton:	C	D	E	F	G	A	H	c	d	e	f	g	a	h	c'	d'	e'	f'	g'	a'	h'	c''	d''	e''	f''	g''	a''	h''
Frequenz (in Hz):	65	73	82	87	98	110	123	130	147	165	175	196	220	247	262	294	330	349	392	440	493	523	587	659	698	784	880	988

9. Rückmeldungen an den Nutzer – der serielle Monitor

In verschiedenen Situationen kann es sinnvoll sein, dass der Mikrocontroller eine Rückmeldung auf einen Ausgabebildschirm gibt (Beispiel: bei der Ampel: „STOPP“ und „Gehen“; beim Lautsprecher: Liedtitel angeben; bei Sensoren: Messwerte anzeigen). Oft wird ein Ausgabebildschirm auch für die Fehlersuche in Programmen eingesetzt: Wenn an bestimmten Stellen im Programm eine Rückmeldung einprogrammiert war, wusste man bei Erscheinen des Textes, dass das Programm bis zu dieser Stelle fehlerfrei durchgelaufen war.

Der Arduino kann über den **seriellen Monitor** Daten, die vom Arduino gesammelt oder produziert werden, anzeigen. Schließe dazu an Port 13 eine LED mit Vorwiderstand an und übernehme folgendes Programm:

```
int i=0;

void setup() {
 Serial.begin(9600);
 pinMode(13,OUTPUT);
}

void loop() {
 i=i+1;
 digitalWrite(13,HIGH);
 Serial.print("Anzahl:");
 Serial.println(i);
 delay(500);
 digitalWrite(13,LOW);
 delay(500);
}
```

Definiert eine Variable `i` als „integer“, d.h. als ganze Zahl, mit dem (momentanen) Wert 0.

Legt die Übertragungsrate für den Ausgabebildschirm fest. Bei uns immer 9600 (Bit pro Sekunde).

Der Wert von `i` wird pro Durchgang um jeweils 1 erhöht und fungiert ist hier als **Zählvariable**.

Der Text zwischen den Anführungszeichen wird auf dem Ausgabebildschirm angegeben.

Der Wert der Variablen `i` wird auf dem Ausgabebildschirm angegeben.

Achtung: Die Anführungszeichen werden beim Kopieren des Codes nicht korrekt übernommen.

In der Entwicklungsumgebung des Arduino findest du oben rechts den Button , mit dem du den Ausgabebildschirm (den „Serial Monitor“) öffnen kannst.

Aufgabe 9.1

a) Was ist der Unterschied zwischen `Serial.print("i");` und `Serial.print(i);` ?

b) Was ist der Unterschied zwischen
`Serial.print(„Hallo Welt“)` und `Serial.print`**`ln`**`(„Hallo Welt“);` ?

kleines L (println liest man als „Print Line“)

c) ✍ Notiere, was das Beispielprogramm von oben macht.

Aufgabe 9.2: Schreibe ein Programm, das ein „Gesicht“ auf dem Ausgabebildschirm erzeugt:

```
_~~~_
x   x
  o
|___|
```

Vielleicht fallen dir noch andere Muster ein?

Aufgabe 9.3:
Damit du flott und sicher programmieren kannst, musst du die Befehle des Arduino auswendig können. Suche dir jemanden zum gegenseitigen Abhören: Könnt ihr alle 10 Befehle und deren exakte Schreibweise und wisst obendrein, wie man ein Unterprogramm schreibt?

10. Variablen

Eine Variable ist ein „Platzhalter“ für eine Zahl. Dieser Platzhalter kann beliebig bezeichnet (Achtung: keine Umlaute!) und mit einem Wert belegt werden:

```
int rot = 2;
```

In diesem Beispiel wird die Variable mit dem Namen „**rot**“ als **Integer**-Variable definiert und ihr wird gleichzeitig der **Wert 2** zugeordnet. An jeder Stelle des Programms hat diese Variable nun den Wert 2.
Die Definition der Variablen erfolgt meist vor dem setup-Teil, sie kann aber auch erst im loop-Teil erfolgen.

Bemerkung: Je nachdem, welchen Wertebereich die Variable umfassen soll, muss ihr Datentyp entsprechend festgelegt werden (siehe nächste Seite). Je „kleiner“ der Datentyp gewählt wird, desto weniger Speicherplatz wird benötigt.

Im folgenden Programm wird zusätzlich noch eine zweite Variable `zeit` eingeführt. Welchen Vorteil hat ihre Verwendung hier wohl?

```
void setup() {
 pinMode(2, OUTPUT);
}

void loop() {
 digitalWrite(2, HIGH);
 delay(1000);
 digitalWrite(2, HIGH);
 delay(1000);
}
```

```
int rot = 2;
int zeit = 1000;

void setup() {
 pinMode(rot, OUTPUT);
}

void loop() {
 digitalWrite(rot, HIGH);
 delay(zeit);
 digitalWrite(rot, HIGH);
 delay(zeit);
}
```

Gut zu wissen: Vermeide beim Anschluss deiner Bauteile die Ports 0 und 1. Wenn die serielle Monitor benutzt wird, funktionieren diese Ports nicht mehr.

Die Verwendung von Variablen hat folgende Vorteile:

- Wird eine LED an einem anderen Pin angeschlossen, muss nur eine Zeile geändert werden.
- Der Sketch ist einfacher zu lesen und zu verstehen, vor allem wenn mehrere LEDs oder andere Bauteile (Motoren…) verwendet werden.

Aufgabe 10.1
Führe bei deinem Programm für die Ampel (Aufgabe 7.2) Variablen ein.
Speichere das Programm unter dem Namen „Ampel_mit_Variable“.

Hilfreiche Tastenkombinationen bei der Erstellung eines Programms

STRG + C	Kopiert einen vorher markierten Text
STRG + V	Fügt den kopierten Text an der Stelle des Textcursors wieder ein
STRG + Z	Macht die letzte(n) Aktion(en) rückgängig (mehrmals hintereinander anwendbar)
STRG + T	Auto-Formatierung eines Programms (korrigiert z.B. Einrückungen)

Wie speichert ein PC Zahlen?
In einem PC oder Mikrocontroller werden alle Informationen digital gespeichert, das heißt mit zwei Werten (0 oder 1). Dabei steht 1 Bit für einen Speicherplatz („Informationshäppchen“) mit dem Wert 0 oder 1. 1 Byte sind 8 Bit, also 8 Plätze, auf denen 0 oder 1 gespeichert werden kann. So können Zahlen im Binärsystem (Zweiersystem) gespeichert werden. Zum Beispiel entspricht die Zahl 13 im Binärsystem $(1101)_2$.

Eine Variable ist im Prinzip ein Stück Speicherplatz, auf den über den Namen der Variable zugegriffen werden kann. Im Beispiel oben hast du die Variablenart mit dem Namen `int` kennengelernt. Wenn in deinem Programm `int x;` steht, bedeutet dies:

*Lieber Arduino, reserviere mir im Speicher ein Stück Speicherplatz der Größe **int** und lass mich auf diesen Speicher über den Namen **x** zugreifen.*

„Int“ ist die Abkürzung für „integer“ und integer ist das englische Wort für „ganze Zahl“. Die Größe des für int reservierten Speicherplatzes beträgt bei unserem Arduino 16-Bit, also 2 Byte, d.h. es gibt 16 Plätze mit 0 oder 1 zum Speichern der ganzen Zahl.

Folgende Variablentypen gibt es unter anderem in der Programmiersprache C:

Type	Bits / Bytes	Zahlenumfang	Beschreibung
byte	8 Bit / 1 Byte	0 bis 255 (0 bis 2^8–1)	natürliche Zahl (d.h. ohne Komma)
int	16 Bit / 2 Bytes	–32 768 bis 32 767 (–2^{15} bis 2^{15}–1)	ganze Zahl mit Vorzeichen
float	32 Bit / 4 Bytes	zwei Nachkommastellen Genauigkeit	rationale Zahl
long	32 Bit / 4 Bytes	-2 147 483 648 bis 2 147 483 647	ganze Zahl mit Vorzeichen

Bemerkungen:

- Integer- und long-Variablen werden bei Überschreiten der Limits „**überlaufen**“. Wenn zum Beispiel int x = 32767 ist und eine Anweisung (z.B. x = x + 1) 1 zu x hinzu addiert, wird 'x' dabei „überlaufen“ und den Wert -32768 annehmen.
- Neben den Variablen für Zahlen gibt es in der Programmiersprache auch Variablen, die sich Text merken können. Diese Variable heißt **char**. Ein char kann einen Buchstaben speichern.
- Ein vorangestelltes `unsigned` macht aus `int` bzw. `long` eine **vorzeichenlose** Variable mit doppelt so großem positiven Zahlenumfang. (Beispiel: *unsigned int* umfasst die Werte 0 bis 65 535)

Aufgabe 10.2

a) ✎ **Notiere**, was der Monitor bei den ersten beiden „Überläufen“ anzeigt, wenn das Programm rechts abläuft. Führe das Programm am Arduino erst aus, **nachdem du die Zahlen notiert hast.** (Tipp: Im seriellen Monitor Häkchen bei „Automatisch scrollen“ entfernen)

b) Ersetze im Programm „byte“ durch „int“. Wiederhole a) und erkläre die Unterschiede!

```
byte i=0;

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

void loop() {
 Serial.print("Aktueller Wert:");
 Serial.println(i);
 i=i+10;
 delay(5);
}
```

Aufgabe 10.3

✎ Stell dir vor es gibt eine 4-Bit-Variable, die **horst** heißt. Mit welchen Zahlen kann **horst** rechnen?

11. LCD

Auf Leuchtreklamen in großen Städten laufen bunte Werbetexte über einen Bildschirm. Auch an den Arduino kann man ein Leucht-Display, ein sog. **LCD** (engl. **L**iquid **C**rystal **D**isplay), anschließen und betreiben.

Wir verwenden ein 16x2 Modul, d.h. das Anzeigefeld besteht aus 16 Spalten und 2 Zeilen. Um unser LCD verwenden zu können, benötigt man sog. **Bibliotheken** (siehe Anhang C).

Aufgabe 11.1

Schließe das LCD an den Arduino an (schwarz → GND, rot → 5 V, weiß → Port 2).
Sobald es angeschlossen wurde, wird kurz der Hersteller und die Typbezeichnung des LCD angezeigt.

a) Übertrage das Programm auf den Arduino. Was bedeutet `(1,0)` und `(2,4)` anschaulich?
b) Lass deinen Namen auf dem LCD anzeigen.
c) Lass deinen Namen auf dem LCD von links nach rechts und wieder zurück laufen.

Programmtext:

```
#include <SoftwareSerial.h>     // Die Bibliothek „SoftwareSerial" wird geladen
#include <SerLCD.h>             // Die Bibliothek „SerLCD" wird geladen

SoftwareSerial mylcd(0,2);      // RX-Pin des LCD ist an Port 2 angeschlossen
SerLCD lcd(mylcd);

void setup() {
 mylcd.begin(9600);
 lcd.begin();
}

void loop() {
 lcd.setPosition(1,0);  lcd.print("Hallo Besitzer");
 lcd.setPosition(2,4);  lcd.print("Wie gehts?");
}
```

Wichtige LCD-Befehle und ihre Bedeutung:

`lcd.setPosition(Zeile,Position);`	geht im Display an die entsprechende Position
`lcd.clear();`	löscht sämtlichen Text wieder vom Display

Aufgabe 11.2

Erweitere das Programm von oben um einen netten Gruß an deinen Teampartner.
Die beiden Textblöcke „Hallo Besitzer – wie geht's?" und dein Text sollen im Sekundentakt abwechselnd auf dem LCD erscheinen.

Der millis-Befehl

Der Befehl `millis()` gibt dir die Zeit in **Milli**sekunden an, seit der das aktuelle Programm läuft.

Aufgabe 11.3

Auf dem LCD soll folgender Text erscheinen: „*Programmstart vor … Sekunden*".

12. for-Schleife

Viele neuere Autos verfügen über einen sogenannden *dynamischen Blinker*. Dabei leuchten die einzelnen LEDs des Blinkers kurz nacheinander auf, bis am Ende das gesamte Blinklicht am Stück leuchtet. Rechts siehst du eine Momentaufnahme des Blinkers des Audi A6.
Anstatt die LEDs des A6 alle nacheinander einzeln anzuschalten, kann man die Programmierung mit einer **for-Schleife** einfacher und schneller realisieren. In diesem Kapitel lernst wie das geht.

Quelle: www.motor-talk.de/bilder

Das folgende Beispielprogramm zeigt, wie eine for-Schleife „funktioniert“:

```
int led = 10;
int i = 1;
int zeit = 500;

void setup() {
  pinMode(led,OUTPUT);
}

void loop()   {
 for( i;  i<5;  i=i+1) {
  digitalWrite(led,HIGH);
  delay(zeit*i);
  digitalWrite(led,LOW);
  delay(zeit*i);
 }
}
```

Die for-Schleife besteht aus drei Teilen:

1. <u>Definition der Zählvariablen:</u> `int i=1`
   Einmalig zu Beginn der Schleife wird festgelegt, um welchen Variablentyp es sich handelt und welchen Startwert die Zählvariable hat.
   Hier im Beispiel ist die Variable i vom Typ integer. Der Startwert beträgt 0.

2. <u>Zustand/Bedingung:</u> `i<5`
   Bei jedem Durchgang der Schleife wird überprüft, ob die Bedingung(i <5) erfüllt ist. Wenn ja, wird die Schleife ausgeführt, wenn nein, wird sie nicht mehr ausgeführt.
   Hier wird die Schleife ausgeführt, solange der Wert von i kleiner als 5 ist (also für i = 1; 2; 3 und 4)

3. <u>Änderung:</u> `i=i+1`
   Nach jedem Durchgang wird der Wert der Zählvariable i neu festgelegt.
   Hier wird der Wert von i nach jedem Durchgang um 1 erhöht.

Diese Programmzeilen werden so lange wiederholt, bis die Bedingung nicht mehr erfüllt ist. Damit das Programm weiß, welche Zeilen mit der for-Schleife wiederholt werden sollen, werden diese Zeilen in eigenen geschwungenen Klammern zusammengefasst.

Man kann die Zählvariable auch direkt im for-Befehl definieren: `for(int i=1; i<5; i=i+1)`. Dann wird die Zählvariable nach jedem loop-Durchlauf wieder auf den alten Wert zurückgesetzt.

Beachte, dass der Ausdruck **i=i+1** hier keine mathematische Gleichung darstellt! Statt diesem Ausdruck kann man auch einfach **i++** schreiben. Entsprechend gibt es den Ausdruck **i = i–1** bzw. **i––**.

<u>Aufgabe 12.1</u>
Was wird man beobachten, wenn man an Port 10 eine LED anschließt und anschließend das Beispielprogramm auf den Arduino überträgt? Überlege <u>zuerst</u> und teste anschließend.

Als weitere Anwendung erzeugen wir mit Hilfe einer `for`-Schleife verschiedene ansteigende Töne:

```
int lautsprecher = 10;

void setup() {
}

void loop() {
 for(int i=100; i<570; i=i+50) {
  tone(lautsprecher,i);
  delay(1000);
 }
}
```

Der Variablen mit dem Namen „lautsprecher“ wird der Port 10 zugewiesen.

Die Zählvariable i wird definiert und bekommt den Wert 100 zugewiesen.
Nach jedem Durchgang wird der Wert von i um 50 erhöht; und zwar so lange, wie i < 570 ist.

Der Lautsprecher spielt einen Ton der Höhe i. Das ist der aktuelle Wert der Zählvariablen.

Achtung: Am Ende der for-Zeile wird kein Semikolon gesetzt!!!

Aufgabe 12.2

a) Welche Töne werden vom Lautsprecher gespielt, bevor die komplette „Tonreihe“ von vorne wiederholt wird? Notiere die Frequenzen der einzelnen Töne.

b) Verändere die for-Schleife so, dass der Lautsprecher Töne vom Ton a‘ (440 Hz) in 10-Hz-Schritten abwärts bis zum Ton a (220 Hz) abspielt. Notiere die neue for-Schleife.

Aufgabe 12.3 „Sirene“

Programmiere eine amerikanische Polizeisirene (auf- und absteigende Sirene), bei der die Tonhöhe mit Hilfe einer for-Schleife automatisch in 1 Hz-Schritten verändert wird.

Aufgabe 12.4 „Dynamischer Blinker“

Baue und programmiere den auf der Vorderseite beschriebenen dynamischen Blinker des Audi A6.

