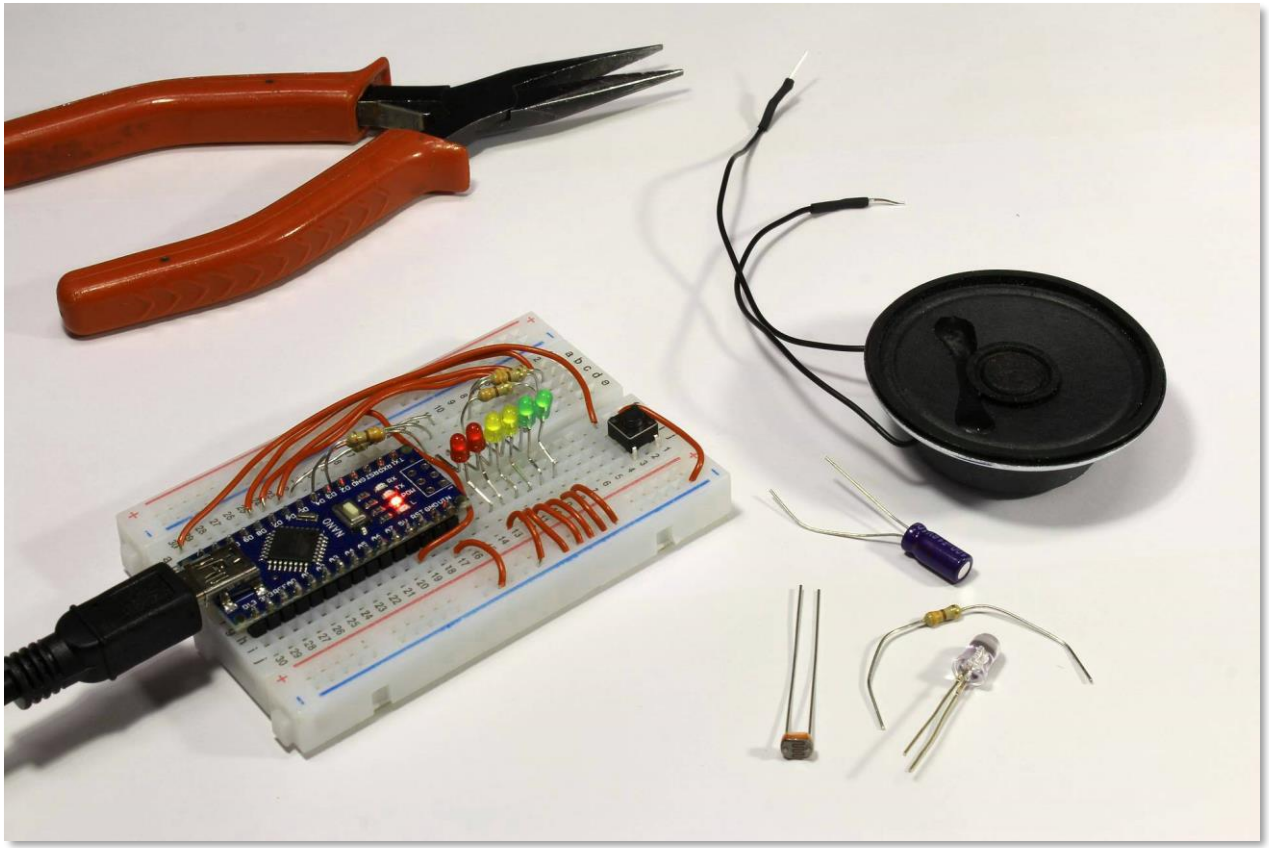
Arduino Schnupper Kurs

Stephan Laage-Witt
FES Lörrach - Juli 2017



FREIE EVANGELISCHE SCHULE
LÖRRACH

Gliederung

1. Einführung und Begriffe
2. Experimente mit LEDs und Tastern
   - Blinker, Lauflicht
   - Eingabe mit Taster
   - Zufallszahlen-Generator und Würfel
3. Experimente mit Frequenzen und Pulsweiten-Modulation
   - Töne aus dem Lautsprecher
   - Mini-Orgel
   - Leistungsregelung von einer LED
4. Experimente mit dem Analog-Digital-Converter (ADC)
   - Helligkeitsmessung mit dem LDR
   - Lichtschranke
   - Lichtorgel

