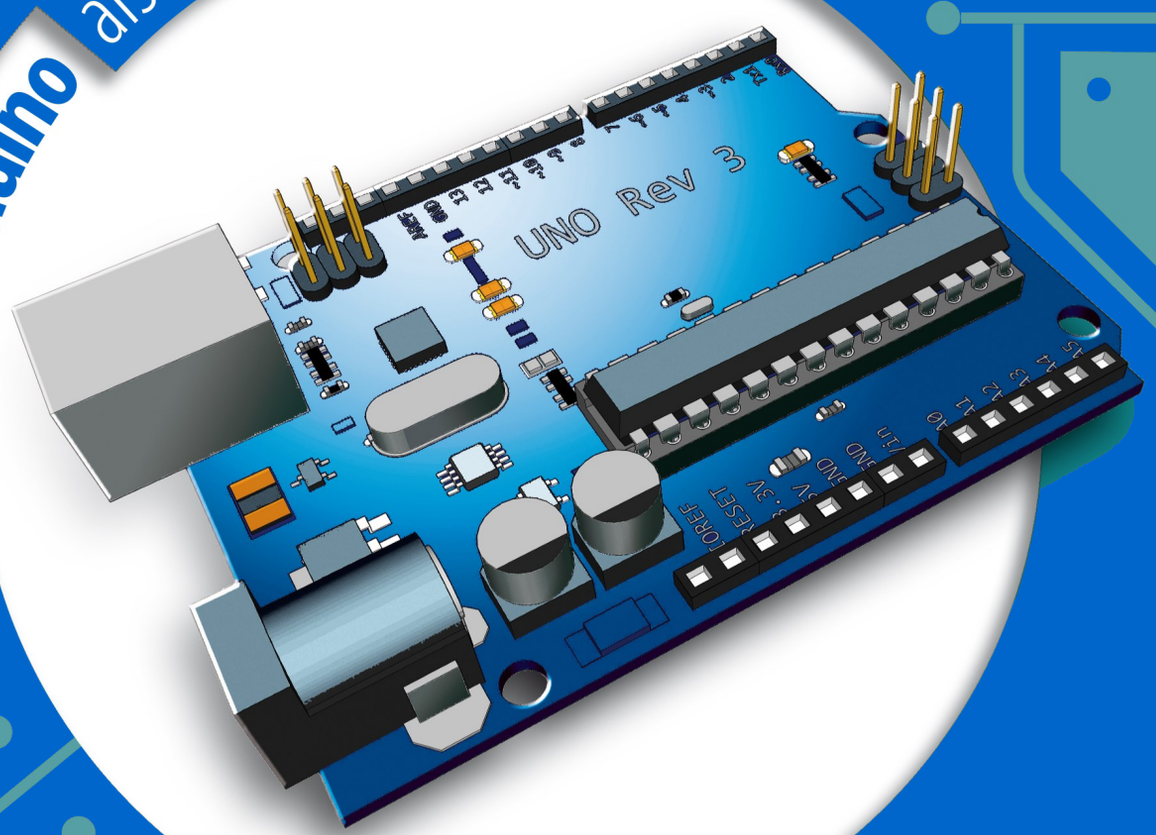
Messen und Steuern mit dem Mikrocontroller

Der Arduino als Steuerzentrale



Version 1.0

Sehr geehrte NwT-Kollegin,
sehr geehrter NwT-Kollege,

Dieses Heft ist die Fortsetzung des LernBausteins „Einführung in den Mikrocontroller Arduino“ (kurz Arduino1) und baut auf diesem auf.

Die grundlegenden Befehle und Strukturen werden hier nochmals aufgegriffen und durch das Erfassen und Verarbeiten analoger Werte erweitert.

Dieser LernBaustein richtet sich an Schülerinnen und Schüler ab Klasse 9, da diese im Physikunterricht schon Erfahrung im Umgang mit der Größe elektrische Spannung und deren Messung gemacht haben. Dieses Wissen wird aufgegriffen, und es wird gezeigt, wie man mit dem Arduino Spannungswerte erfassen und verarbeiten kann. Schnell wird klar, dass der Mikrocontroller ein vielseitiges Werkzeug zum Auslesen selbstentwickelter Sensoren ist und unverzichtbar zur Steuerung von Prozessen.

Wir wünschen Ihnen und Ihren Schülern viel Spaß mit dem Heft und beim Umsetzen eigener Ideen.

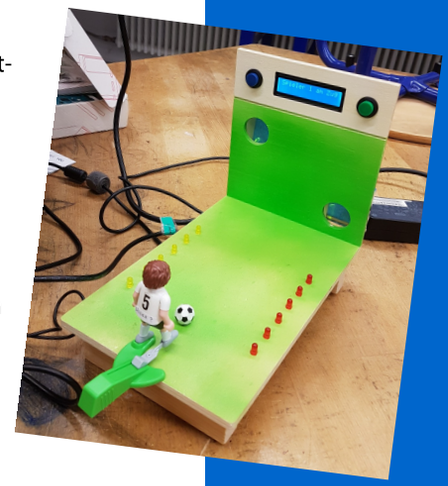
Materialliste für 24 Schüler

12	Arduino/Genuino Uno Rev 3 oder kompatible
12	USB-Kabel AB
12	Breadboard (Steckbrett)
12	Servomotor SG90
12	Lautsprecher, 1-2 W, 4-8 Ohm
12	LC-Display mit I2C-Modul
240	Jump-Wires male-female
240	Jump-Wires male-male
100	Taster
200	LED, gelb
200	LED, grün
200	LED, rot
12	Drehpotentiometer 10kΩ
12	LDR
50	Widerstand 1/4W, 10k Ohm
200	Widerstand 1/4W, 220 Ohm
	Optional
5	Piezo-Vibrations-Sensor
5	TMP36
5	LM35
2	Sharp-Infrarot-Abstandsensor
5	PCF 8574 Remote Modul
5	DHT 11 Temperatursensor
5	Batteriehalter 2 AAA

Inhaltsverzeichnis

Was gibt's **Neues**?

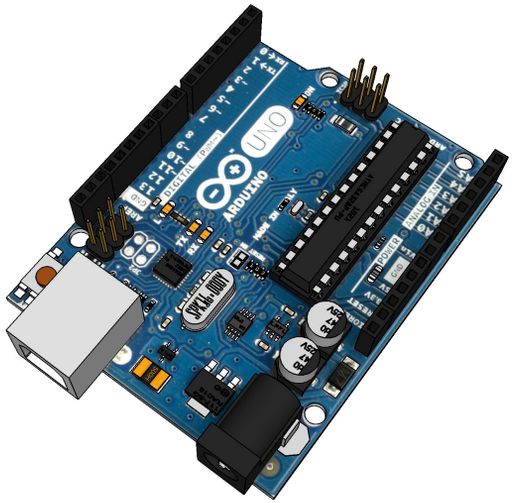
Im vergangenen Schuljahr hast du bestimmt schon erfahren, dass man mit dem Mikrocontroller als Steuerzentrale tolle Projekte verwirklichen kann. Wir verwenden hier einen der erfolgreichsten Microcontroller für Maker—den Arduino. Er ist so erfolgreich, dass sein Herzstück, der Atmel-Chip, inzwischen fast überall dort vorkommt wo Dinge automatisiert sind.



Du hast gelernt, wie man mit Problemen umgeht, hast dich (und vielleicht euch deinen Lehrer;)) lange mit Fehlersuche beschäftigt und warst bestimt auch mal ziemlich frustriert. Aber das passiert auch erfahrenen Programmierern und ist völlig normal.

Du hast dich bei deiner Arbeit hauptsächlich mit der Ansteuerung von verschiedenen Bauteilen beschäftigt. In diesem Heft lernst du, wie man mit Hilfe des Arduino messen und Prozesse mit Hilfe von Sensoren steuern kann. Dazu verwendest du die analogen Eingänge des Arduinos. Außerdem lernst du noch einige interessante „Programmiertricks" wie den Umgang mit Listen oder die Verwendung von Interrupts. Darüberhinaus gibt es auch noch andere neue Dinge zu entdecken, die für deine Projekte sehr nützlich sein werden.

Zunächst aber wirst du im ersten Kapitel du die wichtigsten Anweisungen und Programmstrukturen wiederholen. Außerdem erstellst du Programmablaufpläne oder setzt sie um.



Wir beginnen mit ein paar Aufgaben, die du mit Hilfe deiner Kenntnisse des ersten Heftes lösen kannst.

Los geht's !

Lesehinweis

In diesem Heft sind Aufgaben mit verschiedenen inhaltlichen Bedeutungen enthalten:

5.3 Diese Aufgabe beinhaltet die Einführung neuer Inhalte, die für das weitere Verständnis wesentlich sind.

5.3 Diese Aufgabe stellt einen Exkurs oder eine interessante Vertiefung dar und ist für den Fortgang im Heft optional.

Bild von Drohne: CC0, https://pixabay.com/illustrations/quadrocopter-camera-drone-fly-1658967/

Das Skript wurde, wie auch schon Band 1, als Selbstlernskript konzipiert. Es ist sinnvoll, bei den Wiederholungsaufgaben eine Besprechung mit der ganzen Klasse zu machen. So kann auch leichter abgeschätzt werden, wer wieviel Unterstützungsbedarf hat.

Um selbst Programmablaufpläne zum Ausdrucken am Rechner zu erstellen, gibt es handliche Programme, z.B. den PAP-Designer, den man unter dieser Adresse zum Download erhält:

https://www.heise.de/download/product/papdesigner-51889

Die Zufallsfunktion des Arduino funktioniert am besten im *loop*, im *setup* funktionieren nicht alle Zahlenmengen.