Aufgabe 12.5 „Lauflicht“

Wenn mehrere LEDs schnell hintereinander an- und wieder ausgehen, spricht man von einem **Lauflicht**. Das Auto *KITT* aus der Serie *Knight Rider* verfügt zum Beispiel über ein solches Lauflicht.
Baue und programmiere mit einer for-Schleife ein Lauflicht, das aus mindestens 6 LEDs besteht und vor und wieder zurück läuft.
Verwende für die LEDs einen gemeinsamen Vorwiderstand.

Quelle: www.wikipedia.org/wiki/Knight_Rider

Aufgabe 12.6 „Lauftext“

Lass deinen Namen von links nach rechts und wieder zurück über das LCD „wandern“.

Aufgabe 12.7 „Quadratzahlen“

Der Arduino kann auch rechnen, das Zeichen „*“ dient dabei als Mal-Zeichen.
Lass dir auf dem seriellen Monitor die Quadratzahlen von 1^2 bis 20^2 anzeigen.
Verwende dazu eine for-Schleife.
Auf dem seriellen Monitor soll zunächst die Rechnung gestellt werden (z.B. „3 x 3“), nach kurzer Wartezeit soll der Arduino das Ergebnis anzeigen.

13. Programmablaufpläne (PAP)

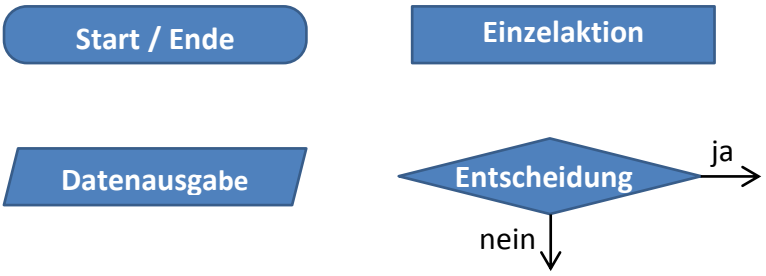
Wenn Programme zunehmend komplexer werden, verliert man schnell mal den Geamtüberblick über das komplette Programm.

Um diesen wieder zu erlangen hilft ein sog. **Programmablaufplan** (PAP). Ein PAP ist eine grafische Darstellung eines Programm und beschreibt die Abfolge von Operationen zur Lösung einer Aufgabe.

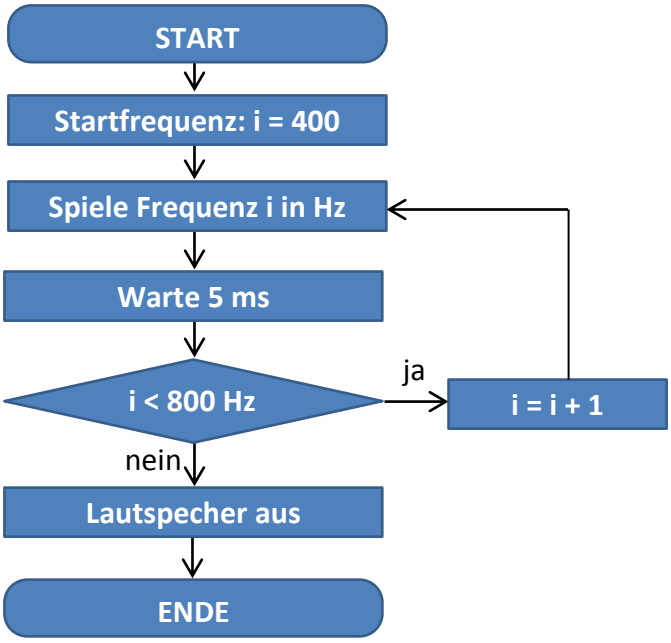
Den PAP für eine Verkehrsampel siehst du rechts abgebildet.

Allgemein kann man für jedes unserer Programme einen PAP zeichnen, der das komplette Programm kurz aber vollständig abbildet.

Für die Darstellung stehen vier verschiedene Formen zur Verfügung:



Der PAP einer aufsteigenden amerikanischen Polizeisirene könnte z.B. folgendermaßen aussehen:



Aufgabe 13.1
Erstelle den PAP zum Lautsprecher-Programm, das auf Seite 19 oben abgebildet ist.

14. while-Schleife und random-Befehl

Bei der for-Schleife wird ein Programmteil so lange wiederholt, wie eine bestimmte Bedingung erfüllt ist. Eine andere Möglichkeit einen Programmteil endlich oft zu wiederholen, bietet die **while-Schleife**. Sie wird ebenfalls nur so oft wiederholt, solange eine bestimmte Bedingung erfüllt ist.

Als Beispiel lassen wir den Arduino mithilfe des Befehls `random` (Erklärung siehe unten) eine Zahl „würfeln“. Sobald er eine 6 würfelt, soll er dies melden und eine Pause machen.

while-Schleife:

```
int a = 0;

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

void loop() {
 while(a!=6) { // != bedeutet ungleich
   a = random(1,7);
   delay(500);
   Serial.println(a);
 }
 Serial.println("Treffer!");
 delay(2000);
 a = 0;
}
```

Der Arduino kann Bedingungen nur mathematisch abfragen, indem er die Werte von Variablen mit Zahlen vergleicht. Hierfür gibt es folgende logische Operatoren:

<	kleiner	bzw.	> größer
<=	kleiner gleich	bzw.	>= größer gleich
==	gleich	bzw.	!= ungleich

Unterschied zwischen „=“ und „==“
Bei der Schreibweise „a=1“ handelt es sich um eine **Zuordnung**: Der Variablen a wird der Zahlenwert 1 zugeordnet. Die Schreibweise „a==1“ dagegen wird oft in Bedingungsschleifen verwendet. An dieser Stelle wird überpüft, ob a den Wert 1 besitzt. Das „==“ kann deshalb mit „ist gleich?“ übersetzt werden.

Aufgabe 14.1

a) Was ändert sich beim Programm von oben, wenn man das „`a=0`“ in der letzten Zeile weglässt?

b) Übernimm den Programmtext von oben lass den Arduino mehrmals „würfeln“.
Starte die Würfelfolge durch drücken auf „Reset“ mehrmals neu. Was fällt dir an den Zahlen auf?

Der `random`-**Befehl** erzeugt eine ganze Zufallszahl. Dabei gibt es zwei verschiedene Schreibweisen:
`random(10,20)` erzeugt eine Zufallszahl aus dem Zahlenbereich zwischen 10 und 19 (größtmögliche Zufallszahl ist also die zweite Zahl minus 1).
`random(10)` erzeugt eine ganze Zufallszahl von 0 bis 9.
Du hast beim mehrmaligen Ausführen des Würfel-Programmes von oben vielleicht festgestellt, dass der Arduino jedesmal dieselben „Zufalls“-Zahlen erzeugt, d.h. die Zufallszahlen des Arduino sind gar nicht zufällig. Der Arduino berechnet seine Zufallszahlen nämlich mit einer (uns unbekannten) Formel, in die er zu Beginn jedes Mal dieselbe Anfangszahl einsetzt. Um für uns brauchbare Zufallszahlen zu erzeugen, müssen wir bei jedem Durchlauf einfach verschiedene dieser Anfangszahlen (sog. seeds) verwenden. Dazu schreiben wir in die Zeile vor dem `random`-Befehl folgendes: `randomSeed(analogRead(A0));`

Verwende im Folgenden zur Erzeugung „echter“ Zufallszahlen jeweils den `randomSeed`-Befehl.

Aufgabe 14.2

Verändere das Beispielprogramm auf der Vorderseite folgendermaßen:
Ein 12-seitiger Würfel soll so lange gewürfelt werden, bis die Zahl 12 fällt.
Das Programm soll anschließend ausgeben nach wie vielen Würfen die 12 gefallen ist.

Aufgabe 14.3

Erstelle einen PAP zum Beispielprogramm auf der Vorderseite.

Aufgabe 14.4 „Kopfrechenaufgaben-Generator“

Programmiere ein Kopfrechen-Übungsprogramm für Fünftklässler, mit dem diese das „große 1×1“ üben können (Multiplikationen von „1 × 1“ bis „20 × 20“). Erzeuge dazu zwei geeignete Zufallszahlen. Ein Schüler soll auf dem seriellen Monitor zehn Aufgaben der Form „7 × 12 =“ gestellt bekommen. Nach jeweils 5 Sekunden Bedenkzeit soll das Ergebnis angezeigt werden.
(Tipp: Hier brauchst du den random-Befehl und eine for-Schleife, nicht aber den while-Befehl)

Aufgabe 14.5

Ein normaler Würfel soll 600-mal geworfen und dabei gezählt werden, wie oft die „6“ gefallen ist. Berechne den Durchschnitt der geworfenen Augenzahl. Wiederhole den Versuch mehrmals und vergleiche deinen Durchschnittswert mit dem theoretisch zu erwartenden Wert.

Aufgabe 14.6 „Tafeldienstgenerator“

Aus einer Klasse mit 20 Schülern sollen zwei Schüler für den Tafeldienst zufällig ausgewählt werden. Schreibe mithilfe der while-Schleife ein Programm, dass zwei **verschiedene** Zahlen zwischen 1 und 20 ausgibt.
Anleitung: „Würfle“ zweimal und speichere die „gewürfelten“ Werte in zwei Variablen a und b ab. Im Normalfall sind die beiden Zahlen bereits verschieden und können angezeigt werden. Falls („while“) beide Zahlen gleich sind (a == b), soll eine der beiden Zahlen neu gewürfelt werden.

Aufgabe 14.7 ❀ (Im Folgenden werden anspruchsvolle Aufgaben mittels ❀ gekennzeichnet)

Lass den Arduino nacheinander 100 Zufallszahlen im Bereich von 0 bis 1000 erzeugen.
Die größte der 100 Zahlen soll ermittelt und ausgegeben werden.

Vergleich von for-Schleife und while-Schleife

Prinzipiell kann eine while-Schleife in eine for-Schleife umgeschrieben werden und umgekehrt.
Beide Schleifenarten bestehen aus den Teilen *Variable*, *Bedingung* und *Änderung*:

```
for(Variable; Bedingung; Änderung)
{…}
```

```
Variable
while(Bedingung)
{… Änderung …}
```

15. Dimmen einer LED - Pulsweitenmodulation

Der Arduino ist ein digitaler Mikrocontroller. Er kann seine Ausgänge nur an- oder ausschalten, also nur 5V (HIGH) oder 0V (LOW) anlegen. Um die Helligkeit einer LED zu verändern, müsste man die Spannung jedoch variieren können. Zum Beispiel 5V, wenn die LED hell leuchten soll, 4V, wenn sie etwas dunkler leuchten soll usw. Das geht bei digitalen Ausgängen aber nicht!

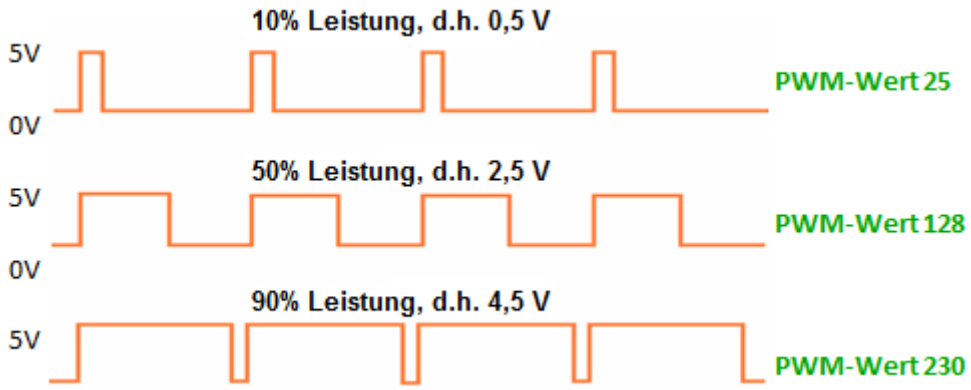
Es gibt dennoch eine Möglichkeit Spannungen zwischen 0 V und 5 V zu erhalten. Sie nennt sich **Pulsweitenmodulation** (kurz: **PWM**). Durch schnelles Ein- und Ausschalten eines Ports (Wechsel zwischen HIGH und LOW im Millisekundenbereich) wirkt es so, als ob am Port ein Potenzial **zwischen** 0V und 5V anliegt. Die Ports auf dem Arduino, die mit (~) versehen sind, können ein PWM-Signal erzeugen.
Allgemein sind PWM-Ports in der Lage 256 verschiedene Zustände einnehmen (0 bis 255). Das bedeutet, die Ports 1 bis 13 können als digitale und als analoge Ausgänge benutzt werden.

Aufgabe 15.1

✍ Notiere in deinem Heft alle PWM-Ports des Arduino.

Der folgende Befehl erzeugt solch ein PWM-Signal: **analogWrite(pin, Wert);**

Der zeitliche Verlauf am Ausgang sieht bei verschiedenen **Werten** dann wie folgt aus:



Aufgabe 15.2

Übernimm das folgende Programm und probiere verschiedene PWM-Werte zwischen 0 und 255 aus. Teste auch, was passiert, wenn du Werte größer als 255 eingibst.

```
int led = 9;      // LED an PWM-Port 9

void setup()  {
}

void loop() {
 analogWrite(led, PWM-Wert 1); delay(500);
 analogWrite(led, PWM-Wert 2); delay(500);
}
```

Hinweis:
PWM-Ports müssen nicht im `setup`-Teil als OUTPUT definiert werden.

Analoge und digitale Signale
Signale (zum Steuern oder zum Messen) können generell **digital** oder **analog** sein. Digitale Signale haben nur zwei eindeutige Zustände, ein hohes und ein niedriges Niveau (anschaulich: „an“ oder „aus“).
Beispiel: `digitalWrite` kennt nur zwei Zustände – „LED an“ oder „LED aus“
Daneben können Signale analog sein. Diese Signale enthalten viele Zwischenstufen zwischen dem niedrigsten und dem höchsten Niveau. Rein analoge Signale besitzen einen kontinuierlichen Verlauf.
Beispiel: analog`Write` kennt viele (256!) Zwischenstufen – eine LED ist in 256 Helligkeitsstufen dimmbar.

Aufgabe 15.3

a) Verwende eine `for`-Zählschleife, um die Helligkeit der LED von 0 auf 255 zu steigern.
b) Schreibe ein Programm, das eine LED kontinuierlich heller und dann wieder dunkler werden lässt. Lass dir zusätzlich den PWM-Wert am seriellen Monitor anzeigen.

Aufgabe 15.4 „MEHR Farben“
Du hast bereits die RGB-LED kennen gelernt. Dabei konnten die Farbeindrücke rot, grün, blau, gelb, magenta, cyan und weiß erzeugt werden. Durch das Mischen der drei Grundfarben in unterschiedlichen Intensitäten lassen sich deutlich mehr Farben erzeugen.

a) Lass den Farbeindruck orange entstehen. (Schau die PWM-Werte in Word nach: *Schriftfarbe* → *Weitere Farben* → *Benutzerdefiniert* → Farbe orange anklicken und PWM-Werte ablesen)
b) ❀ Erzeuge einen schrittweisen Übergang von rot nach grün. (Schnelle: …. und weiter nach blau…)
Tipp: `analogWrite(rot,i)` und `analogWrite(gruen,255-i)` in passender for-Schleife.

Aufgabe 15.5

✍ Das folgende Programm enthält mehrere Fehler. Notiere im Heft die korrekte Version.
Zum Programm: Das folgende Programm soll einen Lautsprecher (Port 5) steuern und dessen gespielte Frequenzen untereinander auf dem Monitor ausgeben. Gleichzeitig ist an Port 13 eine LED angeschlossen, die jeweils blinken soll, sobald sich die Frequenz des Lautsprechers ändert.

```
i = 100;
void setup() {
 SerialBegin(6900);
 pinMode(13,Output)
}
void loop {
 for(i < 1000, i+50); {
  ton(i,5);
  delay(500);
  Serial.print("i");
  digitalWrite(13;HIGH);
  digitalWrite(13,LOW);
 }
}
```

Verständnisfragen

- Wie lange wird jeder Ton gespielt?
- Wie oft blinkt die LED insgesamt?

16. Der Arduino bekommt Sinne

Bisher hat der Mikrocontroller immer nur etwas GETAN. Er hat die Anweisungen ausgeführt, die du ihm gegeben hast. Wenn Menschen sich so verhalten würden, wäre die Welt eintönig – wir müssten nur tun was man uns sagt und wären gleichzeitig taub, blind und gefühlslos.
Abhilfe: Alle Ports, die du bisher als AUSGÄNGE kennengelernt hast, lassen sich auch als EINGÄNGE benutzen. Statt **digitalWrite() und analogWrite()** wird dazu der Begriff **digitalRead()** oder **analogRead()** benutzt.