Liste der Experimente

1. Teil 2: Digitale Ein- und Ausgänge
    1. Eine LED leuchtet
    2. LED-Blinker
    3. Lauflicht
    4. Lauflicht mit Input-Taster
    5. Würfeln
2. Teil 3: Frequenzen erzeugen, Pulse-Weiten-Modulation
    6. Würfeln mit Lautsprecher
    7. Melodie
    8. LED dimmen mit Pulsweiten-Modulation
3. Teil 4: Analog-Digital Converter (ADC)
    9. LDR mit dem Analog Digital Converter (ADC) auslesen
    10. Lichtschranke
    11. Lichtorgel

Arduino C Kurz-Referenz

Programm Block

```
{
      // Anweisungen …
}
```

Bedingte Verzweigung

```
if (x > 100) {
      // Anweisungen, wenn die
      // Bedingung zutrifft
};
```

Bedingte Verzweigung mit Else-Zweig

```
if (x > 100) {
      // Anweisungen, wenn die
      // Bedingung zutrifft
} else {
      // Anweisungen, wenn die
      // Bedingung nicht zutrifft
}
```

For-Schleife

```
for (i = 0; i < 10; ++i) {
      // Diese Anweisungen werden
      // mehrfach (hier 10 mal) ausgeführt.
      // Die Variable i läuft von 0 bis 9
};
```

Die wichtigsten Arduino-Funktionen:

`setup(void)` … wird einmal ganz am Anfang des Programms ausgeführt.

`loop(void)` … wird dauerhaft nach `setup()` ausgeführt

`pinMode(pin, mode)` … setzt die Betriebsart für einen der digitalen Pins. Pin: alle Werte zwischen 2 und 13. Mode: `OUTPUT`, `INPUT` oder `INPUT_PULLUP`

`digitalWrite(pin, value)` … setzt eine der digitalen Pins. Dieser Pin muss vorher als `OUTPUT` definiert sein. Value: `HIGH` oder `LOW`

`boolean digitalRead(pin)` … liest den Wert eines digitalen Pins. Der Pin muss vorher als `INPUT` oder `INPUT_PULLUP` definiert sein. Diese Funktion gibt `HIGH` oder `LOW` zurück.

`delay(msec)` … hält die Programmausführung an. Die Zeit wird in Millisekunden angegeben. Beispiel: `delay(1000)` wartet 1 sec.

`tone(pin, frequency)` … produziert eine Rechteckschwingung auf einem der digitalen Ausgänge. `frequency`: Frequenz in Hz zwischen 31 und 65535.

`noTone(pin)` … beendet die Ausgabe von `tone()`

`analogWrite(pin, value)` … Produziert ein PWM-Signal an einem der digitalen Pins. Value: 0 (= dauerhaft aus) bis 255 (dauerhaft an)

`int analogRead(pin)` … liest den ADC auf dem gewählten Kanal und gibt den eingelesenen Wert zurück. Bereich: 0 bis 1023.

Teil 1: Einführung

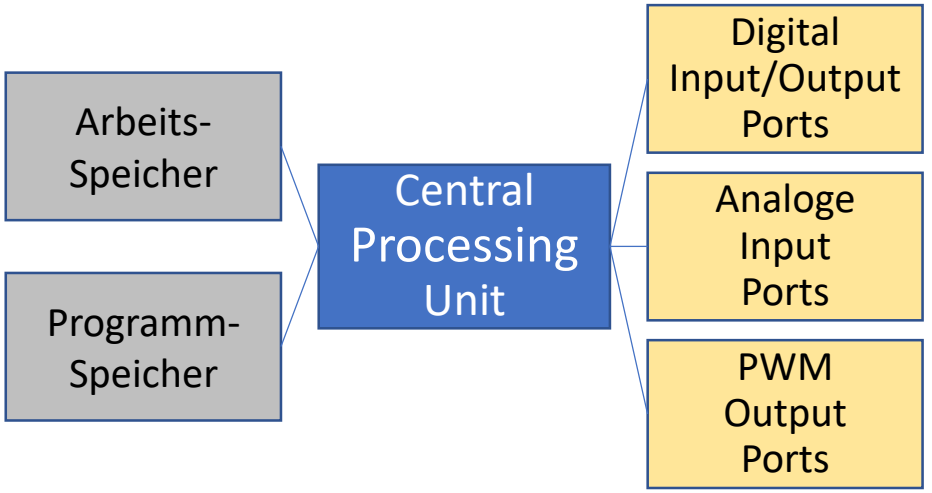
Mikrocontroller
Arduino Plattform
Programmieren

Einführung: Was ist ein Mikrocontroller?