Lösung zu 1.2:

```
int ledRot=12;
….
pinMode(ledRot,OUTPUT);
….
if (augenZahl==1){
 digitalWrite(ledRot,HIGH);
 Serial.println(„Buuh!“);
 }
 else {
 digitalWrite(ledRot,LOW);
 }
….
```

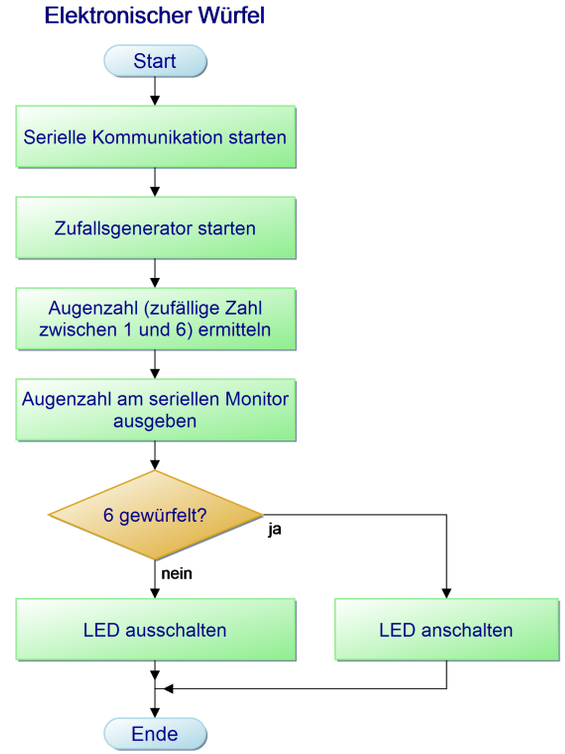
Aus Platzgründen sind bei den Lösungen häufig nur die wichtigsten Programmteile genannt (z.B. werden leere *void loop* und *pinMode* oft weggelassen).

https://pixabay.com/de/vectors/w%C3%BCrfel-rot-zwei-spiel-walzen-25637/

Achte bei der LED auf die Polung:

1 Kurze Wiederholung

Hier in der Abbildung siehst du einen Programmablaufplan (PAP) für einen elektronischen Würfel. Daneben findest du die „Übersetzung“ in den Arduino-Code.



elektronischer_Wuerfel

```
int augenZahl;
int led=13;

void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(A0));
  augenZahl=random(1,7);
  Serial.print(„Augenzahl:“);
  Serial.println(augenZahl);
  if (augenZahl==6){
  digitalWrite(led,HIGH);
  }
  else {
  digitalWrite(led,LOW);
  }

}
void loop() {}
```

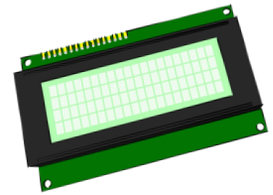
1.1 Beschreibe in eigenen Worten, was das Programm macht.

Tippe das Programm ab und übertrage es auf deinen Arduino. Liegst du mit deiner Vermutung zum Programmablauf richtig? Kommentiere den Programmcode.

1.2 Ergänze das Programm so, dass bei einer 1 eine rote LED leuchtet und „Buuh!“ auf dem seriellen Monitor angezeigt

1.3 Kannst du die Augenzahl auch auf einem LC-Display anzeigen lassen?

Tipp: Helfen kann dir hier das erste Arduino-Heft, oder ein Beispiel-Code. Diesen findest du in der Arduino-IDE unter DATEI > BEISPIELE > Beispiele aus eigenen Bibliotheken > LiquidCrystal_I2C

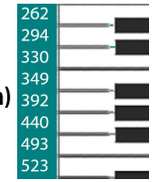


Hier ist ein Code für Musikliebhaber, mit zwei Tastern können zwei unterschiedliche Töne gespielt werden. Hierzu sind ein Lautsprecher an Pin 11 und zwei Taster an den Pins 3 und 4 angeschlossen.

1.4 Tippe das Programm ab, baue die zugehörige Schaltung auf und teste dein Programm. Kommentiere den Code und experimentiere mit verschiedenen Tonintervallen.

1.5 Noch schöner wäre ein Dreiklang. Erstelle ein Programm, das mit den beiden Tastern drei verschiedene Töne spielen kann. (Tipp: Wie man Bedingungen miteinander verknüpft erfährst in der Randspalte)

1.6 **Das Nerd-O-Fon: Schaffst du mit 3 Tastern eine Oktave? (Keine Taste gedrückt muss auch einen Ton geben)**



Mini-Klavier

```
int taster1 = 3;
int taster2 = 4;
int status1;
int status2;
int lsPin = 11;

void setup() {
  pinMode(taster1, INPUT_PULLUP);
  pinMode(taster2, INPUT_PULLUP);
  pinMode(lsPin, OUTPUT);
}

void loop() {
  status1 = digitalRead(taster1);
  status2 = digitalRead(taster2);
  if (status1 == 0) {
    tone(lsPin, 262);
  }
  if (status2 == 0) {
    tone(lsPin, 330);
  }
}
```

Dieser Code hier stellt einen Sonnenaufgang (bzw. ganz viele Sonnenaufgänge) dar: Eine LED wird langsam immer heller. Hierfür wird eine Zählschleife (eine sog. FOR-Schleife) verwendet.

Sonnenaufgang

```
int ledRot = 9;

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

void loop() {
  for (int i=0; i<255; i=i+1){
    analogWrite(ledRot, i);
    delay(50);
  }
  delay(2000);
}
```

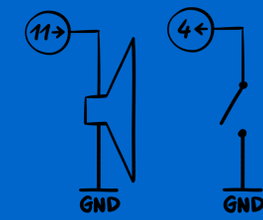
1.7 Tippe auch hier den Code ab, baue die Schaltung auf und teste dein Programm.
Kommentiere wieder den Code, sodass man versteht, was die einzelnen Programmzeilen machen. Achte auch darauf, was die Parameter der FOR-Schleife bedeuten.
Verändere jetzt dein Programm nach Belieben (erst eine rote, dann eine gelbe LED hochdimmen, ein Sonnenaufgang, der genau 30 Sekunden dauert etc.)

1.8 Diesmal soll der Sonnenaufgang auf Knopfdruck gestartet werden – hier benutzen wir eine WHILE-Schleife, also eine Schleife, die so lange durchlaufen wird, wie die Bedingung der Schleife erfüllt wird.

1.9 Schaffst du es, dass auf Knopfdruck die LED hochdimmt und bei einem weiteren Knopfdruck die LED wieder herunterdimmt?

5

So schließt du Taster und Lautsprecher an:



Du kennst schon die IF-Verzweigung: Hier wird der zugehörige Code ausgeführt, falls die Bedingung, die in der Klammer steht, erfüllt ist. Man kann aber auch zwei Bedingungen logisch miteinander verknüpfen:

if (x == 1 && y == 1) Anweisung

sorgt dafür, dass die Anweisung nur dann ausgeführt wird, wenn x **und** y beide gleich 1 sind (AND)

if (x == 1 || y == 1) Anweisung

sorgt dafür, dass die Anweisung nur dann ausgeführt wird, wenn x **oder** y **oder beide** gleich 1 sind (OR-Veknüpfung). Die senkrechten Striche erzeugt man mit AltGr+< oder Alt+7 bei MacOS.

Tipp:
Die Codezeile
while (digitalRead(3)==1) { }
Sorgt dafür, dass das Programm an dieser Stelle bleibt und nichts macht, solange bis ein Taster an Pin 3 gedrückt wird.

Lösung zu 1.5:
Es genügt, das bestehende Programm so zu ergänzen, dass eine weiterer Ton gespielt wird, wenn beide Taster zugleich gedrückt werden:

```
if (status1==0 && status2==0){
  tone (lsPin, 392);
}
```

Lösung zu 1.6:
Am Besten notiert man sich zunächst , welche Tasterkombination welchen Ton spielen soll:

110=c, 101=d, 100=e usw.

Dann kann man im Code die ensprechenden Bedingungen verknüpfen:

```
if (status1==1 && status2==1 && status3==0){
  tone (lsPin, 262);
}
```

Lösung zu 1.9 (für Taster an pin 3)**:**

```
void loop() {
  while (digitalRead(3) == 1)
  {}
  for (int i = 0; i < 255; i = i + 1) {
    analogWrite(ledRot, i);
    delay(50);
  }
  while (digitalRead(3) == 1)
  {}
  for (int i = 255; i > 0; i = i - 1) {
    analogWrite(ledRot, i);
    delay(50);
  }
}
```


https://pixabay.com/vectors/music-note-music-note-musical-notes-1967480/

https://pixabay.com/de/photos/sonnenuntergang-sonnenaufgang-natur-3314275/

Hinweis: Achten Sie darauf, dass die Schülerinnen und Schüler den richtigen Variablentyp verwenden (siehe auch Arduino1 S.12).

Lösung zu 2.2:

```
float messwert=analogRead(sensor);

float spannung=messwert/1024*5;

Serial.println(spannung);
```

Hinweis: 5/1024*messwert ist zwar mathematisch korrekt, führt aber zu einem Rundungsfehler, da 5 und 1024 als ganze Zahlen interpretiert werden. Das Ergebnis wird immer null sein!

Trick: für 5 schreibt man 5.0

Lösung zu 2.3:

```
…
if (spannung >=1.4) {
  digitalWrite (ledGruen,HIGH);
  }
if (spannung <1.4 && spannung >=1.2 ) {
  digitalWrite (ledGelb,HIGH);
  }
if (spannung <1.2) {
  digitalWrite (ledRot,HIGH);
  }
```


Analog-Digital-Wandlung

Am analogen Eingang wird eine Spannung im Bereich von 0V und 5V einem ganzzahligen Wert zwischen 0 und 1023 zugeordnet.