Du baust jetzt mithilfe des Arduino ein Batterietestgerät, indem du das Potenzial des Plus-Anschlusses misst

```
int wahrnehmung;
void setup() {
 pinMode(13, INPUT);
}

void loop() {
 wahrnehmung = digitalRead(13);
 delay(500);
}
```

Setzt den Modus für Port 13 auf „Eingang“

Der Arduino liest das Potenzial, das am Port 13 anliegt, aus: Alles, was über ca. 2,3 Volt ist, wird vom Arduino als „an“, „HIGH“ bzw. „1“ wahrgenommen, Potenziale unter 2,3 Volt spürt er als „aus“, „LOW“ bzw. „0“. Der Wert (0 oder 1) wird dann an die Variable Wahrnehmung übergeben.

Aufgabe 16.1
Stecke ein Kabel in Port 13. Übernimm das Programm von oben. Lass dir zusätzlich den Wert für *Wahrnehmung* **auf dem seriellen Monitor** anzeigen.

a) Stecke nun das andere Ende des Kabels nacheinander in GND, in 3,3V und in 5V.
b) Batterietester Nr. 1:
Lege zwei 1,5V-Batterien in den Batteriehalter ein. Die beiden Batterien liegen dort „in Reihe“.
Stecke das schwarze Kabel des Batteriehalters in GND und das rote Kabel in Port 13.
Prüfe, ob die Batterien voll (Ausgabe: „1“) oder leer (Ausgabe: „0“) sind.

Wenn du die Spannung deiner Batterie genau bestimmen möchtest, hilft dir die Tatsache, dass die Ports A0 bis A5 nicht nur zwei, sondern 1024 verschiedene Potenzialwerte (0 bis 1023) einlesen können.

Je nachdem, welches Potenzial an den Ports anliegt, wird ein Wert zwischen 0 und 1023 ausgegeben. Die Schrittweite („Auflösung“) beträgt dabei 5V : 1023 ≈ 0,005V.
Damit ergibt sich die nebenstehende Tabelle.

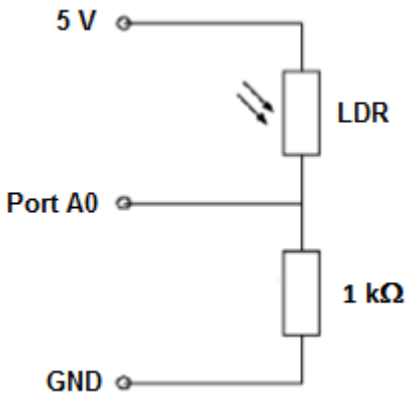
Potenzial am Port	analoger Wert
0 V	0
0,005 V	1
0,010 V	2
…	…
1,00 V	205
…	…
5,00 V	1023

Aufgabe 16.2 „Batterietester Nr. 2“
Ersetze in deinem Programm von oben `digitalRead`(13) durch `analogRead(A0)` und stecke das rote Kabel des Batteriehalters von Port 13 in Port A0.
Rechne mit dem Arduino den Analogwert in den entsprechenden einen Spannungswert (in Volt) um.
Überprüfe mit einem Spannungsmessgerät, ob dein Wert mit der Messung übereinstimmt.

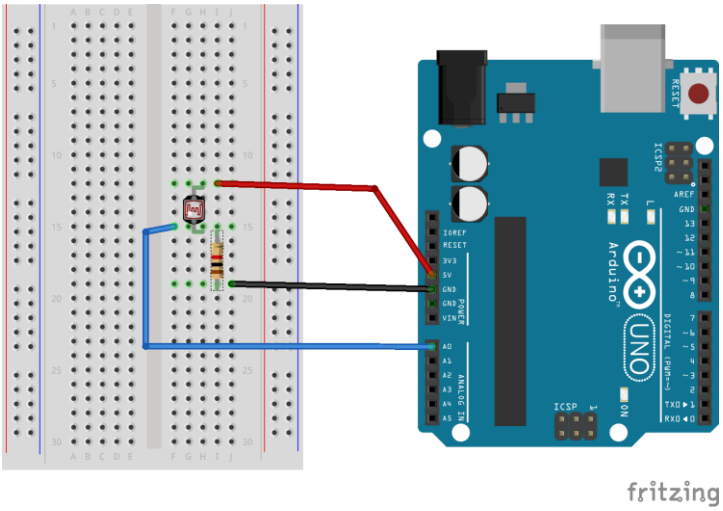
17.
Hell oder dunkel? Der Arduino wird zum Messgerät

Um die Helligkeit zu messen, verwenden wir einen lichtabhängigen Widerstand (**LDR**, engl. **L**ight **D**ependent **R**esistor). Der Widerstandswert eines LDR ändert sich, je nachdem, wie viel Licht auf ihn einfällt: *je mehr Licht, desto geringer der Wert*. Diese Eigenschaft benutzt man, um einen LDR als Lichtsensor zu verwenden. Der LDR kann jedoch nicht einfach wie ein Lautsprecher mit GND und einem Port verbunden werden. Wir schließen den LDR in einer sog. **Spannungsteilerschaltung** an den Arduino an (Erklärung: siehe Kapitel 19).

Schaltplan:



Anschluss an den Arduino:



Programmtext:

```
int ldr = A0;
int helligkeit;

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

void loop() {
 helligkeit = analogRead(ldr);
 Serial.println(helligkeit);
 delay(200);
}
```

Der LDR wird als Sensor an den analogen Eingang A0 angeschlossen.

Für das analoge Einlesen ist im setup-Teil <u>keine</u> INPUT-Definition nötig.

Der Variablen Helligkeit wird der aktuelle Helligkeitswert des LDR (0…1023) zugewiesen.

Der aktuelle Helligkeitswert wird auf dem seriellen Monitor angezeigt.

<u>Aufgabe 17.1</u>

a) Baue den Helligkeitssensor wie oben dargestellt auf und notiere folgende Widerstandswerte:
   - Maximale Beleuchtung des LDR
   - Minimale Beleuchtung des LDR (Abdecken mit <u>einem</u> Finger)

b) Ersetze den 1 kΩ-Festwiderstand aus der obigen Schaltung durch einen anderen und untersuche, wie sich das auf die Helligkeitswerte in a) auswirkt.
   Bei welchem Widerstand decken die Helligkeitswerte einen möglichst großen Bereich ab?
   Bestimme anschließend mit einem Multimeter den Widerstand des LDR. Fällt dir etwas auf?

c) Benenne die Variablen um (LDR in Sensor, Helligkeit in Sensorwert) und speichere das Programm unter „Sensorabfrage“. So kannst du es in Zukunft als Vorlage für alle Sensorabfragen verwenden.

Aufgabe 17.2

Tausche in deiner Spannungsteilerschaltung die Position von LDR und Festwiderstand.
Was beobachtest du beim Abdunkeln des LDR an den Analogwerten?

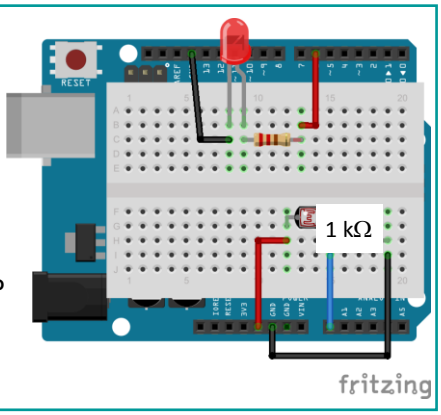
Kannst du dasselbe Ergebnis auch ohne Umstecken der **Bauteile** erreichen?

Aufgabe 17.3 „Helligkeitsanzeiger“

Erstelle einen einfachen Helligkeitsanzeiger:

a) Je dunkler es in einem Raum ist, desto schneller soll eine LED blinken. (Tipp: `delay(helligkeit)`)

b) Je dunkler es ist, desto heller soll die LED leuchten (Tipp: `analogWrite(helligkeit/4)`)
Welche Idee steckt hinter der Division der `helligkeit` durch 4?
Was verändert sich, wenn man diese Division weglässt?
Versuche deine Beobachtung zu erklären.



Du hast bei Aufgabe 17.1 gesehen, dass der LDR, je nach Wahl des Festwiderstandes, unterschiedliche große Bereiche der Analogwerte abdeckt. In der Praxis ist der gesamte Bereich von 0 bis 1023 aber so gut wie in keiner Situation zu erreichen (das liegt u.a. daran, dass vollständige Dunkelheit so gut wie nie auftritt).

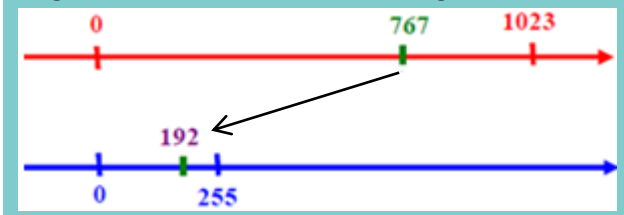
Das führt zu einem Problem, wenn man folgende Aufgabe erledigen möchte:
Eine LED soll bei abgedunkeltem Raum in voller Helligkeit an sein. Bei zunehmender Aufhellung des Raums soll die LED kontinuierlich heruntergedimmt werden, bis sie bei voller Raumhelligkeit schließlich aus ist.

Dieses Problem kann man mit dem **map-Befehl** (siehe folgender Kasten) in den Griff bekommen:

map-Befehl

Der `map`- Befehl wird verwendet, wenn verschiedene Bereiche ineinander umgerechnet werden müssen:
y = map(Wert, a, b, c, d) → rechnet einen Wert vom Intervall [a;b] in das Intervall [c;d] um.

Beispiel: Möchte man LDR-Werte (0…1023) in PWM-Werte (0…255) umrechnen, erledigt das der folgende Befehl: y = map(analogRead(ldr), 0, 1023, 0, 255)
Angenommen, der LDR-Wert beträgt aktuell 767 → dann wird der Variablen y der Wert 192 zugewiesen



Wichtig: Der mit dem map-Befehl umgerechnete Wert der Helligkeit muss an eine Variable (hier: y) übergeben werden!

Die Umrechnung erfolgt linear und das Ergebnis ist immer ein ganzzahliger Wert (integer).

Wir ermitteln also zunächst unseren eigenen Wert **max**imaler Helligkeit (Licht an) und den Wert **min**imaler Helligkeit (Licht aus, Raum abgedunkelt). Mit dem map-Befehl rechnen wir die Helligkeitswerte des LDR in PWM-Werte für die LED um.

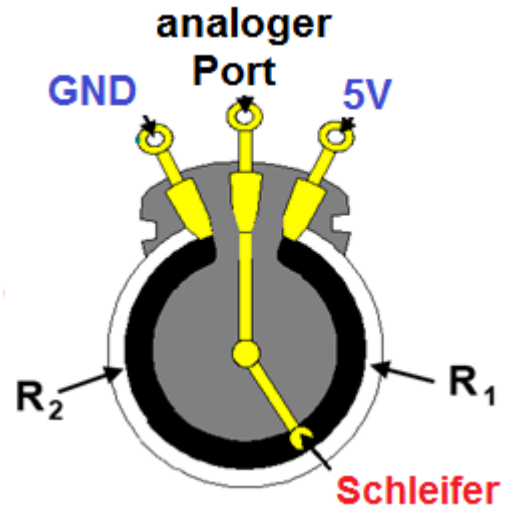
```
void loop() {
 helligkeit = analogRead(ldr);
 pwm = map(helligkeit,min,max,255,0);
 analogWrite(led, pwm);
}
```

Aufgabe 17.4 „Dämmerungsschalter“

Bestimme an deinem Platz die Helligkeitswerte **min** und **max**. Vervollständige das Programm.

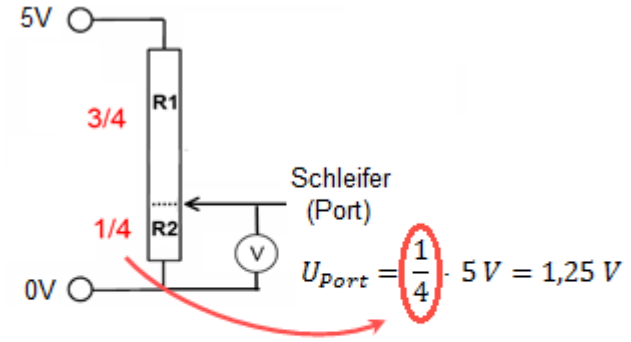
18. Potentiometer

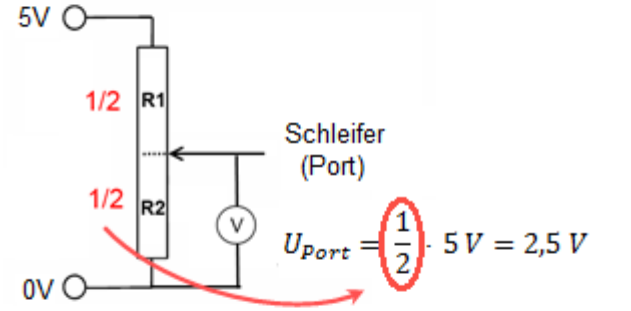
An vielen elektronischen Geräten findet man Regler, an denen man drehen kann, um beispielsweise die Lautstärke zu ändern. Dahinter stecken meist **Potentiometer**, kurz **Poti**.

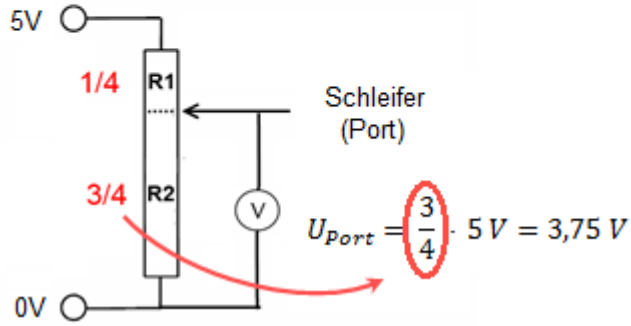


Aufbau eines Potentiometers

Das Potentiometer besteht aus drei Anschlüssen (s. Abb.). Zwei Anschlüsse an den Enden des Widerstandselements und ein beweglicher Gleitkontakt (auch **Schleifer** oder Mittelkontakt genannt). Der Schleifer bewegt sich durch Drehen des Knopfes auf einer Kohlebahn und teilt den festen Gesamtwiderstand in zwei Teilwiderstände R_1 und R_2 auf.







Prinzip der Spannungsteilung

Die Gesamtspannung (hier 5V) teilt sich auf die beiden Widerstände auf.
Am Mittelkontakt (Port) liegt dann die Spannung U_{Port} an, die am Widerstand R_2 abfällt:

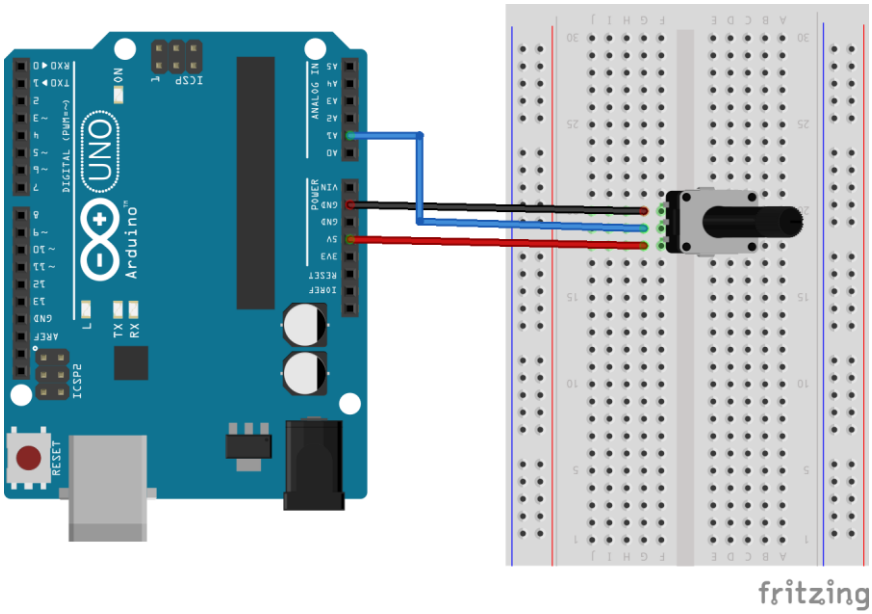
Allgemein gilt:

$$U_{Port} = \frac{R_2}{R_{ges}} \cdot 5V = \frac{R_2}{R_1 + R_2} \cdot 5V$$

Aufgabe 18.1

a) Baue die Schaltung wie in der Abbildung unten auf.
Lass dir den Wert des Analogeingangs auf dem seriellen Monitor anzeigen, wenn du am Poti drehst. (Tipp: Verwende dein Programm „Sensorabfrage“)

b) Gebt euch abwechselnd 10 verschiedene Werte vor, die der andere jeweils mit dem Poti einstellt.