Ein Mikrocontroller ist ein Mini-Computer auf einem Chip. Im Kern des Controllers sitzt eine Central Processing Unit (CPU), die ein Software-Programm abarbeitet und dabei über Ports elektrische Signale von außen liest und nach außen schreibt.

Mikrocontroller werden in vielen Geräten des Alltags verwendet, um sie zu steuern (Waschmaschine, Kaffeemaschine, Radio, Fotoapparat, Auto, usw.)

Mikrocontroller
ATmega328



Vereinfachtes Blockdiagramm mit den Komponenten, die wir im Schnupper-Kurs verwenden

Einführung: Was ist Arduino?

Arduino ist eine übergreifende Plattform für Mikrocontroller, die unabhängig von Herstellern oder Computersystemen ist.

Das Ziel von Arduino ist es, die Anwendung von Mikrocontrollern so einfach wie möglich zu machen. Arduino wird von vielen verschiedenen Gruppen (z.B. Techniker, Künstler, Bastler) und Organisationen (Schulen, Universitäten, Institute) verwendet und weiter entwickelt.

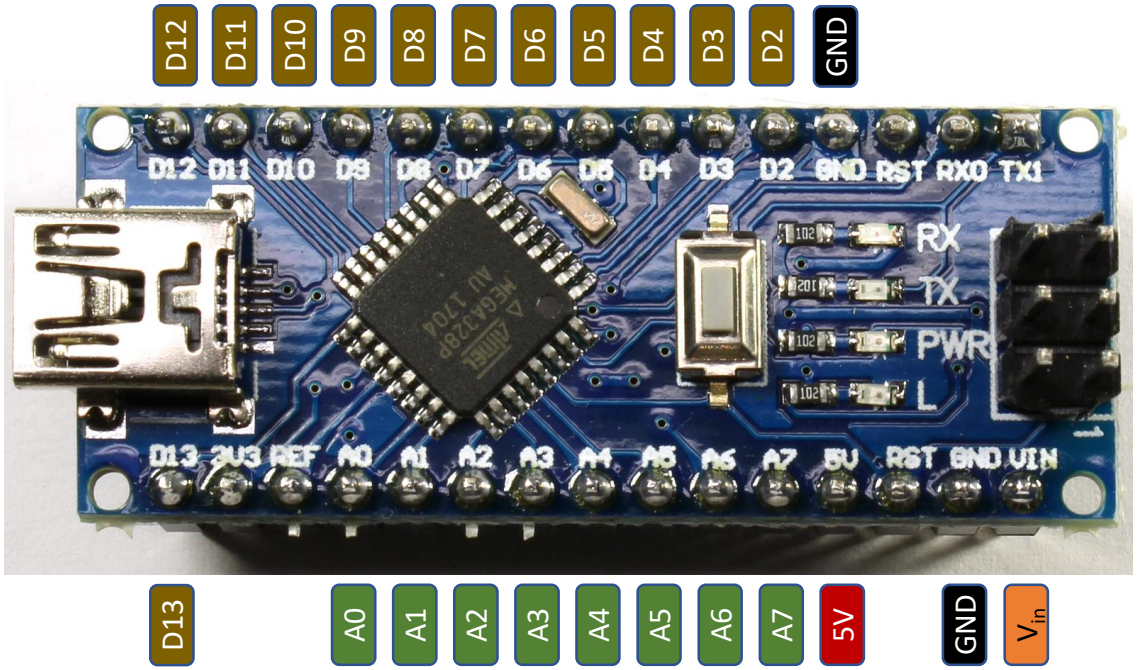
Ein **Arduino Boar**d ist eine elektronische Platine, die einen Mikrocontroller mit minimalem aber notwendigem Zubehör enthält. Wir verwenden das kleinste Arduino Board, den **Arduino Nano**

Abgesehen von der CPU (ATmega328) mit Takterzeugung enthält das Board Bauteile für die Stromversorgung, ein USB-Interface, und Taster und Leuchtdioden

Kurze Vorstellung: Arduino Nano

CPU: ATmega328 von AVR

- 8 Bit CPU mit RISC Befehlssatz
- 32 kByte Programm-Speicher im Flash-Memory
- 2 kByte Daten-Speicher im dynamischen Memory
- 1 kByte Daten-Speicher im nichtflüchtigen EEPROM
- 10 Bit Analog-Digital-Wandler (ADC)
- Taktfrequenz: 16 MHz über externen Quarz
- Optimiert für geringen Stromverbrauch



Die wichtigsten Anschlüsse des Arduino Nano auf einen Blick: Stromversorgung, digitale Ein- oder Ausgänge und analoge Eingänge

Spannungsversorgung

- USB (5V)
- Extern stabilisierte 5V-Versorgung
- Extern 7-20V an V_{in} (Spannungsregler auf der Platine)

Einführung: Was ist ein Programm?