Es ergeben sich 1024 = 2^{10} ganze Zahlen. Um die Zahl 1023 als Binärzahl zu schreiben benötigt man 10 Stellen. Diese Binärzahl benötigt einen Speicherplatz von10 Bit. Man spricht daher von einer 10-Bit-Auflösung.

Tipp:

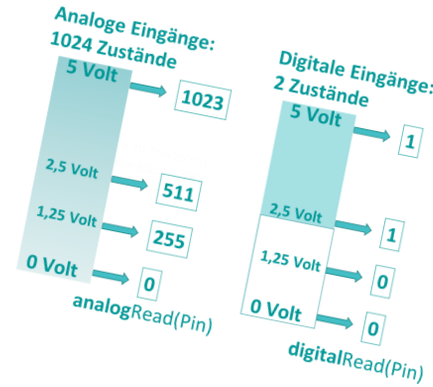
Wenn bei der Messung immer nur 0 angezeigt wird, muss man vielleicht die Anschlüsse tauschen, da der Arduino keine negativen Potenziale messen kann.

Da bei der Berechnung der Spannung in Volt Kommazahlen zu erwarten sind, musst du den geeigneten ***Variablentyp*** wählen: statt ***int*** muss die Variable als ***float*** deklariert werden.

Im seriellen Monitor kann die Anzahl der Nachkommastellen festgelegt werden:

Serial.println (messwert, 1) zeigt eine Nachkommastelle.

2 Analoger Eingang



Im ersten Heft hast du gelernt, wie der Arduino erkennen kann, ob ein Taster gedrückt wurde. Der Taster hat nur zwei Zustände: +5V (HIGH bzw. 1) und 0V (LOW bzw. 0). An den **analogen Eingängen** des Arduinos kann man Werte zwischen 0V und 5V messen. Der Befehl hierfür lautet ***analogRead(PIN).*** *Die Messwerte kannst du dir am seriellen Monitor anzeigen lassen.*

2.1

Übertrage folgendes Programm auf deinen Arduino, starte den seriellen Monitor und verbinde nacheinander den Pin A0 mit dem GND-, 5V- und 3,3V-Pin.

```
int sensor=A0;
int messwert;

void setup() {
  Serial.begin(9600);
}
void loop() {
  messwert=analogRead(sensor);
  Serial.println(messwert);
    delay(500);
}
```

2.2

Spannungsmesser

Lege zwei 1,5 Volt Batterien in den Batteriehalter ein. Die Batterien sind jetzt in Reihe geschaltet. Stecke das schwarze Kabel des Batteriehalters in GND und das rote Kabel in A0. Verändere dein Programm so, dass der angezeigte Wert dem Spannungswert (in mV) entspricht. Überprüfe mit einem Voltmeter, ob dein Arduino-Wert mit dem Messwert des Voltmeters vergleichbar ist.

2.3

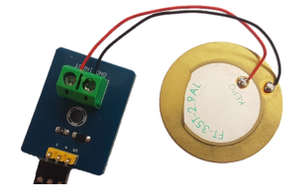
Batterietester

Nun wollen wir den Arduino als Batterietester nutzen. Dazu soll eine beliebige 1,5 V - Batterie angeschlossen werden. Ist die Spannung der Batterie mindestens 1,4 V, soll eine grüne LED leuchten, bei einer Spannung kleiner als 1,2 eine rote und dazwischen eine gelbe.

Tipp: *Man kann zwei Bedingungen miteinander verknüpfen, um einen Wertebereich zu definieren: (x>=3 && x<7) kennzeichnet den Bereich von 3 bis 6.*

2.4

Hau den Lukas!



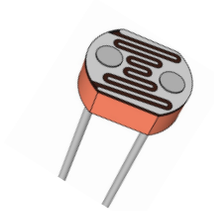
Schließe das Piezo-Vibrationsmodul an den Arduino an. Schreibe ein Programm, um das Sensormodul auszulesen. Wer hat den härtesten Schlag?

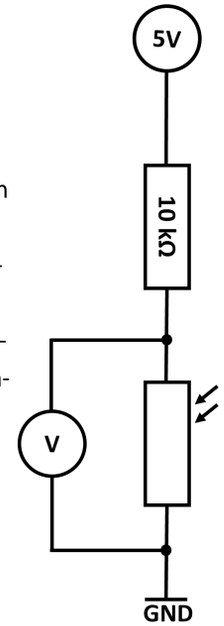
Ein Piezokristall reagiert auf Druck. Ändert sich der Druck, wird eine Spannung erzeugt. Je stärker die Verformung, desto höher die Spannung. Das vorgeschaltete Modul begrenzt die Spannung auf max. 5V.

***Achtung*: Beim Überschreiten von 5V wird der Mikrocontroller zerstört!**

Sensoren auslesen

Hier lernst du, wie du den Arduino als Messgerät für verschiedene Größen verwenden kannst. Da der Arduino nur Spannungen messen kann, benötigen wir zur Messung anderer Größen (z.B. Temperatur, Beleuchtungsstärke) eine Spannungsteiler-Schaltung. Für unseren ersten Spannungsteiler nehmen wir ein elektronisches Bauteil, dessen elektrischer Widerstand von der Beleuchtungsstärke abhängt. Dieses Bauteil heißt **LDR**. Der LDR wird in Reihe mit einem ohmschen Widerstand geschaltet. Ändert sich nun durch Helligkeitsänderung der Widerstandswert des LDR, so ändert sich die Spannung an den Anschlüssen des LDR.





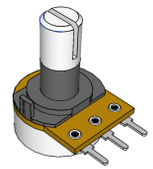
3.1 Baue die Spannungsteiler Schaltung auf (+5 V und GND vom Arduino) und messe mit einem Voltmeter die Spannung zwischen den Anschlüssen des LDRs bei unterschiedlicher Beleuchtung.

3.2 Führe die Messung mit dem Arduino durch und lasse dir die Werte am Bildschirm anzeigen. Verwende den analogen Eingang A0. Damit misst der Arduino automatisch die Spannung, die zwischen A0 und GND anliegt. Miss die Helligkeit an verschiedenen Stellen im Raum. Stelle die Werte mit Hilfe des seriellen Plotters dar (Menü „Werkzeuge“).

3.3 Programmiere einen Dämmerungsschalter. Bei einsetzender Dämmerung soll eine LED angehen. Lass dir, um den Schaltpunkt zu bestimmen, die Helligkeitswerte am seriellen Monitor anzeigen

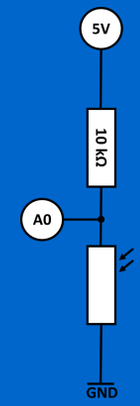
3.4 Ein Potentiometer (kurz Poti) ist ein regelbarer Widerstand. Zwischen den beiden äußeren Anschlüssen hat man einen konstanten Widerstandswert (z.B. R_{Ges}= 10 kΩ). Über den Drehknopf kann der Widerstand zwischen dem mittleren Anschluss und den beiden äußeren Anschlüssen verändert werden. Der Gesamtwiderstand wird hierbei aufgeteilt (z.B. R_1=7 kΩ, R_2=3 kΩ). Dadurch besteht das Poti im Prinzip also aus zwei regelbaren Widerständen in einem Bauteil—also einem Spannungsteiler. Baue eine entsprechende Schaltung auf und lass dir die Werte ausgeben.

3.5 Baue ein Stroboskop. Mit einem Poti als Drehregler soll die Blinkfrequenz einer LED gesteuert werden.



3

In der Regel sind solche Bauteile Widerstände, deren Widerstandswert sich verändert, wenn sich die entsprechende Größe ändert. Beispiele sind:
LDR (lichtabhängiger Widerstand), **NTC** (temperaturabhängiger Widerstand) etc.



Gekaufte Sensoren haben meistens 3 Anschlüsse. Diese folgen einem Farbcode:
Rot = 5 V (Vcc)
Schwarz=GND
Gelb = Messleitung (Vout)

Achtung: Der mittlere Anschluss des Poti darf, falls es als Spannungsteiler verwendet wird, nie mit 5 V oder GND verbunden werden—es droht sonst Kurzschlussgefahr.

Den Schülerinnen und Schülern sollte das Prinzip des Spannungsteilers bekannt sein. Genaueres zum Thema Spannungsteiler findet sich im LernBaustein Schaltungen2 (schaltungen2.nwt.schule)

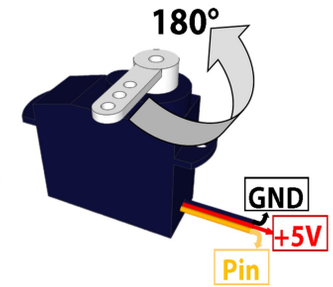
Lösung zu 3.3:

```
int ldr=A0;
int helligkeit;

void setup() {
  Serial.begin(9600);
  pinMode(13,OUTPUT);
}
void loop() {
  helligkeit=analogRead(ldr);
  // Serial.println(helligkeit);
  if (helligkeit > 500){
     digitalWrite(13,HIGH);
     }
  else {
     digitalWrite(13,LOW);
     }
}
```

Lösung zu 3.5:

```
void setup() {
    pinMode(13,OUTPUT);
}
void loop() {
  digitalWrite(13,HIGH);
  delay(analogRead(A0));
  digitalWrite(13,LOW);
  delay(analogRead(A0));
}
```

4 Servos



Im Modellbau und bei Robotern werden oft sogenannte Servo-Motoren verwendet. Sie können sich üblicherweise nicht um 360° drehen, dafür aber vom Mikrocontroller aus gezielt auf einen exakten Winkel eingestellt werden.