Der Poti kann nun z.B. dazu verwendet werden, um andere Bauteile (LED, Lautsprecher, …) zu steuern.

Aufgabe 18.2

Ergänze die Schaltung um eine LED.
Schreibe ein Programm, so dass die LED blinkt. Die Blinkfrequenz soll mit dem Poti geregelt werden.

Aufgabe 18.3

Verwende ein Poti, um…

a) die Helligkeit der LED aus Aufgabe 18.2 zu steuern.

b) die Tonhöhe eines Lautsprechers zu steuern.

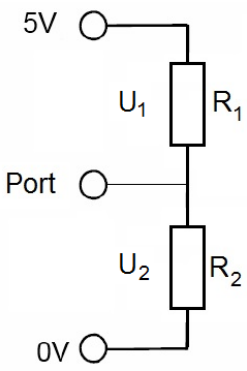
c) bei einer RGB-LED einen kontinuierlichen Farbwechsel von rot nach grün einstellen zu können.
Schnelle: ❀ Mit dem Poti soll ein kontinuierlicher Farbwechsel rot ↔ grün ↔ blau regelbar sein.

d) die Position deines Namens auf dem LCD zu verändern.

Hinweis: Verwende den `map`-Befehl, um den Analogwert des Poti aus dem Intervall [0; 1023] in das Intervall [0; 255] für PWM-Werte umzurechnen.
Beachte dabei, dass du den umgerechneten Wert an eine Variable (z.B. `int i`) übergibst, also z.B. **`int i =`** `map(poti,1,2,3,4)` und nicht nur `map(poti,1,2,3,4)` .

19. Spannungsteiler

Bei vielen Sensoren handelt es sich um Widerstände, die ihren Wert ändern, wenn sich z.B. die Temperatur oder die Helligkeit ändert. Um dies registrieren zu können, haben wir den LDR in einer sogenannten **Spannungsteilerschaltung** verwendet. Auch der Poti als Bauteil stellt einen Spannungsteiler dar. Die Idee, die hinter einer Spannungsteilung steckt, lernst du nun genauer kennen.



Diese Schaltung in der Abbildung rechts bezeichnet man allgemein als **Spannungsteiler** mit einem Mittelkontakt. Eine Spannungsteilerschaltung besteht immer aus zwei Widerständen (meistens ist einer davon veränderbar) und drei Anschlüssen (GND, 5V und Port). Da die beiden Widerstände in Reihe verbaut sind, ist der Gesamtwiderstand R_{ges} die Summe aus den Widerständen R_1 und R_2. Die beiden Widerstände teilen die Gesamtspannung (5V) entsprechend ihren Anteilen am Gesamtwiderstand in die Spannung U_1 und U_2 auf. Das Potential am Port entspricht U_2 und lässt sich aus den Widerstandswerten von R_1 und R_2 und der Gesamtspannung berechnen. Die Spannung U_2 lässt sich mit folgender Formel berechnen (dabei gilt: $U_{\text{ges}} = U_1 + U_2$):

$$\boldsymbol{U_2 = U_{\text{ges}} \cdot \frac{R_2}{R_1 + R_2}}$$

Herleitung der Formel für U_2 :

Da der Stromfluss in Reihe immer gleich ist, gilt: $I_2 = I_{\text{ges}}$

Für Ohmsche Widerstände gilt $U = R \cdot I$. Also auch : $I = \frac{U}{R}$ $\quad \frac{U_2}{R_2} = \frac{U_{\text{ges}}}{R_{\text{ges}}} \quad | \cdot R_2$

$$U_2 = U_{\text{ges}} \cdot \frac{R_2}{R_{\text{ges}}}$$

Mit $R_{\text{ges}} = R_1 + R_2$ erhält man $\quad U_2 = U_{\text{ges}} \cdot \frac{R_2}{R_1 + R_2}$

Aufgabe 19.1 Im Folgenden gilt $U_{\text{ges}} = 5V$. Berechne jeweils die Spannung U_2, wenn…

a) $R_1 = 5\ k\Omega$ und $R_2 = 10\ k\Omega$ gilt.
b) R_1 und R_2 gleich groß sind.
c) R_1 zehnmal so groß wie R_2 ist.
d) R_1 und R_2 durch zwei baugleiche LDR ersetzt werden.

Aufgabe 19.2
Berechne U_2 für eine Spannungsteilerschaltung mit den Widerständen $R_1 = 220\ \Omega$ und $R_2 = 1\ k\Omega$.
Baue dann die Schaltung auf, lass dir den Analogwert von U_2 seriellen Monitor anzeigen und vergleiche

Bau beliebiger Sensoren mit Hilfe einer Spannungsteilerschaltung
Die meisten Sensoren basieren, wie beim LDR, auf dem Prinzip des Spannungsteilers:
Möchte man zum Beispiel ein Thermometer selbst bauen, benötigt man einen veränderlichen Widerstand R_{Sensor} , dessen Größe sich mit der Temperatur ändert. Man muss diesen nur mit einem Festwiderstand R_{fest} in einer Spannungsteilerschaltung verbauen und das Potential am Mittelkontakt auslesen – fertig.
So lassen sich beliebige Sensoren „bauen“, z.B. für Luftfeuchtigkeit, Druck, Lautstärke, …

20. Der Arduino als Thermometer

Im Folgenden sollst du verstehen, wie ein Digital-Thermometer (z.B. ein Fieberthermometer) funktioniert. Du baust selbst einen solchen Thermometer aus seinen Grundbestandteilen auf und programmierst ihn.

Wir verwenden als Temperatursensor einen sog. **NTC-Widerstand** (engl. **N**egative **T**emperatur **C**oeffizient). Oft sagt man statt „NTC-Wideestand“ auch einfach nur „NTC“. Unser NTC wird auch als Heißleiter bezeichnet. Warum das so ist, siehst du in der folgenden Aufgabe:

Aufgabe 20.1 „Thermometer“

a) Nimm den NTC in die Hand und messe seinen Widerstand bei Raumtemperatur und in der warmen Hand. Wie ändert sich sein Widerstand mit der Temperatur? Bilde zwei "*Je ..., desto ...*"-Sätze.

b) Baue nun eine Spannungsteilerschaltung auf, wobei du den zweiten Widerstand etwa gleich groß wie den NTC-Widerstand wählst. Welche Werte misst du am Analogeingang A0?
Benutze den seriellen Monitor, um dir die Analogwerte anzeigen zu lassen.
(Tipp: Verwende zum Auslesen der Sensorwerte dein Programm „Sensorabfrage“)

c) Messe zeitgleich so exakt wie möglich(!) mit einem Referenzthermometer die Wassertemperatur in einem Glas und notiere das Wertepaar (analoger Sensorwert | Referenztemperatur).
Wiederhole die Messung mit Wasser, dessen Temperatur sich möglichst stark von der ersten Messung unterscheidet.
Rechne die gemessenen analogen Sensorwerte mit dem `map`-Befehl in die Temperatur in °C um.
Speichere dein Programm anschließend unter dem Namen „Thermometer“ ab.

d) Teste dein Thermometer in einem Wasserglas vorne auf dem Lehrerpult.

In vielen Situatonen ist der exakte Wert einer physikalischen Größe gar nicht sowichtig. Ein klassischer Bewegungsmelder, den du von zu Hause kennst, detektiert Wärmestrahlung und schlägt nur dann an, wenn sich die Wärmestrahlung in seinem Blickfeld merklich ändert. Er funktioniert dabei bei sommerlichen Temperaturen oder klirrender Kälte gleichermaßen gut. Es kommt nur darauf an, ob sich der aktuelle „Wärme-Wert“ rasch ändert. Das Detektieren der Änderung einer Größe lernst du in der nächsten Aufgabe kennen.

Aufgabe 20.2 „Elektrisches Teelicht“

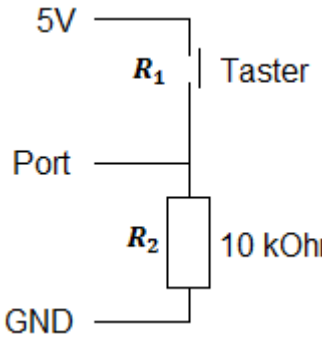
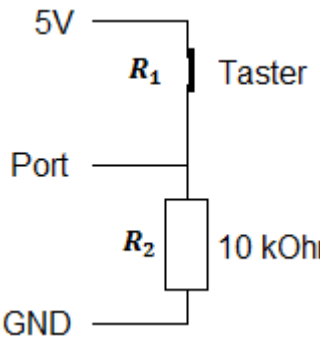
Baue und programmiere mit Hilfe des NTC und einer LED ein elektrisches Teelicht:
Die LED (das Teelicht) ist zunächst an.
Wenn man kräftig auf den NTC-Widerstand pustet, soll die LED erlöschen.

Anleitung: Miss zweimal kurz nacheinander die aktuelle Temperatur:
`a = analogRead(ntc); delay(…); b = analogRead(ntc);`
Wenn sich die Temperaturwerte durch das Pusten merklich verringern (falls also z.B. a – b > 10; konkret: `while(a-b > 10)`), dann soll die LED erlöschen.

Schnelle ergänzen die folgende Option: Die LED ist zu Beginn aus und wird erst durch das Erwärmen des NTC von Hand „entzündet“ (zweite while-Schleife einbauen: `while(a-b < -10)`).

21. Taster – Start per Knopfdruck

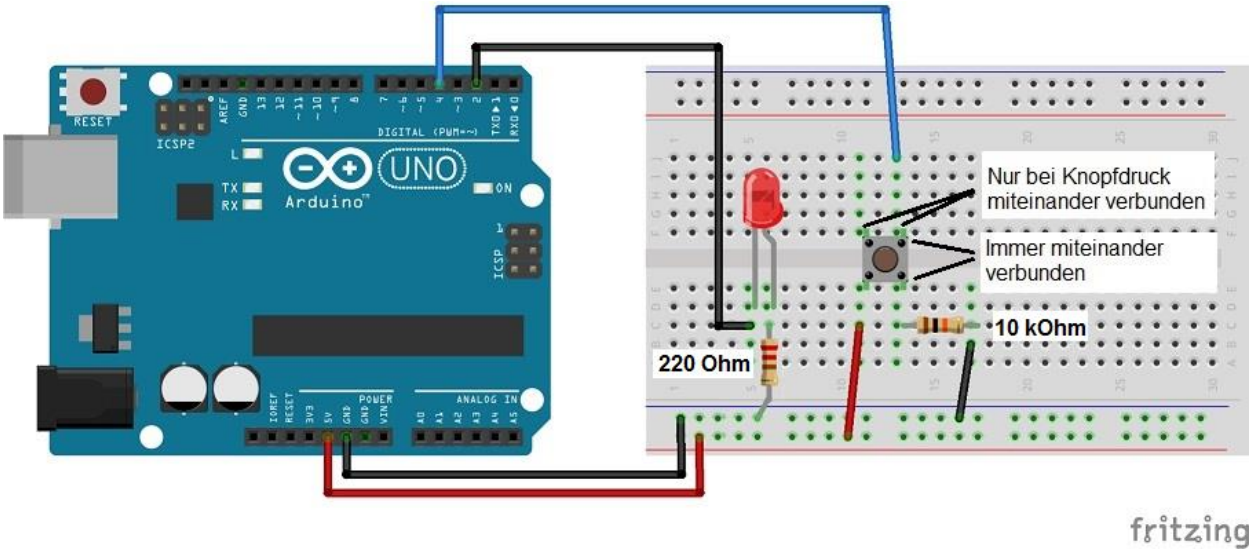
Ersetzt man in einem Spannungsteiler einen der beiden Widerstände durch einen Taster erhält man die folgenden beiden Zustände:

Taster geöffnet Ein geöffneter Taster entspricht einem **unendlich großen** Widerstand.	**Taster geschlossen** Ein geschlossener Taster hat **keinen** Widerstand.
 $U_{Port} = \frac{R_2}{R_1 + R_2} \cdot 5V$ $= \frac{10\ k\Omega}{\infty + 10\ k\Omega} \cdot 5V$ $= \frac{10\ k\Omega}{\infty} \cdot 5V = 0V$	 $U_{Port} = \frac{R_2}{R_1 + R_2} \cdot 5V$ $= \frac{10\ k\Omega}{0\ \Omega + 10\ k\Omega} \cdot 5V$ $= \frac{10\ k\Omega}{10\ k\Omega} \cdot 5V = 5V$

Das Schließen und Öffnen des Tasters liefert die Zustände HIGH (5V) und LOW (0V).

Bei offenem Taster ist der Port über den Widerstand mit GND verbunden. Der Widerstand wird deshalb als **„Pull-Down"-Widerstand** bezeichnet, weil seine Aufgabe darin besteht das Potenzial nach unten („down") zu ziehen („pull"), während der Taster nicht gedrückt wird. Wenn man in der Abbildung oben die Position von Taster und Festwiderstand vertauscht, dann fungiert der Festwiderstand als **„Pull-Up"-Widerstand**. Bei nicht gedrücktem Taster liegt am Port das Potenzial 5 V an.

<u>Aufgabe 21.1</u>
Baue die Schaltung wie in der Abbildung unten auf.
Schreibe mithilfe des `while`-Befehls (Tipp: `while(taster == HIGH)` …) ein Programm, mit dem die LED durch Betätigen des Tasters zum Leuchten gebracht wird.
Nach dem Loslassen des Tasters soll die LED wieder ausgehen.



Projekt 21.3 **„Reaktionszeittester“**

Baue und programmiere ein Reaktionszeitmessgerät:
Nachdem eine LED nach einer bestimmten Zeit zufällig (`random`-Befehl verwenden)aufleuchtet, soll ein Taster gedrückt und die Reaktionszeit zwischen Aufleuchten und Drücken bestimmt werden.

Programmtext:

```
int led = 2;
int taster = 4;

void setup() {
  pinMode(led,OUTPUT);
}

void loop() {
  int zeit = 0;     // Zeit wird auf Null gestellt
  int i = random(1000,10000);
  delay(i);
  digitalWrite(led,HIGH);
  int tasterStatus = digitalRead(taster);

  while(tasterStatus == LOW) { // Solange Schalter nicht gedrückt…
    zeit = zeit+1;
    delay(1);
    tasterStatus = digitalRead(taster);
  }
  digitalWrite(led,LOW);
  delay(5000);
}
```

a) Überleg dir, in welchem Programmabschnitt oben die Zeitmessung stattfindet.
Erweitere das Programm so, dass die Reaktionszeit auf dem seriellen Monitor angezeigt wird.
Profis: Erledige die Zeiterfassung mit Hilfe des millis-Befehls. Auf diese Weise kann man auf die Verwendung der while-Schleife verzichten.

b) Untersuche, ob ein Mensch schneller auf optische oder auf akustische Reize reagiert.
Verwende für den akustischen Reiz einen Lautsprecher, der zufällig einen kurzen Ton spielt.
Bestimme nun für jede Variante (sehen / hören) in jeweils 20 Einzelversuchen deine Reaktionszeit.
Bilde die Mittelwerte für beide Testreihen und vergleiche.

c) Profis: Der Arduino soll die 20 Durchgänge am Stück durchführen (`for(…,i<=20)`) und den Durchschnitt selbst berechnen (Verwende dazu eine Zählvariable `summe = summe + zeit` und am Ende `durchschnitt =summe/20`)

Aufgabe 21.3 **„Menschliche Stoppuhr“** ❀

Deine Aufgabe besteht darin, möglichst exakt eine Zeitspanne von 10 Sekunden abzuschätzen.
Schließe dazu einen Taster und das LCD an den Arduino an. Beim ersten Tastendruck soll die Zeitmessung starten, beim zweiten Tastendruck soll die Zeitmessung stoppen. Verwende den `millis`-Befehl.
Lass deine Zeit und die Abweichung von 10 s auf dem LCD anzeigen.

Aufgabe 21.4 ❀

Mit Hilfe des Poti soll auf dem LCD eine Zahl zwischen 1 und 100 eingestellt werden können.
Die gewählte Zahl soll mittels Tasterdruck „gespeichert“ werden.
Anschließend wird auf dieselbe Weise eine zweite Zahl ausgelesen und „gespeichert“
Beide Zahlen sollen multipliziert und das Ergebnis auf dem LCD ausgegeben werden.