Die CPU im Mikrocontroller braucht Anweisung, was sie tun soll. Die Liste der Anweisung ist das Programm. Arduino verwendet den Begriff **Sketch** für das Programm.

Das Programm ist vergleichbar mit einem Kochrezept. Es gibt Zutaten und Anweisungen, was mit den Zutaten gemacht werden muss.

```
void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  digitalWrite(A2, HIGH);   // set the LED on
  digitalWrite(A3, LOW);   // set the LED on
  delay(100);              // wait for a 1/10
  digitalWrite(13, LOW);    // set the LED of
  digitalWrite(A2, LOW);   // set the LED on
  digitalWrite(A3, HIGH);   // set the LED on
  delay(100);              // wait for a 1/10
  val0 = analogRead(A0);    // read the input
  val1 = analogRead(A1);    // read the input
  Serial.print(val0);
  Serial.print("\t");
  Serial.println(val1);
  analogWrite(10, 255-(val0-500)/2); // set P
  analogWrite(9, 255-(val1-500)/2); // set PW
}
```

Ein Programm ist eine Abfolge von Anweisungen, die eine nach der anderen ausgeführt wird.

Im Gegensatz zum Koch braucht die CPU eine spezielle Sprache, die **Programmiersprache C**

1. Das Öl in einer großen beschichteten Pfanne stark erhitzen.
2. Das Fleisch mit dem Eiweiß mischen, in die Pfanne geben und scharf anbraten.
3. Die Zwiebeln dazu geben und glasig anbraten.
4. Die Temperatur ein wenig herunterschalten und das restliche Gemüse bis auf die Zuckerschoten hinzugeben.
5. Ingwer und Piri-Piri untermischen.
6. Kräftig umrühren und nach 2 Minuten die Zuckerschoten hinzugeben.
7. Wenn das Gemüse fast gar ist, den Orangensaft unterrühren und ein wenig einkochen lassen.
8. Dann die Kokosmilch unterrühren, mit Salz abschmecken und servieren.

Quellen und Referenzen

- Es gibt sehr viel Dokumentation und Tutorials zu Arduino auf dem Internet.
- Zentrale Anlaufstelle ist www.arduino.cc und www.arduino.org (allerdings in Englisch)
- Ein deutsches Tutorial gibt es unter www.arduino-tutorial.de
- Referenz für die Arduino-Programmiersprache: https://www.arduino.cc/en/Reference/HomePage
- Unendlich viele Arduino-Projekte werden auf YouTube gezeigt.