Servos haben nur 3 Kabel und können einfach angeschlossen werden: rot an 5 V, schwarz an GND und gelb an einen Pin der Pins 3, 5, 6, 9, 10 und 11 des Arduino. Das Signal wird ähnlich der Pulsweitenmodulation (PWM, siehe Heft 1 Seite 15) übertragen.

Für die Übungen hier im Heft genügen günstige Microservos (z.B. SG90). Diesen haben ein Nylon-Getriebe, das sich schnell abnutzt. Sie dürfen auf keinen Fall per Hand bewegt werden. Sie halten länger, wenn man sie statt an 5 V an 3,3 V anschließt.

Leistungsstärker, aber auch etwas teurer, sind Servos mit Metallgetriebe.

Wie funktioniert die Übertragung des Winkels an den Servo?

Ein Servo hat eine eingebaute kleine Elektronik, die den eingestellten Winkel mit dem gewünschten Winkel vergleicht und den eingebauten Motor dann in die entsprechende Richtung dreht. Modellbauservos benötigen die Information über den gewünschten Winkel alle 20 ms in Form eines längeren oder kürzeren HIGH-Signals auf der gelben Steuerleitung. Dauert das HIGH-Signal 544 Microsekunden, bedeutet das 0 Grad, bei 2400 µs 180 Grad. Die Werte dazwischen entsprechen dann der jeweiligen Gradzahl.

Die Servo-Bibliothek kümmert sich darum, dass nach dem Aufruf von S1.write(…) der entsprechende „Puls“ alle 20 ms gesendet wird.

4.1

Schließe einen Servo an den Arduino an und übernehme zunächst dieses einfache Programm.

```
#include <Servo.h>
Servo S1;

void setup() {
  S1.attach(9);
}
void loop() {
  S1.write(30);
  delay(1000);
  S1.write(90);
  delay(1000);
}
```

Auch für Servos wird eine Bibliothek eingebunden.

Für jeden angeschlossenen Servo legt man ein Objekt an. Man kann jeden gültigen Variablennamen benutzen: S1, Oma, Klaus…Üblich ist es, Objekte groß zu schreiben.

Hier wird der Servo an Pin 9 initialisiert.

So stellt man den Servo auf 30 Grad ein...

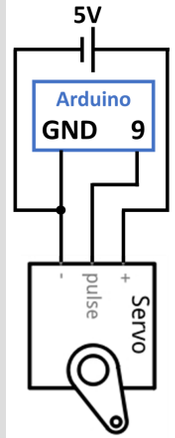
...und so auf 90 Grad.

4.2

Überprüfe, wie genau der Servo die Winkel anfährt. Dabei gibt es zwei Genauigkeiten: die erste ist die **Positionierungsgenauigkeit**: Liegen zwischen den im Programm angegebenen Winkeln 0° und 180° auch wirklich 180° Grad realer Winkel? Die zweite ist die **Wiederholgenauigkeit**: Wenn der Servo immer wieder auf z.B. 30 ° programmiert wird, dreht er sich dann auch immer in die gleiche Position?

Wenn die Positionierungsgenauigkeit nicht stimmt, kann man im Programm korrigierte Pulsdauern (siehe Marginalspalte) für 0° und 180° angeben:
S1.attach(9, 544, 2400);

Hinweis:
Unter Belastung benötigt ein Servo manchmal eine höhere Stromstärke, als der Arduino liefern kann. Dann schließt man das rote und schwarze Kabel an Plus- und Minuspol eines Netzteils bei 5 V an und verbindet zusätzlich den Minuspol des Netzteils mit GND auf dem Arduino.



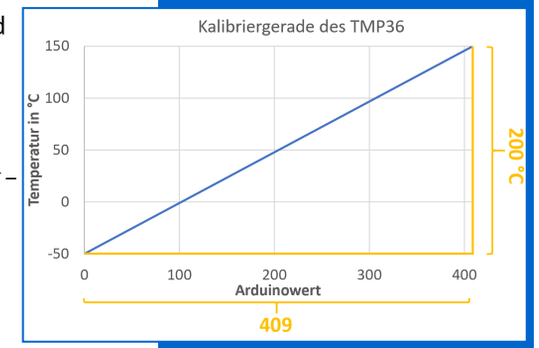
Kalibrieren von Sensoren

5

In Kapitel 3 hast du gelernt, wie man Sensoren ausliest und die gewonnenen Werte weiterverwendet. Die Werte liegen immer zwischen 0 und 1023. Will man den Arduino als Messgerät verwenden, ist es viel praktischer, wenn man sich das Messergebnis gleich in der gewünschten Einheit anzeigen lassen kann, also beispielsweise die Temperatur in °C - diesen Vorgang nennt man ***kalibrieren***. Dazu muss man herausfinden, wie „Arduinowerte" (0-1023) und die Werte in der gewünschten Einheit zusammenhängen. Oft findet man das im Datenblatt des Sensors, wenn nicht, muss man selbst messen...

Lineare Zusammenhänge

Bei vielen Sensoren ist der Zusammenhang zwischen Messwert und Ausgabewert linear. Ein solcher Sensor ist der Temperatursensor TMP36. Laut Datenblatt misst man eine Spannung zwischen 0 V bei –50 °C und 2 V bei 150 °C . Der Arduino zeigt also bei –50 °C den Wert 0 und bei 150 °C den Wert 409 (siehe Aufgabe 2.2). In der Randspalte sind die Werte als Diagramm dargestellt. Hieraus kann man sich nun die Geradengleichung ($y = m \cdot x + b$) bestimmen: Als Steigung erhält man $m = 200 / 409$. Außerdem schneidet die Gerade die y-Achse bei – 50, es ergibt sich also ein Achsenabschnitt von $b = -50$. So erhältst du die Gleichung $y = 200 / 409 \cdot x - 50$. Damit kann nun der Arduino seine gemessenen Werte in °C umrechnen.

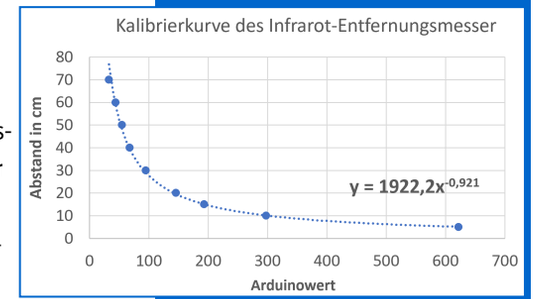



```
void loop() {
tempMessung=analogRead(0);
float temperatur= (200/409 * tempMessung) -50
```

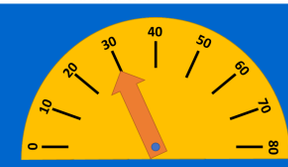
5.1 Ein weiterer Sensor zur Temperaturmessung ist der LM35. Er gibt bei 0 °C eine Spannung von 0V und bei 100 °C eine Spannung von 1 V aus. Bestimme für diesen Sensor die Umrechnung in °C.

Andere Zusammenhänge

Die Kennlinie des Infrarot-Entfernungsmesser ist keine Gerade. Entfernung und ausgegebene Spannung verhalten sich umgekehrt proportional zueinander: Je größer die Entfernung, desto kleiner der Ausgabewert. Die Abbildung zeigt das Ergebnis einer Messreihe mit diesem Sensor. Da du die Funktionsgleichung selber nur schwer bestimmen kannst, übertrage die Werte in ein Tabellenkalkulationsprogramm. Bestimme mit Hilfe des Programms die Funktionsgleichung (Trendlinie hinzufügen). Das Tabellenkalkulationsprogramm gibt dir die Funktion an, die du zur Umrechnung im Arduino-Code benötigst.



5.2 Kalibriere einen Infrarot-Entfernungsmesser und lass dir die Messwerte mit einem Zeiger anzeigen, den du mit einem Servo steuerst.



Tipp:

Lineare Zusammenhänge kann die Arduino-IDE mit Hilfe der map-Funktion auch automatisch umrechnen, hier z.B. den gemessenen Wert in die Temperatur:

```
Temperatur=
map(messwert,0,409, -50,150);
```

Urspr. Wertebereich | neuer Wertebereich

Wer sich noch genauer mit Kennlinien beschäftigen möchte, findet mehr Informationen im LernBaustein Auswertung2 unter auswertung2.nwt.schule.

Lösung zu 5.1:

Dieser Sensor ist noch wesentlich einfacher umzurechnen, da der Achsenabschnitt 0 ist. Es ergibt sich: $y = 5 \div 1023 \cdot 100 \cdot x$

Und damit für den Arduino-Code:

```
int lm35 = 0;
float tempMessung;
float Temperatur;

void setup() {}

void loop() {
tempMessung = analogRead(lm35);
Temperatur = 5 / 1023 * 100 *
tempMessung;
delay(100);
}
```

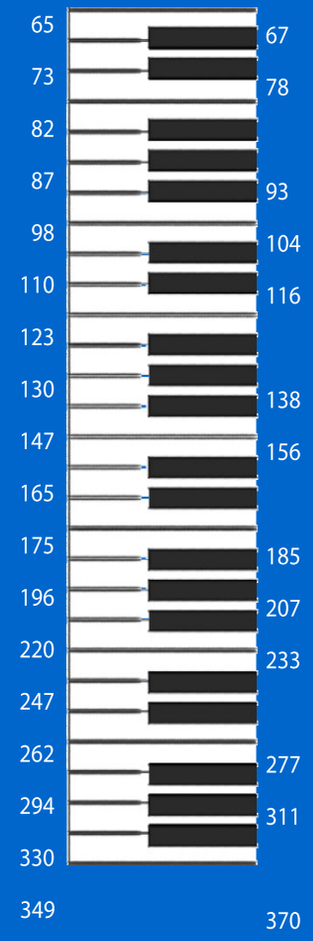
Lösung zu 5.2:

```
#include <Servo.h>

Servo myservo;

int IRSensor = 0;
float messWert;
float entfernung;

void setup() {
  myservo.attach(9);
}
void loop() {
  messWert = analogRead(IRSensor);
  entfernung = 3650 / messWert;
  myservo.write(entfernung);
  delay(15);
```

Eine Variable zu **deklarieren** bedeutet, dass man den Datentyp und den Namen festlegt. Weist man der Variablen einen Wert zu, so bezeichnet man diesen Vorgang als **Initialisieren**.