22. if-Bedingung
Der Arduino reagiert auf seine Umwelt

Den Schirm aufmachen, wenn es regnet, Trinken, wenn man Durst hat – die Welt steckt voller Reaktionen auf Bedingungen. Hier lernst du, wie der Arduino auf Dinge von außen reagieren kann.

Bis jetzt hast du beides getrennt gelernt: Wie der Arduino Ausgänge steuern und wie er Sensorwerte auslesen kann. Nun soll beides miteinander verknüpft werden.

Eine Reaktion oder Entscheidung heißt in der Programmiersprache: „Bedingungsschleife“. Sie lautet allgemein: `if(Bedingung) {Reaktion}`

In Worten: Wenn die Bedingung in der runden Klammer erfüllt ist, werden die Befehle aus der geschweiften Klammer abgearbeitet. Das gilt auch umgekehrt: Wenn die Bedingung nicht erfüllt ist, werden die Befehle in der geschweiften Klammer ignoriert.

Mit der if – Bedingung kann man den Arduino dazu bringen, eine **Reaktion** auszuführen, **wenn** eine bestimmte **Bedingung** erfüllt ist, z.B soll er eine LED anschalten, wenn die Helligkeit der Umgebung abnimmt. Die **Bedingung** wäre hierbei also, dass die Helligkeit unter einen bestimmten Wert fällt, die **Reaktion** wäre, die LED anzuschalten.

Das Programm *„wenn die Helligkeit unter 400 ist, dann schalte die LED an, ansonsten schalte sie aus“* heißt in der Sprache des Arduino (ein LDR ist in einer Spannungsteilerschaltung an A0 angeschlossen):

```
void setup() {
 pinMode(led,OUTPUT);
}

void loop() {
 helligkeit = analogRead(A0);
 if (helligkeit < 400) {
   digitalWrite(led, HIGH);
 }
 else {
   digitalWrite(led, LOW);
 }
}
```

Erster Teil: *Bedingung (wenn / if):*
Zu Beginn wird geprüft, ob die Bedingung erfüllt ist. Hier ist dies der Fall, wenn der Befehl `analogRead(A0)` einen Wert kleiner als 400 liefert.

Zweiter Teil: *Bedingung erfüllt (dann / then):*
Ist die Bedingung erfüllt, dann führt der Arduino das Programm in der ersten geschwungenen Klammer aus. Hier wird z.B. eine LED angeschaltet.

Dritter Teil: *Bedingung nicht erfüllt (ansonsten / else):*
Ist die Bedingung nicht erfüllt, dann führt der Arduino das Programm in der zweiten geschwungenen Klammer aus. Vor diese wird der Befehl else geschrieben. Hier im Beispiel wird die LED ausgeschaltet.

Allgemein sieht eine if – Bedingung so aus:

```
if (Bedingung) {
  Reaktion 1;
}

else {
  Reaktion 2;
}
```

Bei der Abfrage der `if`-Bedingung werden die gleichen logischen Operatoren verwendet wie bei der while-Bedingung:

< kleiner	bzw.	> größer
<= kleiner oder gleich	bzw.	>= größer oder gleich
= = gleich	bzw.	! = ungleich

Beachte: Der vergleichende Operator besteht aus **zwei** Gleichheitszeichen!

Aufgabe 22.1
Vervollständige das Programm von der vorigen Seite und baue die zugehörige Schaltung auf.
Verwende für den LDR eine Spannungsteilerschaltung mit einem 1 kΩ-Festwiderstand.

Aufgabe 22.2
Erstelle den PAP zum Beispielprogramm auf der Vorderseite.

Aufgabe 22.3 „Helligkeitsanzeiger 1“
Baue und programmiere ein Gerät, das mithilfe von zwei LEDs anzeigt, ob im Raum das Licht an ist:

- Bei eingeschalteter Beleuchtung soll eine grüne LED leuchten.
- Bei ausgeschalteter Beleuchtung (*nicht mit dem Finger abdecken!*) soll eine rote LED leuchten.

Mit `else if(zweite Bedingung) {Reaktion 3}` lassen sich zusätzliche Bedingungen abfragen.

```
if (erste Bedingung) {
  Reaktion 1;
}
else if {zweite Bedingung) {
  Reaktion 2;
}
else {
  Reaktion 3;
}
```

Aufgabe 22.4 „Helligkeitsanzeiger 2“
Erweitere deinen „Helligkeitsanzeiger 1“ aus Aufgabe 22.2 mithilfe von `else if` wie folgt:

- Bei eingeschalteter Beleuchtung soll (wie bisher) eine grüne LED leuchten.
- Bei ausgeschalteter Beleuchtung soll (wie bisher) eine rote LED leuchten.
- Wenn man den LDR mit dem Finger abdeckt (neu!) soll eine blaue LED schnell blinken.

Über die Befehle „&&“ (bedeutet „und“) bzw. „||“ (bedeutet „oder“) lassen sich zwei Bedingungen miteinander verknüpfen.

Stell dir vor, man müsste menschliche Reaktionen nach dem Muster des Arduino programmieren und dein „Programm“ würde so lauten.

```
if (draußen ist es kalt && draußen regnet es) {Jacke anziehen}
```

Was würdest du tun, wenn es regnet? Was, wenn es kalt ist? Was, wenn es kalt ist **und** regnet?

Wie sähen die drei Fälle aus, wenn dein „Programm“ so lautet:

```
if (draußen ist es kalt || draußen regnet es) {Jacke anziehen}
```

Aufgabe 22.5 „Diebstahlschutz“ ❀
Baue und programmiere mit Hilfe eines LDR eine Diebstahlsicherung. Ein Alarmton soll ertönen, sobald ein Gegenstand von seiner Position entfernt wird. (Tipp: Funktioniert ähnlich wie Aufgabe 20.2)

Aufgabe 22.6

An Port 13 ist ein Drucktaster angeschlossen. Er liefert HIGH („1“), wenn er gedrückt ist, und LOW („0“), wenn er nicht betätigt wird.

Erkläre euch gegenseitig die vier Abschnitte des Programms.

```
int taster = 13;
int led = 8;

void setup() {
 pinMode(led, OUTPUT);
 pinMode(taster, INPUT);
}

void loop() {
 int entscheidung = digitalRead(taster);
 if (entscheidung == 1) {
   digitalWrite(led, HIGH);
 }
 else {
   digitalWrite(led, LOW);
 }
 delay(10);
}
```

1 2 3 4

Vermeidung der Variante mit `else`

Wenn bei einer `if`-Schleife mehrere Fälle unterschieden werden müssen, kann man manchmal auch auf die Verwendung von `else` verzichten.

Angenommen mittels `entscheidung = digitalRead(eingang);` wird ein Taster abgefragt.

Beispiel: mit `else`

```
if (entscheidung == 1) {
  digitalWrite(rot, HIGH);
}
else {
  digitalWrite(gruen, HIGH);
}
```

Beispiel: ohne `else`

```
if (entscheidung == 1)  {
  digitalWrite(rot, HIGH);
}
if (entscheidung == 0) {
  digitalWrite(gruen, HIGH);
}
```

Bedenke: Die Variante ohne `else` funktioniert, weil es außer „`entscheidung == 1`“ nur als einzige Alternative „`entscheidung == 0`“ gibt. Deshalb kann hier `else` durch ein zweites `if` ersetzt werden. Gäbe es statt den möglichen Werten „1“ und „0“ für `entscheidung` noch eine dritte Möglichkeit „2“, dann müsste im Beispiel ohne `else` im Programmtext eine drittes `if` erscheinen:

Beispiel: mit `else`

```
if (entscheidung == 1) {
  digitalWrite(rot, HIGH);
}
else if (entscheidung == 0) {
  digitalWrite(gruen, HIGH);
}
else {
  digitalWrite(blau, HIGH);
}
```

Beispiel: ohne `else`

```
if (entscheidung == 1)  {
  digitalWrite(rot, HIGH);
}
if (entscheidung == 0) {
  digitalWrite(gruen, HIGH);
}
if (entscheidung == 2) {
  digitalWrite(blau, HIGH);
}
```

Aufgabe 22.7

Für folgendes Programm ist an den Ports 1, 2 und 3 jeweils ein Taster und an den Ports 4, 5, 6 und 7 jeweils eine LED angeschlossen. Trage in untenstehender Tabelle ein, welche LED jeweils leuchtet.

```
void setup() {
  pinMode(1,INPUT);
  pinMode(2,INPUT);
  pinMode(3,INPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
}

void loop() {
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);

  if(digitalRead(1)==0 && digitalRead(2)==1) {
    if(digitalRead(3)==0) {
      digitalWrite(5,HIGH);
    }
    else {
      digitalWrite(6,HIGH);
    }
  }
  else if(digitalRead(3)==1) {
    digitalWrite(7,HIGH);
  }
  else {
    digitalWrite(4,HIGH);
  }
  delay(1000);
}
```

Port 1	Port 2	Port 3	welche LED leuchtet?
0	0	0	
0	0	1	
0	1	0	
1	0	0	
0	1	1	
1	0	1	
1	1	0	
1	1	1	

23. Fernbedienung

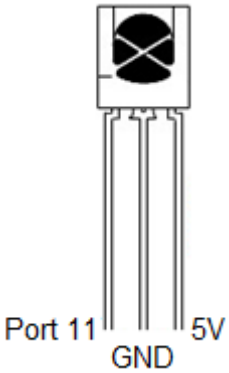
Viele elektronische Geräte (Fernseher, Stereoanlage, …) lassen sich elegant mit einer Fernbedienung steuern. Die meisten Fernbedienungen arbeiten mit einer Infrarot-LED (kurz: IR-LED) als Sender, die ein codiertes Signal an den Infrarot-Empfänger sendet, der sich im Elektrogerät selbst befindet.

Aufgabe 23.1

Drücke auf einige Tasten der Fernbedienung und betrachte die IR-LED auf der Vorderseite der Fernbedienung. Betrachte anschließend die IR-LED indirekt durch eine Handykamera. Was stellst du fest?

Das folgende Programm schaltet eine LED bei Tastendruck ein und aus:

```
#include <IRremote.h>

int ir = 11;       //  Anschluss des IR-Empfängers an Port 11
IRrecv irrecv(ir);
decode_results results;

void setup() {
  Serial.begin(9600);
  pinMode(2,OUTPUT);    // LED an Port 2
  irrecv.enableIRIn(); // Empfangprozess starten
}

void loop() {
  if (irrecv.decode(&results)) {    // Falls Daten ankommen…
    Serial.println(results.value);
    if (results.value == 16724175) {
      digitalWrite(2,HIGH);
    }
    if (results.value == 16718055) {
      digitalWrite(2,LOW);
    }
    irrecv.resume();          // Empfange die nächsten Daten…
  }
}
```



Aufgabe 23.2

a) Finde heraus, welche Tasten der IR-Fernbedienung im Programm die LED ein- bzw. ausschalten. Halte in einer Excel-Tabelle fest, welcher Wert bei welchem Tastendruck gesendet wird.

b) Untersuche, ob Sichtkontakt der IR-Fernbedienung zum zu steuernden Gerät erforderlich ist. Teste, ob ein Geodreieck / Fensterscheibe das IR-Licht reflektiert.

c) Verändere / Ergänze das Programm von oben folgendermaßen:

- Tasten 1 und 2: rote LED geht an bzw. aus
- Tasten 4 und 5: grüne LED geht an bzw. aus
- Tasten 7 und 8: gelbe LED geht an bzw. aus
- Taste 0: Alle LEDs gehen aus (Schnelle: Bei Dauertastendruck blinken alle LEDs)

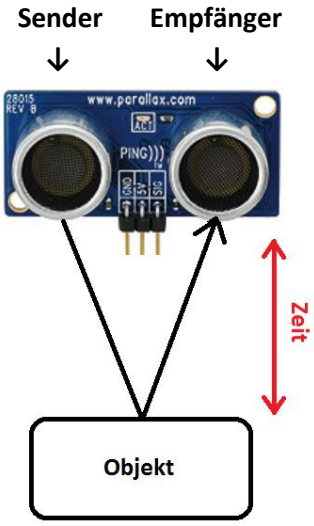
24. Ultraschall-Entfernungsmesser

Problemstellung:

Mit Hilfe des Arduino soll ein digitaler Entfernungsmesser aufgebaut werden. Die Entfernung soll auf dem seriellen Monitor angezeigt werden.

Zur Verfügung stehen dir:

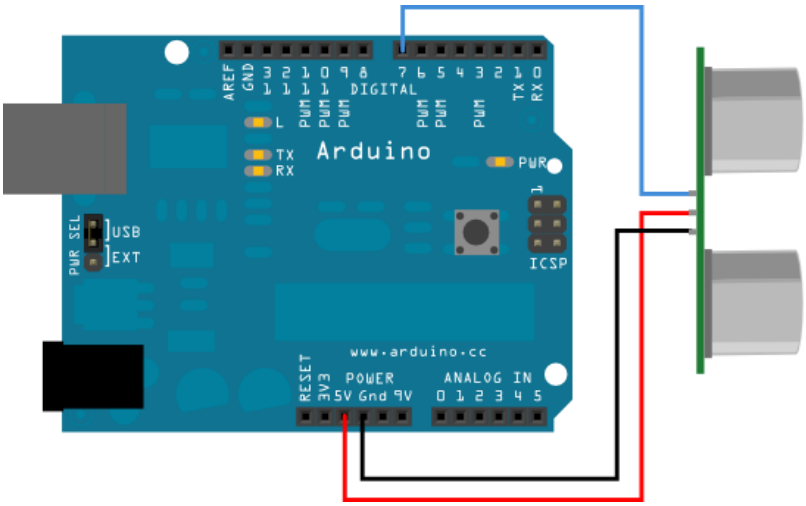
- Ein PING-Ultraschall-Sensor der Firma Parallax
- Meterstab



Der PING-Sensor besitzt drei Anschlüsse: Zwei für die Spannungsversorgung (5V und GND) und einen für das Signal (SIG).
Aufgabe des PING))) – Sensors (Bild rechts) ist es, den Abstand zu einem Gegenstand zu bestimmen. Das Funktionsprinzip dieses Sensors ist dabei einfach:

Nach dem Starten des Sensors schickt der **Sender** ein Ultraschall-Signal aus, welches am Objekt reflektiert wird und schließlich zum **Empfänger** gelangt. Beim Aussenden des Signales schaltet der SIG-Ausgang auf HIGH, wenn das Signal wieder ankommt, geht SIG wieder auf LOW. Wenn man nun weiß, wie schnell sich der Schall ausbreitet, kann die Distanz zum Objekt bestimmt werden!

Anschluss an den Arduino:



Aufgabe 24.1
Schließe den PING-Sensor an den Arduino an.
Übernimm den Programmcode auf der nächsten Seite per *copy and paste* und teste den PING-Sensor.

Vorsicht: Achte unbedingt darauf, dass du den Ultraschallsensor richtig anschließt. Ansonsten kann er kaputt gehen! Lass deine Schaltung im Zweifelsfall nochmals durch deinen Lehrer überprüfen.

Programmcode:

```
int ping = 7;
void setup() {
  Serial.begin(9600);
}

void loop() {
  pinMode(ping, OUTPUT);  // der PING sendet als AUSGANG einen 5ms HIGH-Impuls
  digitalWrite(ping, LOW); delayMicroseconds(5);
  digitalWrite(ping, HIGH); delayMicroseconds(5);
  digitalWrite(ping, LOW);
  pinMode(ping, INPUT);   // der PING misst als EINGANG die Zeit bis zum Echo

  int dauer = pulseIn(ping, HIGH); // Zeit bis Signal zurückkehrt in µs
  int entfernung = 0.0343*dauer/2; // Zeit in Entfernung umrechnen; s = v*t
                                   // Schallgeschw.: 343m/s = 0.0343cm/µs
  Serial.print(entfernung);
  Serial.println("cm");
  delay(1000);
}
```

Aufgabe 24.2

Untersuche folgende Dinge und halte deine Antworten schriftlich fest:

a) Wie groß ist der Messbereich des PING-Sensors (minimale und maximale Entfernung)?

b) Überprüfe mithilfe eines Lineals, ob der PING die Entfernungen korrekt bestimmt.

c) Funktioniert der Ultraschallsensor auch bei durchsichtigen Materialien?

d) Von welcher Stelle aus beginnt der PING-Sensor seine Messung (d.h. wo ist sein „Nullpunkt“?)

Aufgabe 24.3

Bestimme mit dem PING-Sensor deine Körpergröße. Wie groß ist die Abweichung?

Aufgabe 24.4 „Einparkassistent“ ❀

Baue und programmiere eine elektrische Einparkhilfe. Der Abstand zu einem Hindernis soll akustisch mitgeteilt werden: Je näher das Hindernis ist, desto schneller soll ein Piepton ertönen. Der Piepton soll dabei erst bei Abständen < 40 cm beginnen, bei Abständen < 5 cm soll ein Dauerton zu hören sein.