Teil 2: Digitale Ein- und Ausgänge

Steckplatte und Bauteile
LEDs und Taster
Programmieren in C

Zum Aufbau der Experimente

Zu jedem Experiment gehören **zwei Komponenten**

1. Der Aufbau mit dem Bread-Board (Steck-Platte)
2. Das Programm (Sketch)

Wichtig: Drei-Schritt Regel

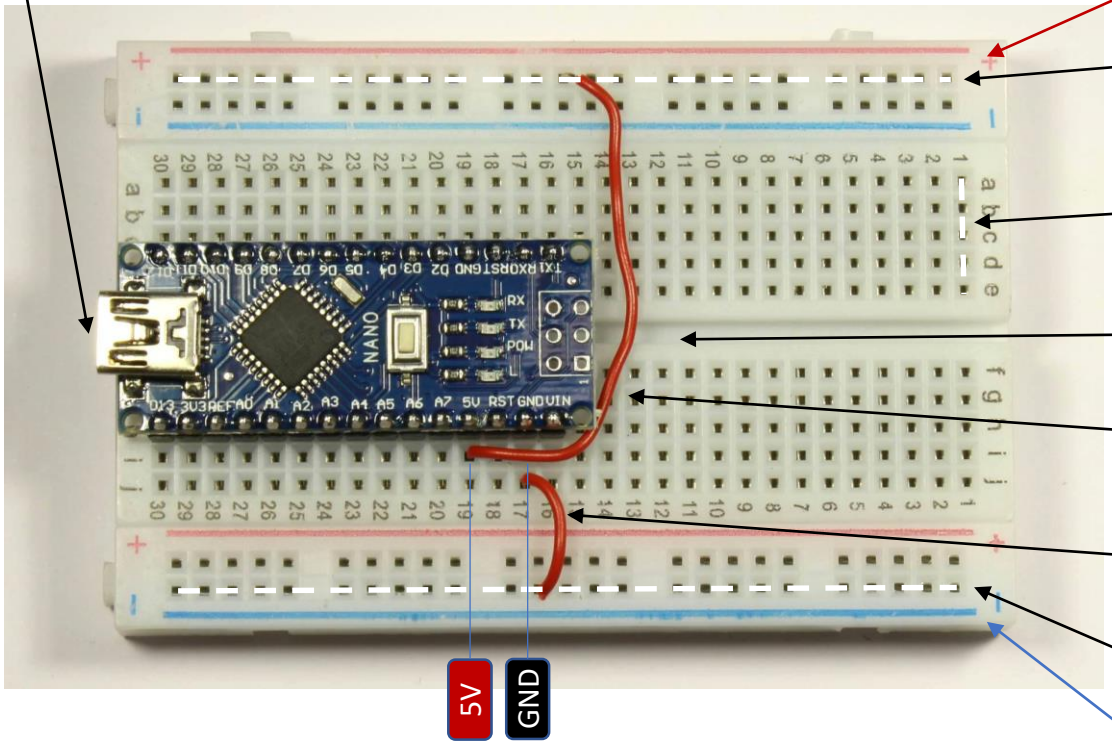
1. Das USB-Kabel immer abziehen, bevor der Aufbau verändert wird
2. Den Aufbau gründlich checken!
3. Erst danach das USB-Kabel anstecken

Vorgehensweise: Zuerst wird die Schaltung auf dem Bread-Board zusammen gebaut. Dann wird das Programm mit der Arduino-IDE geschrieben. Zum Schluss wird das USB-Kabel an den Arduino gesteckt und das Programm auf den Arduino übertragen.

Grundaufbau mit dem Bread-Board

Der Grundaufbau ist in jedem Experiment enthalten und bleibt immer gleich.

USB-Buchse für die Stromversorgung
und zum Programmieren des Mikrocontrollers



Rote Linie „+“ oben

Spannungsleiste,
durchgehend +5V

Immer 5 Löcher sind vertikal
miteinander elektrisch
verbunden

Arduino Nano ganz nach links

Draht von „5V“ zur oberen
Spannungsleiste

Draht von „GND“ zur unteren
Spannungsleiste

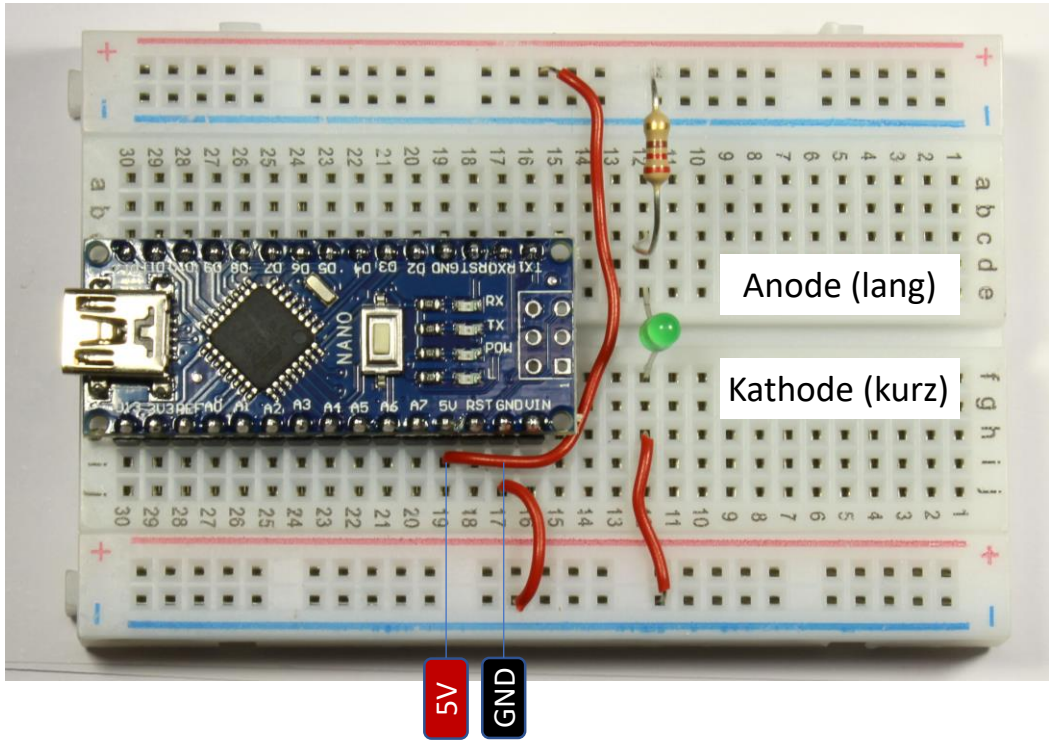
Spannungsleiste,
durchgehend GND (0V)