Der Datentyp eines Arrays legt fest, welchen Datentyp die einzelnen Elemente des Arrays haben. Die Länge eines Arrays steht in den eckigen Klammern. Die Länge des Arrays kann innerhalb eines Programms nicht mehr geändert werden. Sind in einem Programmcode alle Elemente in einem Array bereits eingetragen, kann man die Länge des Arrays weglassen. Ist nur ein Teil eingetragen, muss die Länge angegeben werden.

Um ein Buchstabe, eine Ziffer oder ein Sonderzeichen zu speichern, benötigt man einen Speicherplatz von der Größe eines Bytes (= 8 Bit). Als Datentyp verwendet man char. **Jeder** Buchstabe, Ziffer oder Sonderzeichen wird als Zahl gespeichert. In der ASCII Tabelle ist festgelegt, welchen Zahlenwert die einzelnen Zeichen haben. Die Ziffer 1 hat z.B. laut ASCII Tabelle den Wert 49, deshalb ist es nicht sinnvoll mit der 1 zu rechnen, wenn sie als Zeichen (**char)** gespeichert wurde. Der richtige Variablentyp hilft dem Mikrocontroller das Zeichen richtig zu interpretieren.



6 Arrays - arbeiten mit Listen

```
void setup() {
  tone(8, 175);
  delay(150);
  noTone(8);
  delay(10);
  tone(8, 175);
  delay(150);
  noTone(8);
  delay(10);
  tone(8, 196);
  delay(300);
  noTone(8);
  delay(10);
```

Im ersten Heft hast du gelernt, wie man mit dem Arduino Töne spielen kann - man benutzt die tone-Anweisung (*tone(Lautsprecherpin, Frequenz)*). Allerdings wird so ein Programm sehr schnell unübersichtlich, wenn man alle Töne einzeln als Befehl eingeben muss. Viel einfacher ist es, die Frequenzen in eine Liste zu schreiben, und diese Liste dann eins nach dem anderen abarbeiten zu lassen. Solche Listen nennt man **Arrays.** Ein (leeres) Array für unsere Melodie **deklariert** man im Code so:
int *Melodie[6];* Die Zahl in den eckigen Klammern gibt an, wie viele Elemente das Array enthält (kann man auch leer lassen).

Das Array mit dem Namen Melodie enthält 6 Elemente. Dabei handelt es sich um die Frequenzen für unser Lied.

In dieses Array kann man nun die Frequenzen schreiben, aus denen unsere Melodie besteht:

Melodie	175	175	196	175	262	247
Index	0	1	2	3	4	5

int *Melodie[] = {175,175,196,175,262,247};*

Das erste Element einer Liste bekommt immer den Index 0. Die Frequenz mit dem Index 2 ist hier also 196.

Jedes Element der Liste bekommt einen **Index** - so kann man die einzelnen Elemente wieder aufrufen:
tone(8, Melodie[0]); spielt am Pin 8 einen Ton mit der Frequenz von 175 Hz,
tone(8, Melodie[2]); spielt am Pin 8 einen Ton mit der Frequenz von 196 Hz.

Jetzt können wir unsere Melodie abspielen lassen:

```
int melodie[] = {175,175,196,175,262,247};
int notenWert[] = {8,8,4,4,4,2};
int tonDauer;
int pauseZwischenNoten;
void setup() {
  for (int i = 0; i < 6; i = i + 1) {
    tone(8, melodie[i]);
    tonDauer = 1000 / notenWert[i];
    delay(tonDauer);
    noTone(8);
    pauseZwischenNoten = tonDauer * 0.30;
    delay(pauseZwischenNoten);
    }
}
void loop() {}
```

Das Array mit den Frequenzen

Ein weiteres Array für die Notenwerte, also Achtel, Viertel, Halbe...

For-Schleife, bei der die Variable i (wie Index) von 0 bis 5 läuft.

Hier spielt man den Ton aus dem Array mit dem Index i, also beim ersten Durchlauf den Ton mit Index 0 (175 Hz), dann Index 1 ...

Die Tondauer berechnet man, indem man eine Sekunde durch den Notenwert aus dem anderen Array teilt. Eine Achtel dauert dann 125 ms, eine Viertel 250 ms ...

Die Pause zwischen den Noten soll 30% der Tondauer betragen, das hört sich ganz gut an, du kannst aber auch mit anderen Werten experimentieren.

6.1 Tippe den Code ab, schließe einen Lautsprecher an und lass die Melodie spielen.
Um welches Lied handelt es sich?

Die restlichen Frequenzen und Notenwerte sind:
int melodie[] ={196,196,220,196,262,247,196,196,220,196,294,262,196,196,392,330,262,247,220,349,349,330,262,294,262};

int notenWert[] = {8,8,4,4,4,2,8,8,4,4,4,2,8,8,4,4,4,4,4,8,8,4,4,4,2};

Ergänze das Lied. Lass den Arduino bekannte Melodien spielen oder erfinde eigene Kompositionen.

Wenn du in einem Array Buchstaben statt Zahlen auflisten möchtest, musst du einen anderen Variablentyp benutzen - mit einem ***char*** **(**kommt von Charakter) kann man Zeichen (Buchstaben, Ziffern und Sonderzeichen) speichern:

char *zeichen[] = {'P', 'f', 'e', 'r', 'd', '1'};* ist ein Array, das 6 Zeichen enthält.

Mit ***Serial.println****(zeichen[3]);*

kannst du dir das Zeichen mit dem Index 3, also das **r**, am Bildschirm anzeigen lassen.
Lässt man den Index und die eckigen Klammern weg kann man alle Zeichen aufrufen:

Serial.print*(zeichen);* zeigt **Pferd1** am Monitor an.

Ganze **Wörter** (man spricht auch von *„Zeichenketten"*) kann man in einem Array speichern, wenn man noch zusätzlich einen sog. *Pointer* (*) verwendet:

char* mitSchueler[] = {"Martin","Volker","Frank","Kolja","Alexander","Mario","Peter","Rainer"};

Serial.println(mitSchueler[1]); zeigt **Volker** am seriellen Monitor an.

6.2 **Tafeldienst** auslosen
Fertige ein Array mit den Namen deiner Klassenkameraden an. Schreibe ein Programm, welches mittels einer Zufallsfunktion über den Index den Tafeldienst bestimmt.

CC0, https://pixabay.com/de/photos/graffiti-tafel-liebe-hand-2227941/

6.3 **Literaturautomat**
Wer reitet so spät durch x und y?
Fertige Arrays an mit möglichen Wörtern, die in die Lücken passen. Erfinde deinen eigenen Literaturautomat.

6.4 **Hausarbeiten**
Programmiere eine Maschine, welche die anfallenden Hausdienste auf deine Familienmitglieder verteilt. Niemand darf doppelt eingesetzt werden.
Am Monitor erscheint per Knopfdruck zum Beispiel: Papa darf abwaschen.

Achtung: Wenn du eine Ziffer als **char** speicherst, wird sie als Text behandelt - du kannst dann keine korrekten Rechenoperationen mit ihr durchführen.

In einem char-Array werden die Zeichen immer mit einem einfachen Anführungszeichen versehen. Benutzt du einen Pointer (*), um ganze Wörter zu speichern, so müssen diese in doppelten Anführungszeichen stehen.

Arrays können beschrieben, und auch wieder geändert werden. Die Anweisung

werte[3] = **analogRead**(A0);

speichert den analogen Wert auf dem Listenplatz 3.

Beachte dass die Messwerte, die im Programmablauf in ein Array geschrieben wurden, verloren gehen, wenn der Arduino neu gestartet wird!

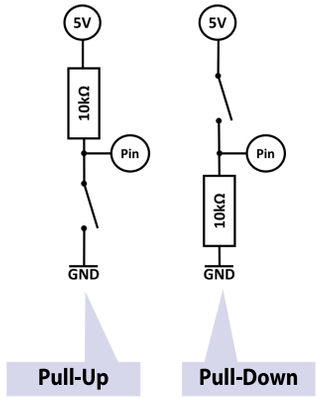
Tipp zu 6.1:
Die For-Schleife muss hier bis i<25 laufen, sonst werden trotzdem nur 6 Töne gespielt.

In einem char-Array können nur einzelne Zeichen gespeichert werden. Um das Array zu erweitern, verwenden wir einen Pointer * . Der Speicherplatz für ein einzelnes Zeichen wird für das Speichern einer Zeichenkette erweitert. Die **Zeichenkette** muss innerhalb von **doppelten** Anführungszeichen stehen.