Aufgabe 24.5 „Blitzer“ ❀

Erweitere das Beispielprogramm von oben so, dass die Geschwindigkeit eines Objektes, das sich dem Ultraschallsensor nähert, bestimmt und auf dem seriellen Monitor angezeigt wird.

Falls das Fahrzeug z.B. schneller als 10 cm/s fährt, soll es „geblitzt“ (rote LED geht an) werden.

*Tipp: Um die Geschwindigkeit(**v = s/t**) zu bestimmen, muss man zweimal kurz hintereinander (z.B. im **Zeit**abstand von 100 ms) die Entfernung messen und damit die zurückgelegte **Strecke** bestimmen.*

Aufgabe 24.6 „Instrument“ ❀

Je nach Abstand deiner Hand zum PING-Sensor sollen verschieden hohe Töne gespielt werden. Spiele deine Lieblingslied. (Tipp: Verwende zur Umrechnung *Abstand* ↔ *Frequenz* den `map`–Befehl)

25. Motoren mit dem Arduino steuern

Wenn wir mit dem Arduino z.B. ein (selbstfahrendes) Fahrzeug bauen wollen, müssen wir für den Antrieb des Fahrzeugs einen oder mehrere Motoren ansteuern. So wie sich eine LED mit dem Befehl digitalWrite(…) an- und ausschalten lässt, kann man auch einen Motor ein- und ausschalten.

Achtung Verletzungsgefahr! Halte den (fest montierten!) rotierenden Propeller von Auge / Gesicht fern!!!

Aufgabe 25.1

Schreibe ein Programm, das den Motor 5 Sekunden einschaltet und dann für 2 Sekunden ausschaltet.

Aufgabe 25.2

✍ Das folgende Programm steuert einen Motor. Was passiert dabei genau? Kommentiere im Heft.

```
int motoranschluss1 = 2;
int motoranschluss2 = 7;

void setup() {
  pinMode(motoranschluss1, OUTPUT);     //
  pinMode(motoranschluss2, OUTPUT);     //
}
void loop() {
  digitalWrite(motoranschluss1, HIGH); //
  digitalWrite(motoranschluss2, LOW);  //
  delay(4000);
  digitalWrite(motoranschluss1, LOW);  //
  digitalWrite(motoranschluss2, HIGH); //
  delay(4000);
}
```

Aufgabe 25.3

Mit Hilfe der Pulsweitenmodulation (PWM) kann man eine LED heller und dunkler leuchten lassen. Mit dem gleichen Vorgehen lässt sich ein Motor schneller und langsamer drehen.

a) Finde heraus, ab welchem PWM-Wert sich der Motor zu drehen beginnt. Notiere den Wert.
b) Verändere das Programm aus Aufgabe 25.2 so, dass der Motor aus dem Stillstand immer schneller wird, bis er die maximale Drehzahl erreicht und anschließend wieder langsamer wird Sobald er zum Stehen kommt, soll er die Drehrichtung ändern.
   Sinnvoll ist hier, den Bereich auszublenden, bei dem der Motor noch nicht anläuft.

Aufgabe 25.4 ❀

Steuere die Drehzahl deines Motors mithilfe eines Potis. (Tipp: Verwende den `map`-Befehl)

26. Servo-Motor für genaue Drehwinkel

Im Modellbau und bei Robotern werden oft sog. **Servo-Motoren** verwendet. Diese werden mit PWM-Signalen gesteuert. Servos haben drei Anschlüsse.

rot = 5V
schwarz = GND
orange = Steuer-Signal

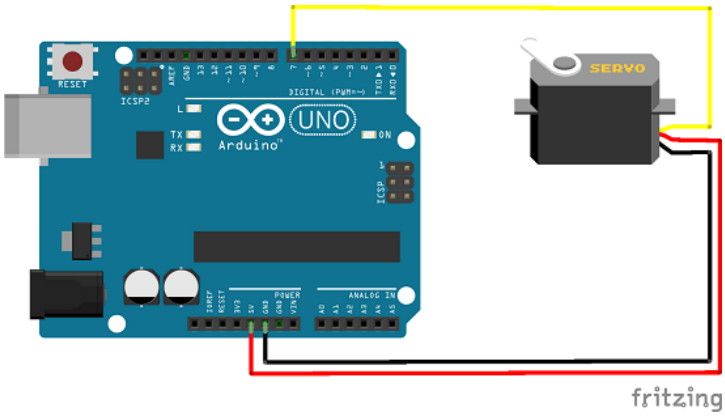
Es gibt zwei Arten von Servos:

- solche, die je nach PWM-Wert einen bestimmten Winkel drehen und stehen bleiben.

- solche, die sich je nach PWM-Wert schneller oder langsamer drehen (360°-Servos).

Hier lernst du, wie man mit einem Servo-Motor einen bestimmten Winkel zwischen 0° und 180° ansteuert. Die „Servo-Library“ macht dies sehr einfach:

```
#include <Servo.h>     // stellt die Befehle aus der „Servo-Library“ bereit
Servo meinservo;       // benennt den Servo-Motor als „meinservo“

void setup() {
  meinservo.attach(Port); } // als Anschluss-Ports sind nur PWM-Ports möglich

void loop() {
  meinservo.write(GRAD); }  // Steuerwert: 0 ≤ GRAD ≤ 180
```



Vorsicht: Achte unbedingt darauf, dass du den Servo-Motor richtig anschließt. Ansonsten kann er kaputt gehen! Lass deine Schaltung im Zweifelsfall nochmals durch deinen Lehrer überprüfen.

Aufgabe 26.1

Lass den Servo im Abstand von 2 Sekunden verschiedene zufällige (`random`-Befehl) Winkel anfahren.

Aufgabe 26.2

a) Steuere den Servo mithilfe eines Potis. (Tipp: Verwende den `map`-Befehl)

b) ❀ Baue mit einem Servo ein Zeigerinstrument, das auf einem Schirm die Helligkeit anzeigt.

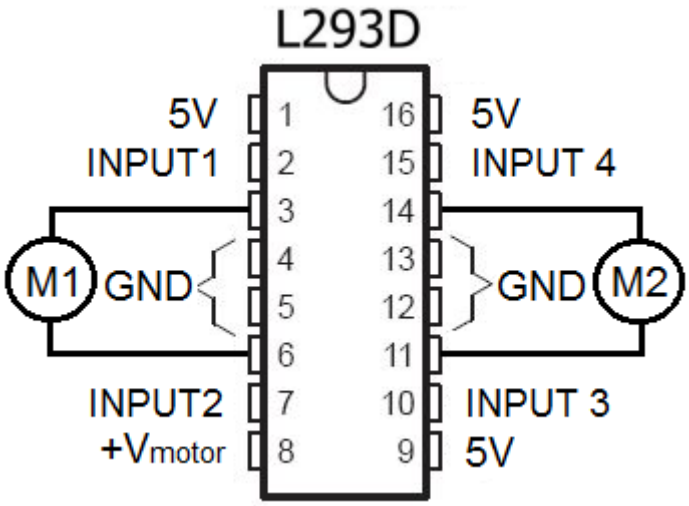
Projekt 26.3 „Radar“ ❀❀

Konstruiere eine Radaranlage, die auf dem LCD den Abstand zu Objekten in ihrer Umgebung anzeigt.

27. Motortreiber

Bislang haben wir einen Solarmotor und einen Servomotor zum Laufen gebracht. Diese Motoren laufen bei geringer Stromstärke. Der Arduino liefert allerdings eine Stromstärke von **maximal 40 mA** – für einen ordentlichen Motor, mit dem man beispielsweise einen Roboter bauen oder ein Fahrzeug antreiben kann, ist das viel zu wenig. Deshalb benutzt man einen sogenannten **Motortreiber**. Der Motortreiber, den wir benutzen, heißt **L293D**. An ihn können bis zu zwei Motoren angeschlossen werden, die sich jeweils vor- und rückwärts drehen lassen.

Funktion des L293D

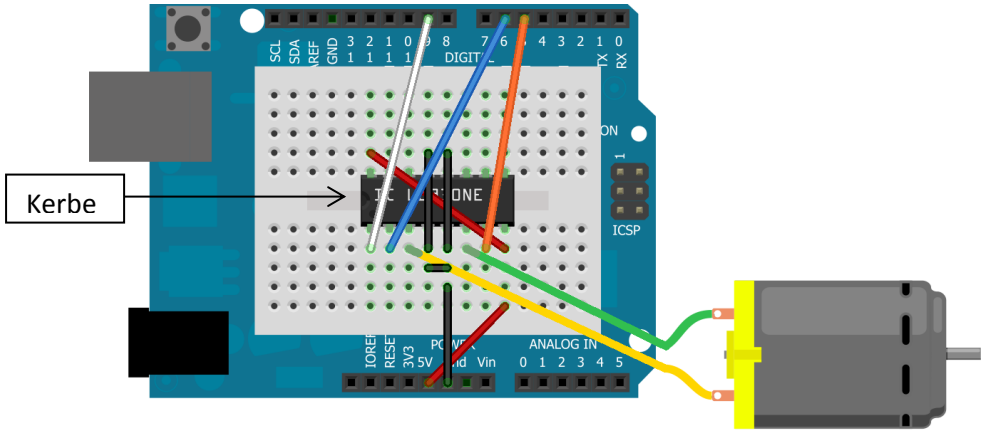


Pinbelegung L293D	Bedeutung
4, 5, 12, 13	Erdung (0V)
16	Energieversorgung des Motortreibers (5V)
1 und 9	Aktivierungs-Pin der Motoren. Ist dieser mit 5V verbunden, sind die Motor-Pins 3 und 6 bzw. 11 und 14 aktiv.
8	Energieversorgung der Motoren (5V)
3 und 6	Anschluss Motor 1
11 und 14	Anschluss Motor 2
2 und 7	Steuerung der Drehrichtung des Motors 1: **Pin2: HIGH Pin7: LOW → Motor dreht sich** Pin2: LOW Pin7: LOW → Motor dreht sich nicht **Pin2: LOW Pin7: HIGH → Motor dreht sich umgekehrt** Pin2: HIGH Pin7: HIGH → Motor dreht sich nicht
10 und 15	Steuerung der Drehrichtung des Motors 2 (analog zu den Pins 2 und 7 von Motor 1)

Aufgabe 27.1: Baue die Schaltung wie in der Abbildung (**siehe nächste Seite**) auf.

a) Schreibe ein Programm, das den Port zur Aktivierung des ICs (hier Port 9) auf HIGH setzt und einen der Ports zur Steuerung der Drehrichtung (entweder Port 5 oder 6) auf HIGH und den anderen auf LOW. Der Motor sollte sich jetzt in eine Richtung drehen.

b) Vertausche die Zustände der beiden Steuerungs-Ports (HIGH ↔ LOW).
Der Motor sollte sich jetzt in die andere Richtung drehen.



Aufgabe 27.2

a) Schreibe ein Programm, dass sich der Motor sekundenweise abwechselnd in die eine und die andere Richtung dreht.

b) Schreibe ein Programm, dass sich der Motor 3 mal hintereinander jeweils 2 Sekunden in eine Richtung und nach einer Pause von 1 Sekunde 2 Sekunden in die andere Richtung dreht.

c) Schließe einen zweiten Motor an den Motortreiber an.
Schreibe ein Programm, dass sich die Motoren zunächst in die gleiche Richtung, in die umgekehrte Richtung und anschließend in verschiedene Richtungen drehen. (Pause 1 Sekunde)
Erzeuge eine Ausgabe auf dem seriellen Monitor, die über die Drehrichtung der einzelnen Motoren informiert.

Aufgabe 27.3

a) Schreibe für das Vorwärts- und Rückwärtsdrehen sowie für das Stoppen des Motors Unterprogramme mit den Namen „vorwaerts“, „rueckwaerts“ und „stopp“.

b) Schreibe mithilfe der Unterprogramme ein Programm, das den Motor 1 Sekunde vorwärts, ½ Sekunde rückwärts laufen lässt und ihn dann für zwei Sekunden stoppt.

Aufgabe 27.4

Um die Geschwindigkeit des Motors zu steuern, kann man den Aktivierungs-Pin 1 (bzw. bei Verwendung von zwei Motoren zusätzlich Pin 9) des L293D an einen PWM-Ausgang des Arduino anschließen. Damit lässt sich die Leistung des L293D drosseln, indem man ihn z.B. nur 50% der Zeit aktiviert.

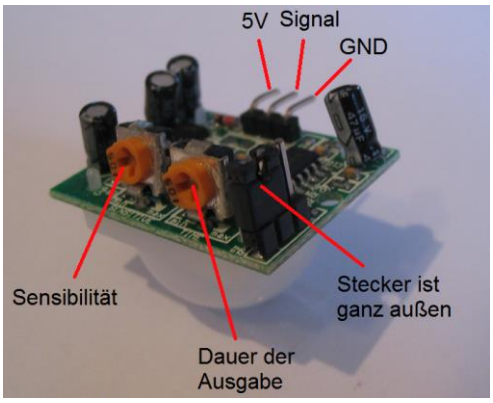
a) Lass deinen Motor langsamer als im bisherigen „Normalbetrieb“ laufen.

b) Lass deinen Motor aus dem Stand bis zur maximalen Drehzahl anfahren. Drossle anschließend die Drehzahl wieder bis zum Stillstand des Motors.

c) Regle die Geschwindigkeit des Motors mit Hilfe eines Potis.
Erzeuge eine Ausgabe auf dem seriellen Monitor, die die Leistung der Motoren in Prozent angibt.

28. Bewegungsmelder
Der Arduino als „Wachhund“

An vielen Häusern sind Bewegungsmelder installiert, die dafür sorgen, dass bei Dunkelheit das Außenlicht eingeschaltet wird, sobald sich jemand dem Hauseingang nähert. Im Inneren eines solchen Bewegungsmelders befindet sich ein sog. **PIR-Sensor** (**P**assiv-**I**nfrarot).



Der PIR-Sensor ist sehr einfach konstruiert: Sobald er eine Bewegung detektiert, gibt er auf dem mittleren Anschlusspin eine Spannung von 5 Volt aus. Das Ausgangssignal bleibt dabei so lange aktiv, wie vom Bewegungsmelder eine Bewegung detektiert wird. Dieses Signal muss nur ausgelesen und vom Arduino verarbeitet werden. Die Dauer des Ausgangssignals und die Sensibilität kann über zwei Drehregler (S: Sensibility; T: Time) eingestellt werden (siehe Abb.). Die Kunststofflinse ist nur leicht aufgesteckt. Wenn man sie abhebt kann man den Infrarotdetektor erkennen.

Funktionsweise

Jeder Mensch strahlt aufgrund seiner Körpertemperatur Infrarotstrahlung aus, die wir jedoch mit unserem Auge nicht wahrnehmen können. Basis eines PIR-Sensors sind Kristalle, die bei Änderung der einfallenden Strahlung (z.B. wenn sich eine Person nähert) eine elektrische Spannung im µV –Bereich abgeben. Dieses Ausgangssignal wird verstärkt und führt zu einem Wechsel der Ausgangsspannung von 0V auf 5V. Der Bewegungsmelder soll aber natürlich nicht bereits durch eine kalte Windböe ausgelöst werden. Dazu sind in jedem PIR-Sensor zwei Kristalle antiparallel geschaltet. Einer der Kristalle gibt bei Auftreffen von IR-Strahlung einen positiven, der andere einen negativen Spannungsimpuls ab. Änderungen der Strahlung, die gleichzeitig und mit gleicher Intensität (z.B. durch durch eine kühle Windböe) auf beide Kristalle einwirken, lösen so keinen Erfassungsvorgang aus. Anders verhält es sich bei schnellen Bewegungen. Bei schnellen punktuellen Änderungen der IR-Strahlung wechselt das Ausgangssignal von LOW auf HIGH.

Programmtext:

```
int pir = 2;

void setup() {
 pinMode(pir, INPUT); }

void loop() {
 int wert = digitalRead(pir);
 if(wert == HIGH) {
  Reaktion }
 else {
  andere Reaktion }
 delay(500);
}
```

Aufgabe 28.1

a) Schließe den PIR-Sensor an den Arduino an und lass dir die Werte von Port 2 auf dem seriellen Monitor anzeigen.

b) Experimentiere: Versuche dich dem PIR-Sensor so langsam zu nähern, dass er deine Bewegung nicht erkennt.

c) Überprüfe: Reagiert der PIR-Sensor auch auf die Bewegung von Nicht-Lebewesen (z.B. Bleistift)?