Blaue Linie „-“ unten

Experiment 1: Eine LED leuchtet

Experiment 1: Aufbau und Bauteile

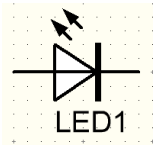
Genug der Theorie und hinein in die Praxis. Der 1. Aufbau verwendet den Arduino nur zur Stromversorgung und noch ohne Programm.



Die Drahtenden ca. 5mm abisolieren

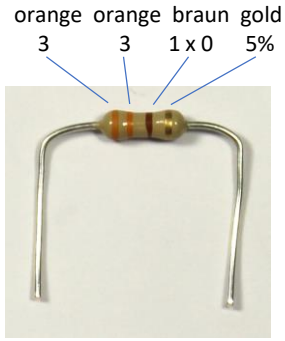
Verbindungs-Draht

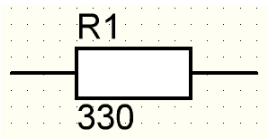
Die Drahtenden ca. 5mm abisolieren



Anode Kathode

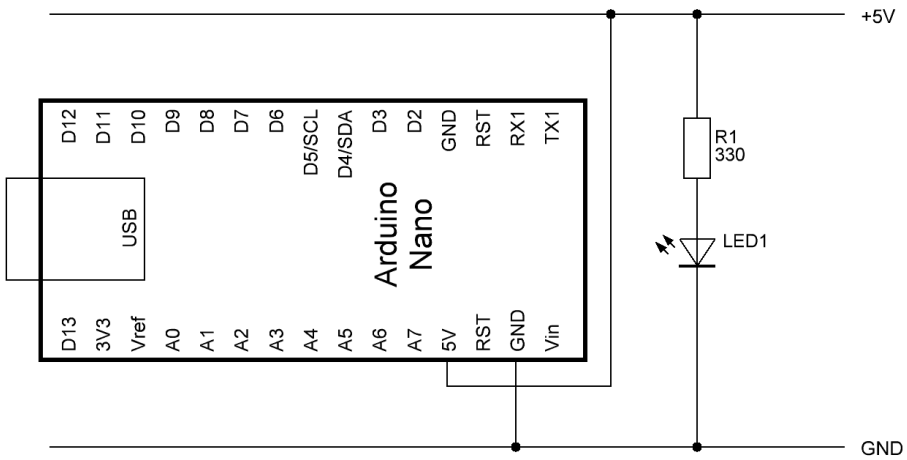
Leuchtdiode





Widerstand 330 Ohm

Experiment 1: Was passiert hier?



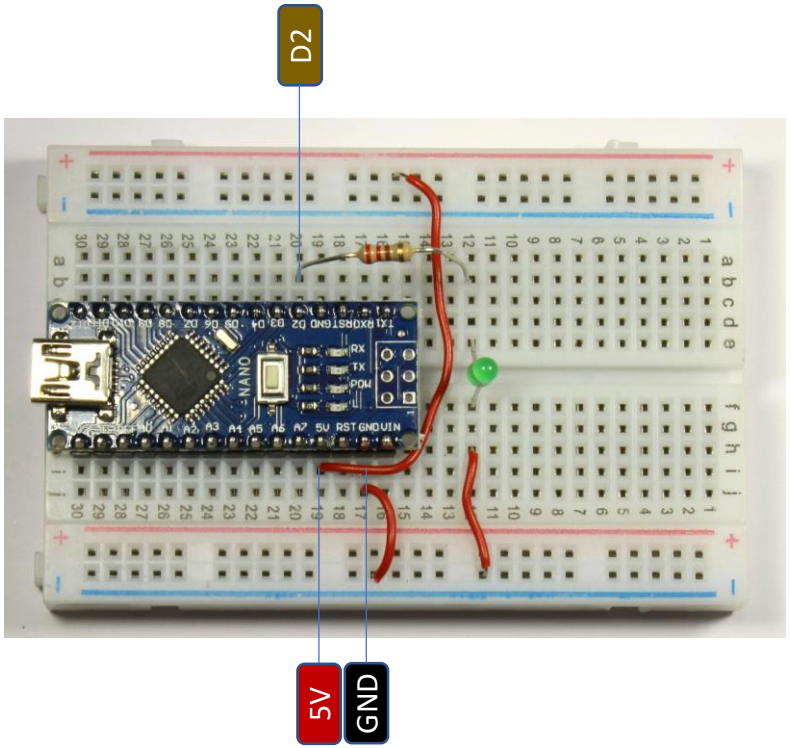
Nach dem Einstecken des USB-Anschluss leuchtet die LED. Es fließen etwa 10mA Strom von +5V durch den Widerstand und die LED. Der Widerstand begrenz den Strom, so dass die LED leuchtet aber keinen Schaden nimmt. Man nennt ihn Vorwiderstand.