Lösung zu 6.2:
```
char*  mitSchueler[] =
{"Martin","Volker","Frank","Kolja",
"Alexander","Mario","Peter","Rainer"};
int x;
void setup () {
Serial.begin (9600);
randomSeed (analogRead(A0));
x = random (0,8);
Serial.println(mitSchueler[x]);
}
```
Lösungsvorschlag zu 6.3:
```
char*  wort[] =
{„Pfuetze",„Wueste",„Schlamm",„Wie
se",„Fluss",„Geroell",„Kueche","Su
ppe"};
int x;
int y;
void setup {
Serial.begin (9600);
randomSeed (analogRead(A0));
x = random (0,8);
y = random (0,8);
Serial.print(„Wer reitet so spaet
durch ");
Serial.print(wort[x]);
Serial.print(„ und ");
Serial.println(wort[y]);
}
```

7 Interrupt

Um Schalter abzufragen benötigt man im Prinzip einen Spannungsteiler. Dazu gibt es zwei grundsätzliche Schaltungen:



Bei geöffnetem Schalter ist das Potenzial am Pin in der Pull-Up-Schaltung 5V (up) , weil der Schalter in diesem Fall einen unendlich hohen Widerstandwert hat. Entsprechend ist das Potenzial am Pin in der Pull-Down-Schaltung 0 V (down).

Je nach Schaltungstyp ändert sich das Potenzial beim Tasterdruck am Pin.

Vertauscht man GND und 5V, so wird aus einer Pull-Up-Schaltung eine Pull-Down-Schaltung. Diesen Vorgang nennt man **invertieren**.

Verwendet man den Parameter INPUT_PULLUP wird ein interner Pull-Up-Widerstand verwendet. Man muss den Taster nur mit dem Pin am Arduino und dem GND verbinden.

Einen internen Pull-Down-Widerstand gibt es für den Arduino nicht.

Ein Interrupt unterbricht kurzzeitig das laufende Programm, damit eine hinterlegte Funktion durchgeführt werden kann.

Zweck eines **Interrupts** ist es auf ein **äußeres Ereignis**, ein **Zeitsignal** oder ankommende **Daten** sofort **zu reagieren**.

Die Abfrage eines Sensor an verschiedenen Stellen im Programm nennt man **Polling**.

Nur an den Pin **2** und **3** kann die Interrupt-Funktion aufgerufen werden

attachInterrupt() hat drei Parameter.
1. Interrupt-Pin
2. Name der **ISR**
3. Signaländerung

Die hinterlegte Funktion nenn man auch **Interrupt Service Routine (ISR).**

Tastersignale sicher erkennen

Mit Zähl-Schleifen kann man Lauflichter und Musikstücke elegant programmieren. Doch was macht man, wenn man eine zufällige, einmalige Aktion auf Tasterdruck durchführen möchte, während die Schleife abgearbeitet wird? Eine solche Aktion könnte zum Beispiel eine rote LED sein, die nach **kurzem** Tasterdruck angeht. Befindet sich innerhalb der Schleife ein **delay()** ist diese Aufgabe kaum zu lösen, da die Tasterabfrage nur dann stattfindet, wenn sich der Mikrocontroller grade an dieser Stelle des Programms befindet. Damit der Tasterdruck erkannt wird, müsste man also exakt zu diesem Zeitpunkt den Taster drücken. Doch woher soll man diesen Zeitpunkt kennen? Reine Glückssache! Das Gleiche gilt natürlich auch für alle anderen Programmstrukturen. Hier müsste man zusätzlich in regelmäßigen Abständen die Tasterabfrage im Programm platzieren. Je länger das Programm desto öfter!
Mit einem Interrupt geht es sicherer. Der Interrupt reagiert auf eine Änderung an einem Pin und unterbricht das Hauptprogramm sofort, um eine hinterlegte Funktion aufzurufen. Ist diese abgearbeitet, so macht das Hauptprogramm an der unterbrochenen Stelle weiter.

7.1

Tippe das Programm ab. Baue das Lauflicht mit acht Leds und einem Taster.

```
int rot = 12; int taster = 2;
void setup() {
  for (int i = 4; i < 13; i++) {
    pinMode(i, OUTPUT);
  }
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(taster),roteLed ,FALLING);
}
void loop() {
  for (int i =  4; i < 12; i++) {
    digitalWrite(i, HIGH);
    delay(500);
    digitalWrite(i, LOW);
  }
  digitalWrite(rot, LOW);
}
void roteLed(){
  digitalWrite(rot,HIGH);
}
```

Hier wird festgelegt, welches Unterprogramm aufgerufen wird, wenn der Interrupt auslöst.

Festlegung des Interrupt-Pins

Hier wird der Interrupt aktiviert

Hier wird festgelegt, auf welches Signal der Interrupt reagieren soll:
FALLING: das Signal wechselt von 1 auf 0.
RISING: das Signal wechselt von 0 auf 1.
LOW: das Signal ist 0.
CHANGE: das Signal wechselt.

Weitere Anwendungen des Interrupts

Grundsätzlich bieten sich Interrupts an, wenn ein Signal sofort registriert werden soll (z.B. Kollisionswarnung im Auto) oder wenn die erwarteten Signale sehr kurz sind, so dass die Gefahr besteht, dass sie nicht registriert werden. Ein Beispiel hierfür wäre die Induktionsschleife an einer Ampel. Wenn die Spannungsänderung nicht registriert wird, schaltet die Ampel nicht um.
Wichtig ist, dass die Interruptfunktion möglichst kurz gehalten wird. Deshalb verwendet man in der **ISR kein delay(), keinen seriellen Monitor, kein LC-Display, keine aufwändigen Berechnungen**. Die millis()-Funktion wird während des Interrupts nicht aktualisiert!

Wenn es ausreicht, dass man sich das eingehende Ereignis merkt, kann man folgende Lösung verwenden.

7.2 Enwickle eine Idee für die leere Funktion **machIrgendwas()!**

```
int taster = 2;
volatile int zustand = 0;

void setup() {
  for (int i = 4; i < 13; i++) {
    pinMode(i, OUTPUT);
  }
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(taster),meineISR ,FALLING);
}
void loop() {
  for (int i =  4; i < 12; i++) {
    digitalWrite(i, HIGH);
    delay(500);
    digitalWrite(i, LOW);

    if(zustand == 1){
       machIrgenwas();
       zustand = 0;  //zurücksetzen der Variablen zustand auf 0
    }
  }
}
void meineISR(){zustand = 1;}

void machIrgenwas(){…} // ist noch leer, ergänze!
```

Variablen im Interrupt müssen als volatile deklariert werden!

Interrupt-**S**ervice **R**outinen (ISR) müssen **kurz** gehalten werden.

Variablen die im Interrupt geändert werden, müssen als **volatile** markiert werden.

Wenn man eine Variable nur zwischen 0 und 1 ändert könnte man auch den Variablentyp **boolean** verwenden.

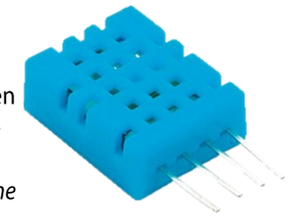
volatile boolean zustand = 0;

Vorsicht bei Tastern!

Wenn man die eingehenden Signale aufaddieren möchte, dann sollte der Taster entprellt sein!

Eine Hardwareentprellung mit Kondensator ist die wirksamste Methode. Oder man kauft sich einen bereits entprellten Taster.

8 Neue Dinge **entdecken**



Die „Arduino-Welt“ bietet eine riesige Anzahl verschiedenster fertiger Sensoren und Aktuatoren, die du verwenden kannst, um deine eigenen Projekte zu verwirklichen. In diesem Kapitel wird dir gezeigt, wie du mit Hilfe der Arduino-Community neue Dinge entdecken kannst. *To boldly go, where no man has gone before…*

Um einen passenden Sensor zu finden und in Betrieb zu nehmen, musst du drei Dinge klären:

- welcher Sensor ist geeignet (seine Bezeichnung),
- wie schließt man ihn an,
- wie programmiert man ihn.

Meist findest du beides gleichzeitig auf einer der gängigen Arduino-Plattformen. Mit etwas Übung hast du schnell herausgefunden, mit welchen du am Besten zurechtkommst (z.B. Funduino, Adafruit, Seeed Grove, Elegoo, Keystudio, Smraza, Sunfounder, Heise, Sparkfun…).

Das erste Beispiel ist ein Sensor zur Messung der relativen Luftfeuchtigkeit und der Lufttemperatur – der **DHT11**.

Suche nach einem Beispielprogramm (ein „hack“). Was soll in einem guten Hack sein? Schaltplan, kommentierter Beispielcode und, falls benötigt, die Programmbibliothek.

Öffne das Beispielprogramm, das für die Arduino-IDE gedacht ist. An #include <DHT.h> zu Beginn des **Sketches** kannst du erkennen, ob und welche Programmbibliothek du benötigst. Wie man Programmbibliotheken herunterlädt, findest du im Arduino1–Heft auf Seite 21.

Schließe deinen Sensor laut Schaltplan an. Wenn kein Schaltplan vorhanden ist, findest du oft Hinweise in der Kopfzeile des Beispielcodes. Achte auf die richtige Verkabelung. Lade den Sketch hoch. Öffne den seriellen Monitor, über den die Daten ausgegeben werden.

Lass dich nicht verunsichern, wenn Programmteile auftauchen, die du nicht verstehst; es kommt nur auf das Verständnis der Grundfunktion an. Schaue, wo im Programm die sichtbaren Aktionen z.B. Displayausgaben, SerialMonitor ausgeführt werden. Experimentiere vorsichtig mit dem Code herum, kommentiere zum Beispiel die Dinge aus, die du nicht brauchst.