Aufgabe 28.2

Baue und programmiere mithilfe eines PIR-Sensors eine (akustische / optische) Alarmanlage.

29. Projekt – Fußgängerampel

Aufgabe: Baue und programmiere eine Verkehrsampel mit Fußgängerampel.

- Zunächst ist die Autoampel auf Grün. Wird ein Taster gedrückt, dann geht die Autoampel nach einiger Zeit auf Rot und die Fußgängerampel auf Grün.
- Während die Fußgängerampel grün ist, soll für blinde Menschen ein akustisches Signal zu hören sein.
- Kurz bevor die Fußgängerampel wieder auf Rot geht, soll ein kurzes Warnsignal erklingen.
- Auf dem LCD soll an der richtigen Stelle die Ausgabe „Gehen" bzw. „Stehen" erscheinen.
- Auf dem LCD soll für die Fußgänger die Restlänge ihrer Grünphase angezeigt werden.
- Verwende Unterprogramme, um das eigentliche Hauptprogramm übersichtlich zu gestalten.
- Speichere alles unter „Projekt_Fussgaengerampel_(Name)" ab.
- Zusatzfunktionen:
  - Mit Hilfe eines Drehpoti soll sich die Länge der Grünphase der Fußgängerampel einstellen lassen.
  - Es gibt – je nach Helligkeit (LDR!) – einen Tag- („normale" Ampel, siehe oben) und einen Nachtbetrieb (gelbes Dauerblinken)
- Erstelle mit der Fritzing-Software einen übersichtlichen(!) Schaltplan
- Gib die fertige Ampel, das kommentierte(!) Programm sowie einen <u>selbst gedrehten Videoclip</u> der laufenden Ampel (auf USB-Stick) ab.

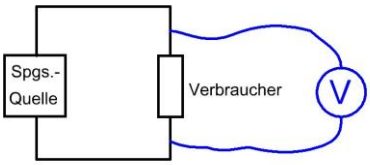
30. Anhang A
Grundlagen zum elektrischen Stromkreis

Ob im Handy, im Laptop, im Taschenrechner oder der elektrischen Zahnbürste – überall in diesen kleinen und großen Helfern des Alltags befinden sich elektrische Stromkreise. Da du bei der Arbeit mit dem Arduino auch sehr viel mit Stromkreisen in Berührung kommst, hier das Wichtigste in Kürze:

Es gibt vier physikalische Grundgrößen, die in Stromkreisen von Bedeutung sind: **Spannung**, **Strom(stärke)**, **Widerstand** und **elektrisches Potenzial**. Der el. Stromkreis ist mit einem Wasserstromkreis vergleichbar.

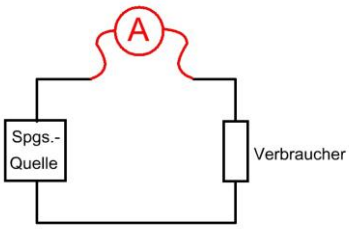
a) **Spannung**

Die Spannung ist vergleichbar mit dem Höhenunterschied eines Wasserfalls. Ohne Höhenunterschied kann kein Wasser fließen. Ohne elektrische Spannung kann kein elektrischer Strom fließen.
Eine Spannung liegt zwischen zwei Punkten an (sie fließt aber nicht).
Sie kann „durch Abgreifen" gemessen werden.
Ein Spannungsmesser ist immer parallel zu schalten.
Durch das Spannungsmessgerät selbst fließt kein Strom.
Eine Spannung U wird in der Einheit Volt (V) gemessen.



b) **Stromstärke**

Der elektrische Strom ist vergleichbar mit dem Wasserfluss eines Wasserfalls.
Ohne Höhenunterschied (Spannung) kann kein Wasser (Strom) fließen.
Elektrischer Strom kann nur fließen, wenn der Stromkreis von Plus (+) nach Minus (-) geschlossen ist. Ein Strommesser muss in den Stromkreis hinein geschalten werden, da der el. Strom nur gemessen werden kann, wenn er durch das Messgerät fließt. Deshalb ist ein Strommesser immer in Reihe zu schalten.
Die Stromstärke I wird in der Einheit Ampere (A) gemessen.



c) **Widerstand**

Der Widerstand ist bei einem Wasserfall vergleichbar mit mehr oder weniger Steinen im Flussbett.
Der Wasserfluss wird durch die Steine im Flussbett „gebremst".
Das Flussbett setzt dem Wasserfluss einen Widerstand entgegen. Ein metallischer Leiter (Kupfer, Eisen, usw.) setzt dem elektr. Strom ebenfalls einen Widerstand entgegen, der durch die elektr. Spannung überwunden werden muss. **Der Widerstand R wird in der Einheit Ohm (Ω) gemessen.**

d) **Potenzial**

Das elektrische Potenzial ist eine Art „elektrischer Druck" auf einem Kabel. Ganz ähnlich wie auch in einer Wasserleitung ein hoher oder tiefer Wasserdruck herrschen kann.
Verbindet man eine Leitung hohen Drucks mit einem Bereich, in dem ein tiefer Druck herrscht, dann fließt Wasser. Genauso beim Strom: Nur wenn es einen Potenzialunterschied gibt, fließt Strom. **Der Potenzialunterschied ist die bekannte Spannung**. Man kann einen elektrischen Anschluss erden, das bedeutet, man weist ihm das elektrische Potenzial 0 V zu.
<u>Achtung</u>: Weil sowohl das Potenzial als auch der Potenzialunterschied (=Spannung) die Einheit Volt haben, werden die Begriffe Spannung und Potenzial oft verwechselt!!!

Zusammenhang zwischen Spannung, Stromstärke und Widerstand (sog. Ohmsches Gesetz):

$$\boldsymbol{U = R \cdot I}$$

31. Anhang B
Kleiner Lötkurs

Ob im Alle Projekte in diesem Heft können mithilfe eines Steckbretts aufgebaut werden. Möchtest du anschließend eigene Projekte realisieren, musst du früher oder später löten. Löten ist die Verbindung von Metallen mit einem anderen, leichter schmelzbaren Metall.

Zum Löten benötigst du:

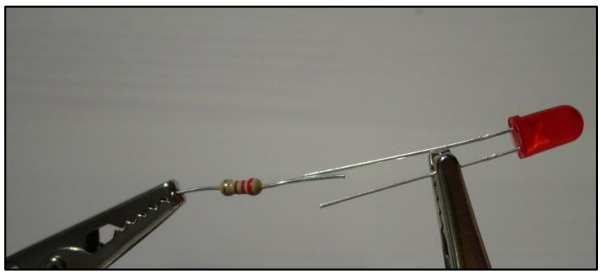
- einen Lötkolben / eine Lötstation mit Schwamm
- Lötzinn (bleifrei)
- eine unempfindliche Unterlage (z.B. Holzbrett)
- evtl. eine „dritte Hand“ (siehe Bild rechts)

Ablauf:

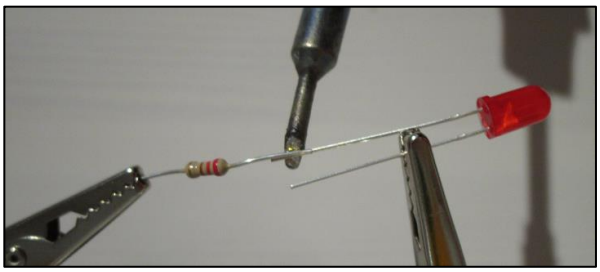
1. Bereite den Arbeitsbereich vor: Sorge dafür, dass alle Werkzeuge und Bauteile einfach erreichbar sind.
2. Heize den Lötkolben an (wenn du eine Temperaturanzeige hast, wähle 350 °C, auf keinen Fall mehr!).
3. Halte die zu verbindenen Drähte / Bauteile parallel nebeneinander. Eine „dritte Hand“ ist dabei ein nützliches Werkzeug. Klemme die Bauteile ein und richte sie passend aus, dann hast deine Hände frei.
4. Wenn der Lötkolben die notwendige Temperatur erreicht hat, verzinne seine Spitze mit einer winzigen Menge Lötzinn. Dadurch wird die Spitze geschützt und die Hitze verteilt sich besser auf die Bauteile
5. Halte die heiße Lötspitze nun an die Verbindungsstelle der beiden Drähte / Bauteile und erhitze diese Stelle. Achte darauf, dass du mit der Lötspitze beide Teile erhitzt. Das Erhitzen dauert ca. 5 Sekunden.
6. Berühre mit dem Lötzinn die erhitzte Verbindungsstelle: Das Lötzinn verflüssigt sich und verteilt sich automatisch im deren Umfeld. Dieser Vorgang sollte nicht länger als zwei Sekunden dauern.
7. Wenn sich das flüssige Lötzinn verteilt hat, nimm die Lötspitze schnell weg und lass die Verbindung abkühlen – fertig! Zum Schluss **Lötspitze reinigen**: Schwamm befeuchten und Lötspitze darauf abreiben

Löten funktioniert grundsätzlich so: Bauteile mit Lötspitze erhitzen → mit Lötzinn heiße Bauteile berühren

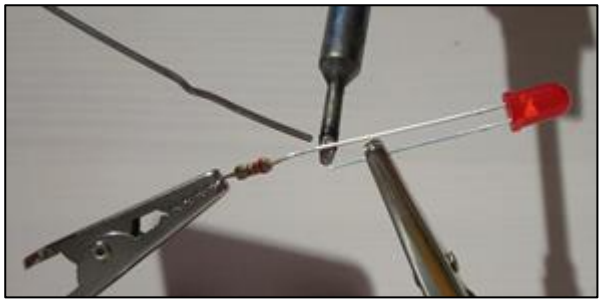
Löten geht auf <u>keinen Fall</u> so: Lötzinn auf Lötspitze verteilen → heißes Zinn auf kalten Bauteilen verteilen



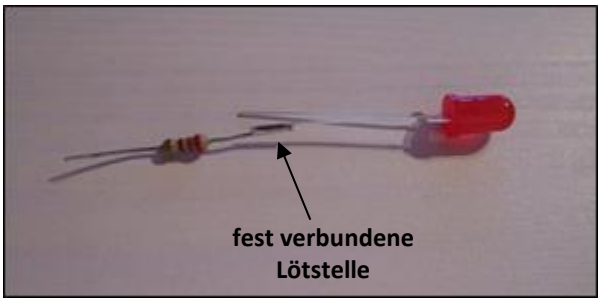
Zu verbindende Bauteile parallel halten



Bauteile mit heißer Lötspitze erhitzen



Mit Lötzinn heiße Verbindungsstelle berühren



Verbindungsstelle abkühlen lassen – fertig!

Lötübung: Widerstandsmännchen

Stückliste

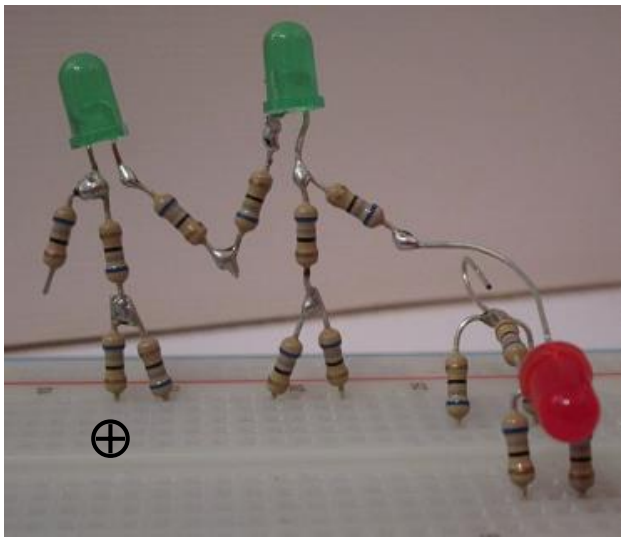
Anzahl	Bezeichnung
15	Widerstand
2	LED rot (Vorwärtsspannung ca. 2 V)
1	LED grün (Vorwärtsspannung ca. 3 V)
1	Lochrasterplatine (ca. 50 mm x 40 mm)
1	Batteriehalter 9V

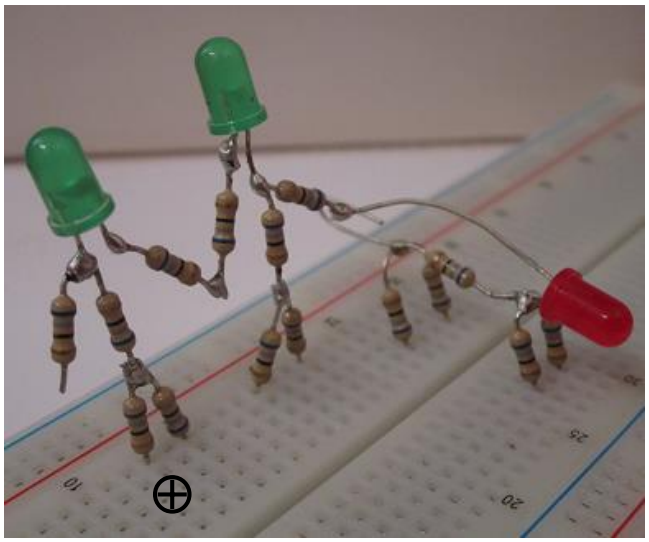
Vorüberlegung

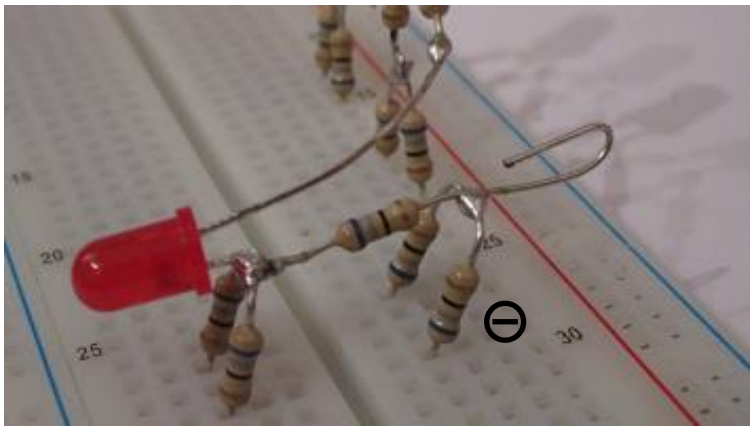
Berechne die Größe der benötigten Widerstände (die Abbildungen unten helfen dir hierbei)

Aufgabe

Löte die Bauteile entsprechend der Abbildung zusammen. Tipp: Beginne mit dem Hund und löte anschließend ein Männchen fertig. Bei genügend Zeit kannst du das zweite Männchen ergänzen. Achte auf die richtige Orientierung der LEDs (jeweils langes und kurzes Beinchen zweier benachbarter LEDs verbinden) und auf das korrekte Anlöten des Batteriehalter (+ der Batterie an langes Beinchen, – der Batterie an kurzes Beinchen)





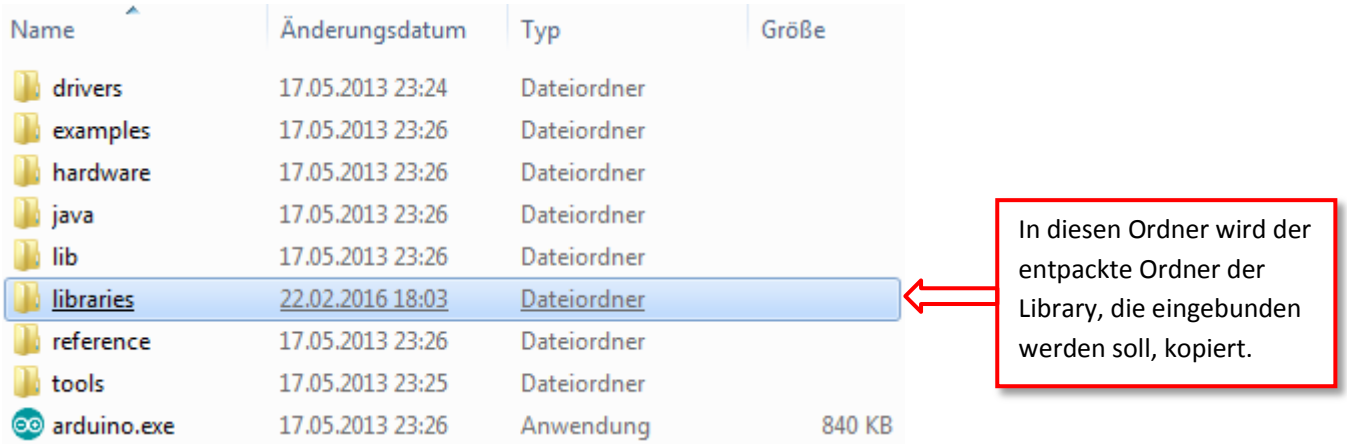


32. Anhang C
Einbinden einer Bibliothek

Bibliotheken (sog. libraries) erweitern den Funktionsumfang der Arduino-Software um weitere Befehle. Es gibt Bibliotheken für LCDs, für Servos und viele weitere Bauteile. Will man sie verwenden, müssen sie in das Programm eingefügt werden. Die Arduino-Software hat bereits viele verschiedene Bibliotheken „an Bord“. Im Hauptmenü findet man unter Sketch den Befehl *Library importieren*. Hier wählt man einfach die Bibliothek aus, die man verwenden will und im Programm erscheint die Include-Zeile, z.B.:

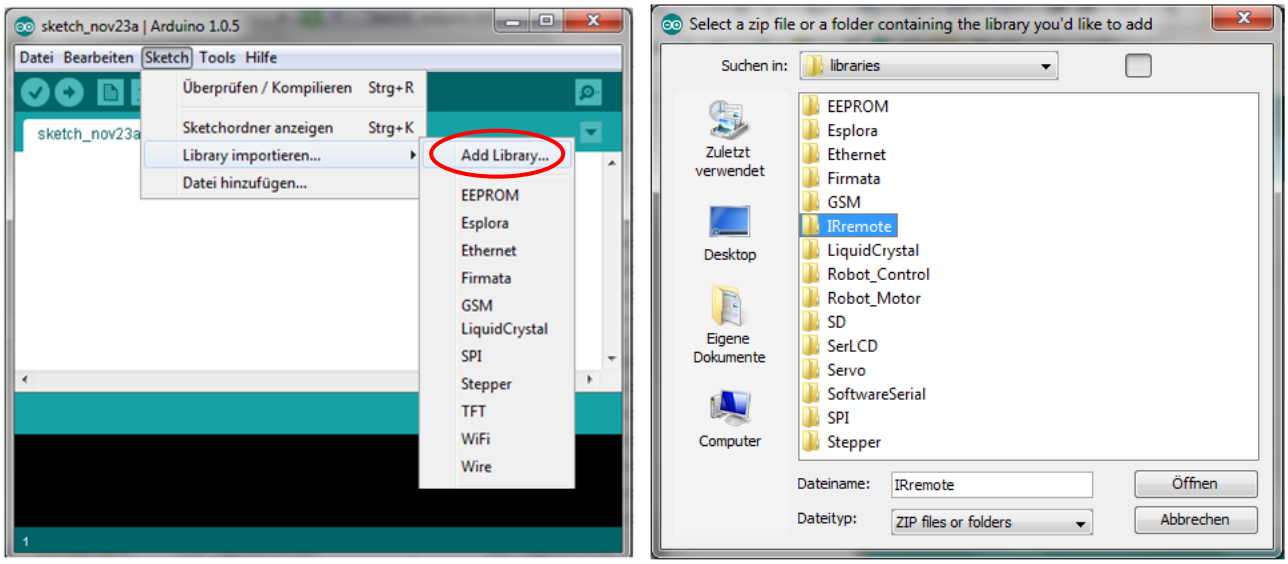
`#include <IRremote.h>` (Bibliothek mit Befehlen zur Verwendung einer Fernbedingung)

Man kann auch eine neue Library hinzufügen, die man z.B. aus dem Internet herunter geladen hat. Der entpackte Ordner der Library muss in den libraries-Ordner im Arduino-Ordner kopiert werden.

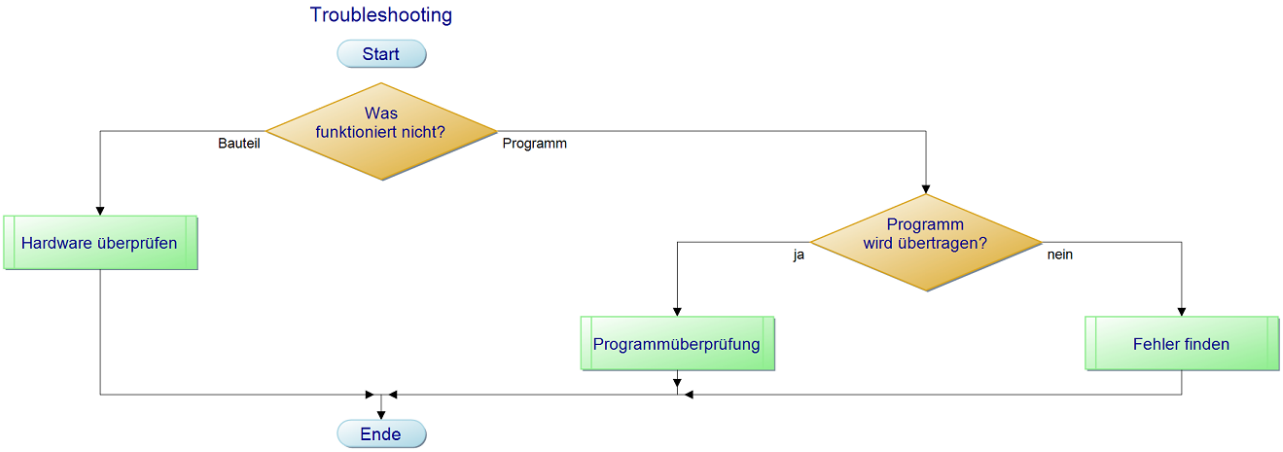


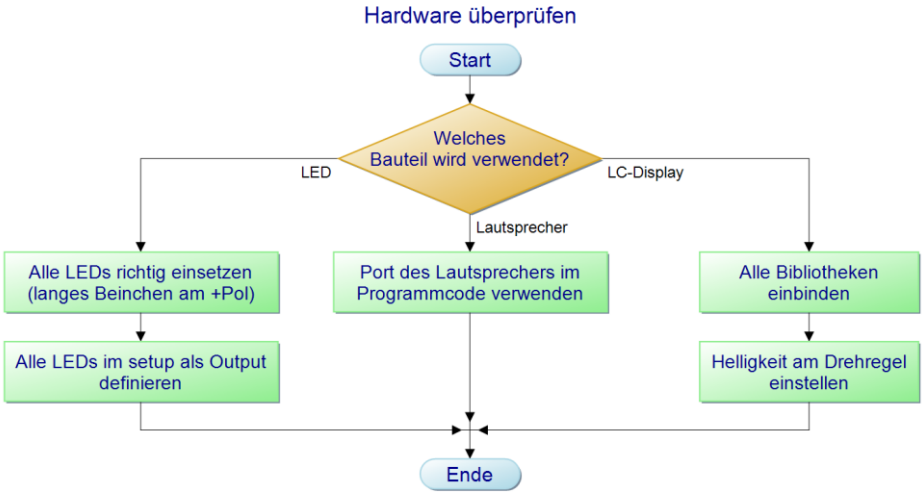
Nach dem Öffnen der Arduino-Software, wird die neue Library folgendermaßen eingebunden:
Sketch → Library importieren → Add Library → gewünschten Ordner einmal anlicken → Öffnen

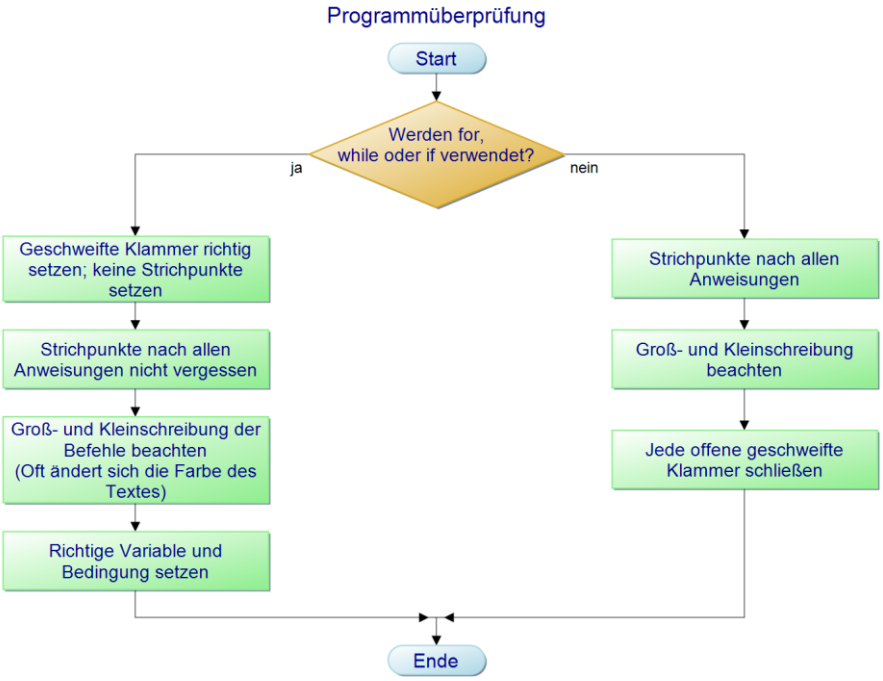
Anschließend kann die Library mittels *Sketch → Library importieren* eingebunden werden.



33. Anhang D
Trouble-Shooting



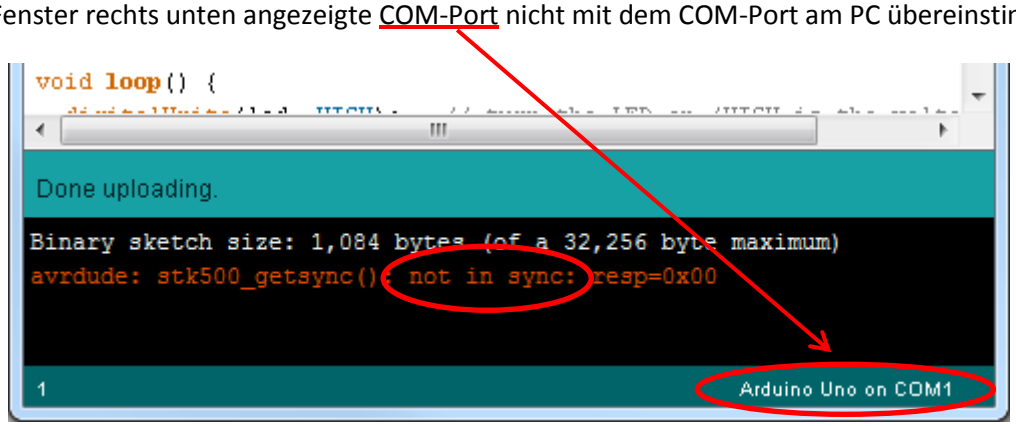




Fehlermeldung „not in sync“

Manchmal taucht beim Versuch, ein Programm auf den Arduino zu übertragen, die Fehlermeldung „not in sync“ (vgl. Abb. 1) auf. Sie signalisiert, dass die Übertragung nicht funktioniert hat. Meist liegt das daran, dass der im Fenster rechts unten angezeigte COM-Port nicht mit dem COM-Port am PC übereinstimmt.

Abbildung 1



Problembehebung:

1) USB-Kabel des Arduino aus- und wieder einstecken. **Manchmal genügt das bereits**!

   Falls der Fehler jedoch beim nächsten Übertragungsversuch weiterhin angezeigt wird:
2) Programmierumgebung schließen (Programmcode vorher abspeichern).
3) USB-Kabel des Arduino vom Computer trennen.
4) Programmierumgebung neu starten (arduino.exe ausführen).
5) Erst nachdem das Programm fertig geladen ist, USB-Kabel wieder mit dem Computer verbinden.
6) Gegebenenfalls in der Menüzeile unter TOOLS → Serieller Port → den aktuellen Port einstellen (im Gerätemanager (vgl. Abb.2) nachschauen). Hinweis: Der Gerätemanager kann bequem mit der Tastenkombination (Windows-Taste + Pause Taste) geöffnet werden. (vgl. Abb.3)

Abbildung 2:

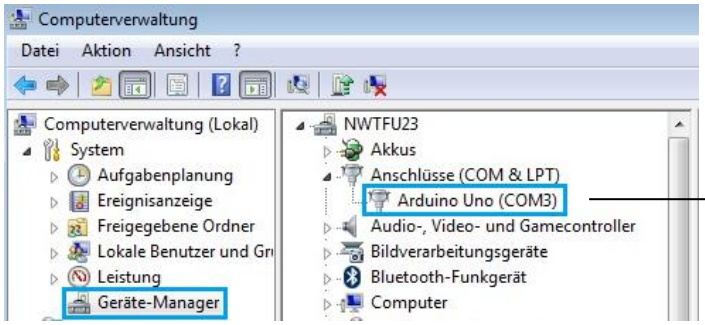
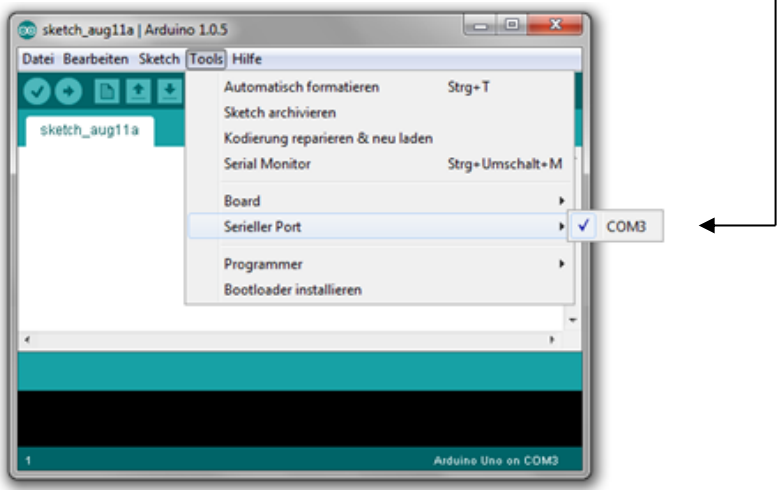
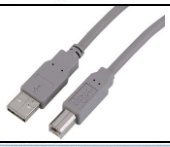


Abbildung 3:



34. Anhang E
Verwendete Bauteile

Grundausstattung

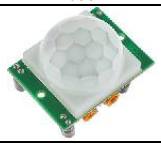
Bauteil	Abbildung	Funktion	Preis (Firma)
Arduino Uno		Mikrocontroller	ca. 20 – 25 € (Elmicro/Watterott)
USB-Kabel (0,5 – 1 m) (A-Stecker, B-Stecker)		Anschlusskabel	1 € (Reichelt)
Breadboard		Steckbrett	3 – 5 € (Reichelt)
LED 3 mm (gelb, grün, rot)		Leuchtdioden (LED)	0,08 – 0,20 € (Conrad)
Kohleschichtwiderstände: 100Ω, 220Ω, 1 kΩ, 10 kΩ, 100 kΩ		elektr. Festwiderstand	0,10 – 0,20 € (Conrad)
Flexible Drahtbrücken (male – male)		Verbindungskabel	3 € / 10 Stk. (Reichelt)
		Kosten Grund-Set	ca. 35 – 40 €

Aktoren

SC 52-11 RT (13,5 mm)		7-Segment-Anzeige (gemeinsame Kathode)	0,60 € (Reichelt)
RGB 3-Farben-LED-Modul		RGB-LED-Modul	1 € (miniinthebox)
BL 45		Lautsprecher	1 € (Reichelt)
RK09K111-LIN10K		Potentiometer (10 kΩ)	1 € (Reichelt)
RF 300 (0,4 – 5,9 V)		Solarmotor mit Propeller	3 € (Solexpert)
Modelcraft RS 2 JR		Servomotor	6 € (Conrad)

16x2 LCD (serielle Ansteuerung, 3 Anschlusskabel)		Display	25 € (Watterott)
I2C-LCD oder HD44780 (nicht seriell) mit I2C-Modul		Display	4 € (Amazon)

Sensoren

LDR		Fotowiderstand	2 € (Reichelt)
LDTR-34162		Drucktaster	0,50 € (miniinthebox)
NTC-0,2 10K		NTC-Temperatursensor	0,40 € (Reichelt)
Parallax PING)))		Ultraschallsensor	25 € (Watterott)
PIR-Sensor		Bewegungsmelder	3 – 7 € (Watterott)
CNY 70		Reflexlichtschranke	1 € (Reichelt)

Sonstiges Zubehör

Breadboard Shield		Aufsatz mit Steckplatine	2 – 5 € (Amazon, Ebay)
IR Remote Control		Fernbedienung	2 – 5 € (Amazon, Ebay)
MAX7119 8x8		LED Matrix Modul	4 € (Amazon)
BOB-12758		Mikrofon	6,50 € (SparkFun)
		Kosten Komplett-Set	ca. 100 – 125 €