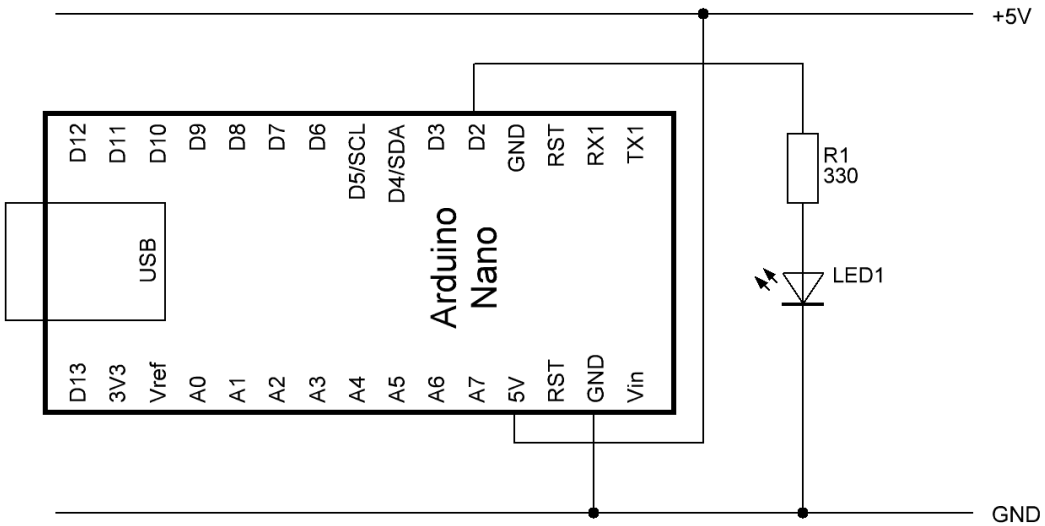
Wichtig: LEDs niemals ohne Vorwiderstand in anschließen! Die LEDs vertragen maximal 20 mA.

Der Arduino ist bisher noch nicht beteiligt und stellt hier nur die Betriebsspannung über den USB-Anschluss bereit.