CC0, https://pixabay.com/photos/weather-house-weather-station-2728931/

Jetzt kannst du deine eigene Wetterstation damit bauen, die dir dann auf einem Ausgabegerät deiner Wahl (z.B. ein LC-Display) die Wetterdaten für Luftfeuchtigkeit und Temperatur ausgibt. Oder du kannst bei dir daheim das Raumklima überwachen…

Aufbau des DHT11-Sensors mit einem Pullup-Widerstand für eine zuverlässigere Datenübertragung. Dieses Schaltelement findet man häufig bei Datenbussen.

Achtung! Es gibt manchmal verschiedene Programmbibliotheken mit gleichem Namen.

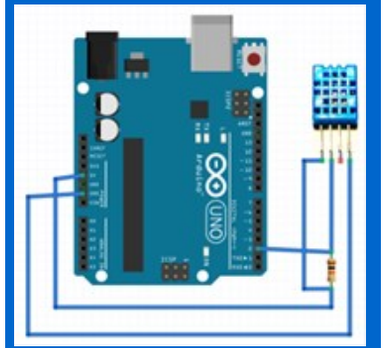
Hat man eine falsche, aber namensgleiche Bibliothek, so muss diese vorher gelöscht werden!

Die heruntergeladenen Bibliotheken findet man im Ordner *Dokumente/Arduino/libraries* .

#define bezeichnet man als Präprozessordirektive. Beim Kompilieren wird im Sketch überall dort, wo der Bezeichner (hier: DHTPIN) auftaucht, der zugeordnete Wert eingesetzt.

Es ist zu beachten, dass #define kein Befehl darstellt. Deshalb steht am Ende kein Semikolon! #define DHTPIN 2 kann innerhalb des Sketches nicht mehr geändert werden.

Aufbau des DHT11-Sensors mit 10 kΩ Pullup-Widerstand an der Datenleitung:



Häufig werden mit der Programmbibliothek Beispielprogramme mitinstalliert. Du findest sie unter Datei/ Beispiele unter der Rubrik Beispiele aus eigenen Bibliotheken.

```
#include <DHT.h>

#define DHTPIN 2
```

Manchmal findest du in Beispielcodes *#define*—damit definiert man Konstanten. Im oberen Beispiel hättest du auch int DHTPIN=2; schreiben können. Damit hättest du aber Arbeitsspeicher verbraucht, da für Variablen Arbeitsspeicher benötigt wird.

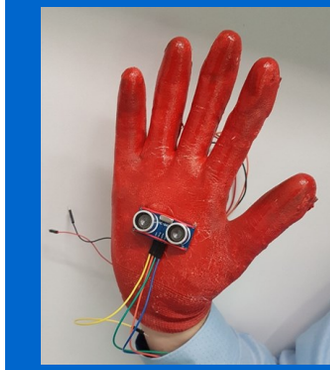
Noch mehr **Neues**

Neopixel

Neopixel (oder auch WS2812 LEDs) sind RGB-LEDs, die miteinander vernetzt werden können. Man erhält sie als Einzel-LEDs oder als Streifen, Ringe, Matrix, etc. Das Besondere daran ist, dass sie nur 3 Anschlüsse haben (+5V, GND und Datenleitung). So kannst du mit nur einem Digital-Pin des Arduino vielen LEDs einzeln alle mögliche Farben zuordnen - Für tolle Licht-Effekte bei deinem nächsten Projekt...

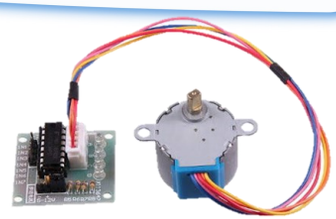
Ultraschall-Entfernungs-Sensor

Ein Ultraschall-Entfernungs-Sensor kann einen Abstand zu einem Objekt messen, ähnlich wie der Infrarot-Entfernungsmesser von S.9. Die Vorteile dieses Sensors sind, dass er wesentlich günstiger ist, und dass er die Entfernung direkt in cm ausgeben kann, ohne dass man ihn kalibrieren muss. Allerdings ist er lange nicht so schnell, wie der Infrarot-Entfernungsmesser. Für die allermeisten Projekte ist er aber schnell genug.



Schrittmotor

Der Servo-Motor von S. 8 kann ganz bestimmte Winkel anfahren. Der Schrittmotor ist ein weiterer Motor-Typ. Er kann eine bestimmte Anzahl an Schritten gehen (oft entsprechen ca. 2000 Schritte einer Umdrehung). Er eignet sich besonders für die Projekte, in denen etwas sehr präzise gesteuert werden soll.

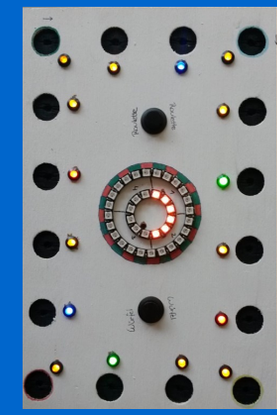


Keypad

Ein Tastenfeld für den Arduino. Dir fallen bestimmt selbst viele spannende Anwendungsmöglichkeiten ein.

UV-Sensor

Ein UV-Sensor kann die Intensität von UV-Strahlung messen. Damit ist er einer von vielen Sensoren zur Messung von Umwelt-Parametern und ein gutes Werkzeug für Forschungsprojekte.



9 I²C-Bus

Die Präprozessordirektive #include <Wire.h> bindet die Wire.h Bibliothek ein. In ihr sind die Befehle für das I2C-Protkoll abgelegt. In einem Protokoll wird geregelt, zu welchem Zeitpunkt oder in welcher Reihenfolge ein Vorgang durch wen oder durch was durchgeführt wird. Das Vorhandensein eines Protokolls ist die Grundlage für eine Kommunikation zwischen Geräten. Immer wenn der I2C-Datenbus verwendet wird, benötigt man diese Bibliothek. Manchmal erfolgt der Aufruf dieser Bibliothek automatisch, wenn man für das angeschlossene Gerät eine weitere Bibliothek benötigt. Man erkennt es daran, dass im Beispielprogramm #include <Wire.h> fehlt . Man kann sie ohne Schaden nochmals aufrufen.

Bei den meisten Geräten sind die in der Abbildung dargestellten 10kΩ-Widerstände bereits verbaut.

Das I2C-Bussystem hat eine bidirektionale Master-Slave-Architektur. Der Master kann Daten verschicken und empfangen.

SCL und *SDA* werden an die entsprechenden Pins am Arduino auf der „digitalen" Seite angeschlossen oder auf der „analogen" Seite an Pin A5 (SCL) und an Pin A4 (SDA).

Werden mehrere Geräte angeschlossen (es sind bis zu 112 möglich) kann die Leistung des +5V-Ausgangs des Mikrocontrollers an seine Grenzen kommen.

Ein **Datenbus** ist ein System zur Datenübertragung zwischen mehreren Teilnehmern über einen gemeinsamen Übertragungsweg.

Damit Geräte am I2C-Bus teilnehmen können, müssen sie über ein I2C-Modul verfügen.

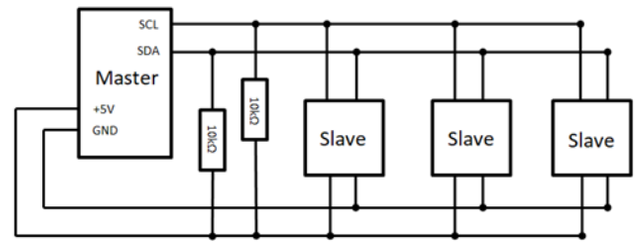
Eine Hexadezimalzahl erkennt man an dem vorangestellten **0x**.

In diesem Kapitel lernst du, wie man ganz einfach ohne großen Aufwand mehrere elektronische Bauteile mit dem Mikrocontroller verbinden kann, damit sie untereinander Daten hin und her schicken können. Einfach und ohne großen Aufwand hört sich gut an, oder?

Höchstwahrscheinlich hast du diese Möglichkeit des Datenaustauschs schon einmal genutzt, als du das I2C LC-Display verwendet hast. Der geringe Aufwand besteht darin, dass man nur vier Kabel zum Anschließen benötigt: V_{cc}, *GND*, *SCL* und *SDA*.

Möchte man weitere elektronische Bauteile einbinden, schließt man diese an die bereits vorhandenen Leitungen an.

SCL (Serial Clock Line, Taktleitung) und *SDA* (Serial Data Line, Datenleitung) dienen dem eigentlichen Datenaustausch.
Vcc (+5V oder +3,3V) und *GND* dienen der Energieversorgung der elektronischen Teilnehmer.



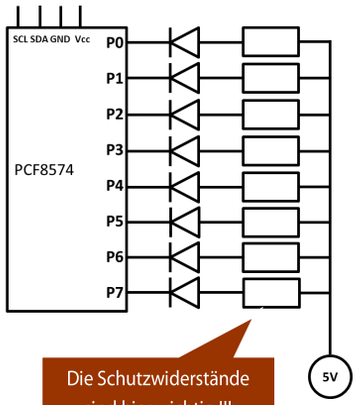
Die Abbildung stellt das I2C-Bussystem mit einem Master und drei Slaves dar.
Der Mikrocontroller (Master) steuert die Übertragung und fordert die Teilnehmer (Slave) gezielt dazu auf, Daten zu senden oder zu empfangen. Der Datenstrom erfolgt in beide Richtungen.
Da alle Teilnehmer die gleichen Datenleitungen benutzen, kann eine störungsfreie Kommunikation nur erfolgen, wenn der Ablauf genau geregelt ist. Die Regeln für einen Datenaustausch bezeichnet man als Protokoll.
Damit ein Teilnehmer weiß, dass er Daten senden oder empfangen soll, benötigt er eine eindeutige Adresse. I2C-Module haben in der Regel Standardadressen. Sie werden als Hexadezimalzahl angegeben. Für das I2C-LC-Display ist die Standardadresse 0x27.
Adressen von Teilnehmern kann man leicht mit einem ***I2C-Scanner*** herausfinden. Du musst nur das entsprechende Programm auf den Mikrocontroller laden, den Teilnehmer anschließen und das Programm starten. Auf dem seriellen Monitor wird die Adresse angezeigt.

Wenn Pins fehlen - der **Portexpander** 10

Sicherlich hattest du schon viele tolle Projektideen und musstest feststellen, dass dein Arduino **zu wenige Pins** hat. Kein Problem, mit Hilfe eines **Portexpanders** kannst du die Anzahl der Ein– und Ausgänge deines Mikrocontrollers erweitern. Das PCF8574-Remote-Modul wird über einen I2C-Datenbus mit dem Arduino verbunden.

10.1 a) Bestimme mit Hilfe des I2C-Scanners die Adresse deines PCF-Remote-Moduls.

b) Schließe die acht LEDs an das Modul an. Achte auf die korrekte Polung der LEDs. Teste das Programm. Ändere das Lichtmuster!

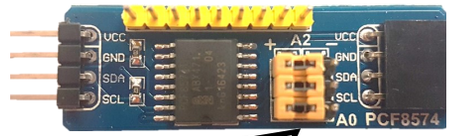



```
#include <Wire.h>
#define address 0x20

void setup() {
  Wire.begin();
  Wire.beginTransmission(address);
  Wire.write(0b10101010);
  Wire.endTransmission();
  delay(5); //wichtig!
}
void loop() {} //leer
```

10.2 Programmiere verschiedene Lichtanimationen. Verwende Arrays, in denen du die Binärzahlen für die verschiedenen Lichtmuster ablegst.

10.3 Programmiere ein Lauflicht für das Modul. Nutze den **bitweise Verschiebungsoperator**.

Tipp:
byte a = 0b00000001 << 1
verschiebt die eins in der Binärzahl um eine Stelle nach links und speichert sie in der Variablen a.
Mit Serial.println(a, BIN) kann man Binärzahlen am Monitor anzeigen.



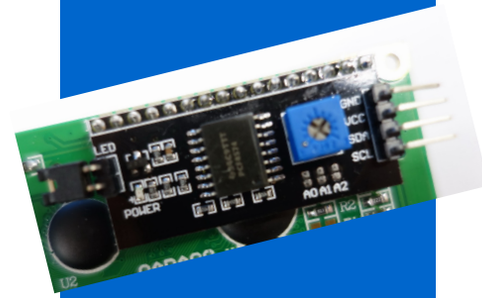
Man kann bei einigen PCF-Modulen die Adresse manipulieren, indem man die Jumper an A0, A1, A2 umsteckt. Aufgrund der acht Kombinationsmöglichkeiten, kann man sieben weitere Module mit unterschiedlicher Adresse ansteuern!

Das PCF8574-Modul hat eine sehr kleine Ausgangsstromstärke von 100µA. Das reicht nicht um eine LED zum leuchten zu bringen. Die Eingangsstromstärke ist mit 25mA deutlich höher. Deshalb verwendet man die Pins P0-P7 als GND.

An heißt **0**.

Aus heißt **1**.

10101010 ist eine Binärzahl. Damit der Mikrocontroller sie als solche erkennt, wird 0b vorangestellt.



Das I2C-Modul deines LC-Displays ist auch ein PCF-Remote-Modul. Auch hier kannst du die Adresse ändern indem du einen Steg in AO, A1 oder A2 einlötest .

Wire.write(0b11111110) bedeutet, dass eine LED an geht.

Es gibt verschiedene Portexpandermodule mit teilweise unterschiedlichem Datenbus. Die meisten kommunizieren über I2C-Datenbus mit dem Arduino. Ein solches ist das MCP23017 mit 16 Pins. Seine Ansteuerung ist allerdings etwas aufwändiger.

An dieser Stelle bietet sich an Zahlensysteme im Unterricht zu besprechen. (Dezimal-, Binär– und Hexadezimal)

Das PCF8574 kann auch bidirektional verwendet werden. Man kann die Pins P0-P7 auch auslesen, wenn man Taster anschließen möchte.

Beispiel:

```
#include <Wire.h>
byte adr = 0x20; // PCF-Adresse
byte daten = 0; // gelesenes Byte
void setup() {
  Wire.begin();
  Serial.begin(9600);
}
void loop() {
Wire.requestFrom(adr, 1);
if (Wire.available()) {
    daten = Wire.read();
  }
  Serial.println(daten, BIN);
// Daten als Binärzahl anzeigen
  delay(1000);
}
```

mit **bitRead**(daten,2) kann man den Zustand des **Pins 2** am PCF abfragen

11 Troubleshooting

Es reicht nicht aus die Programmbibliothek ins Programm zu schreiben, sie muss auch in der IDE vorhanden sein. (Sketch -> Bibliothek einbinden)

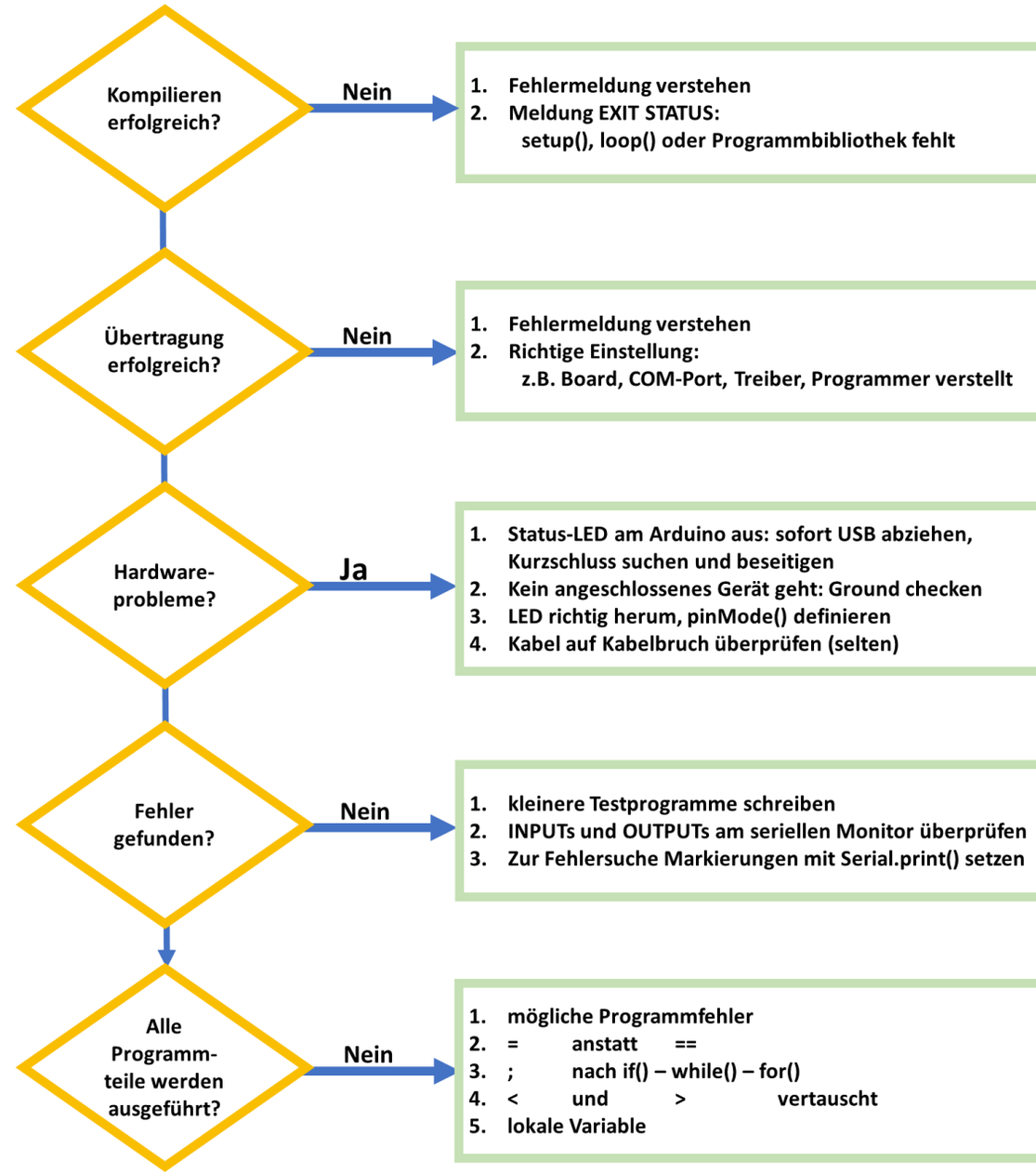
Der richtige Programmer ist “AVRISP mkll“. (Werkzeuge -> Porgrammer)

In eine While-Schleife könnte man z.B. schreiben Serial.print(“Jetzt bin ich in der While-Schleife“);

Setzt man nach der runden Klammer einer Schleife oder Verzweigung ein Semikolon, so wird der Programmblock in der geschweiften Klammer nicht ausgeführt und übersprungen.

Fehler sind völlig normal beim Programmieren. Sie selbst zu finden und zu beheben ist eine wichtige Fähigkeit. Auch hier gilt Übung macht den Meister.

Eine weitere Fehlerquelle ist die Hardware. Sind alle Geräte und Bauteile richtig angeschlossen?



Index

Autorenteam:
Mario Klein, Rainer Kügele,
Martin Merkle, Kolja Meyer,
Alexander Mink, Volker Rust,
Frank Trittler, Peter Weber