Experiment 2: LED Blinker

Experiment 2: Aufbau und Schaltplan



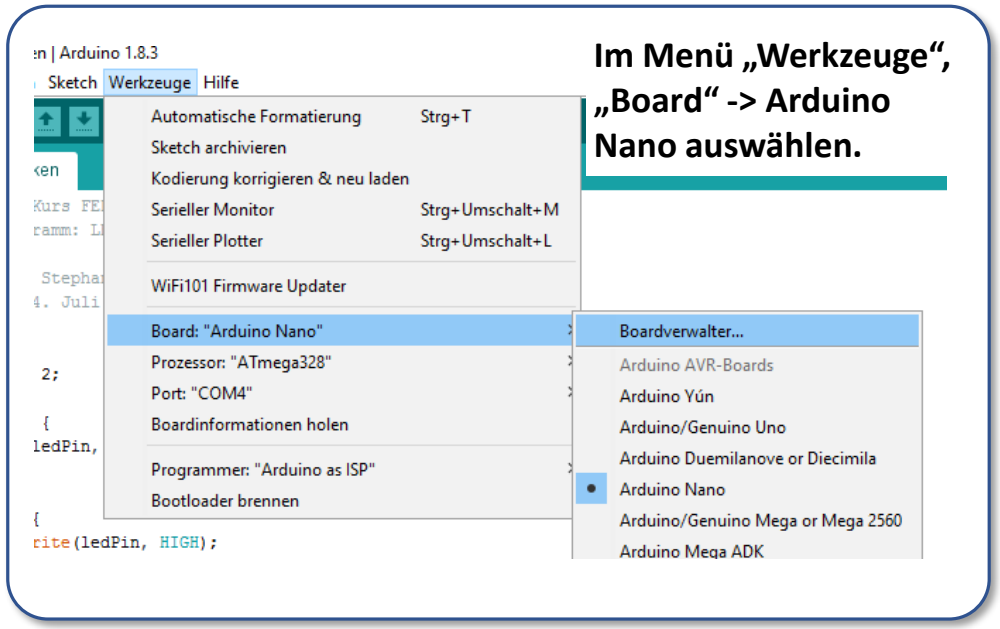


330 Ohm Farbcode:
- Orange = 3
- Orange = 3
- Braun = 1

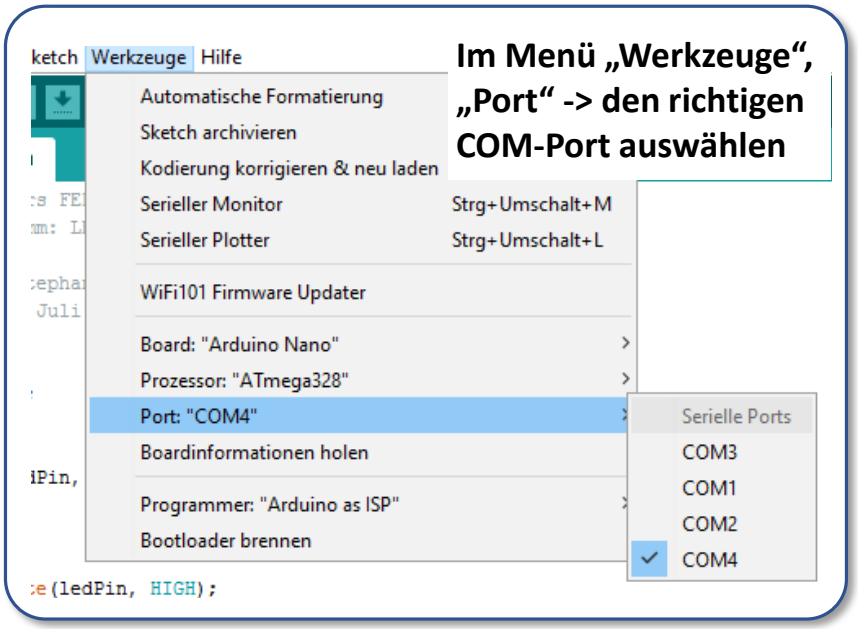
Jetzt wird programmiert: Arduino IDE

Die Arduino IDE (Integrated Development Environment) ist ein Computer-Programm, mit dem man Programme für Arduino-Boards erstellen kann. Der Programm-Text wird in der Programmiersprache C eingegeben und dann in einen Maschinencode übersetzt (= kompiliert), den der Controller verarbeiten kann. Wenn alles geklappt hat, wird das Programm auf das Board geladen. Der Controller beginnt sofort, das Programm auszuführen.

Zwei einmalige Einstellungen der Arduino IDE:



Im Menü „Werkzeuge“, „Board“ -> Arduino Nano auswählen.



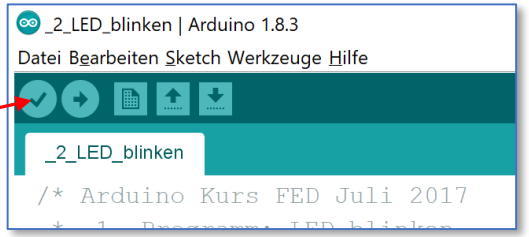
Im Menü „Werkzeuge“, „Port“ -> den richtigen COM-Port auswählen

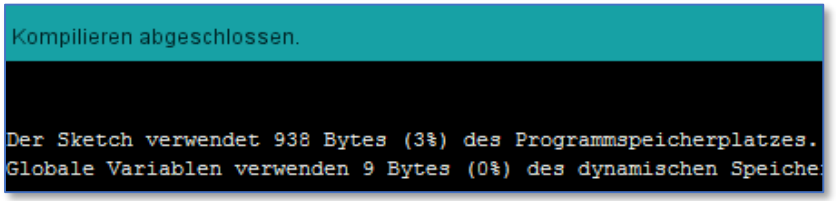
Arbeitsablauf für Programm testen und hochladen

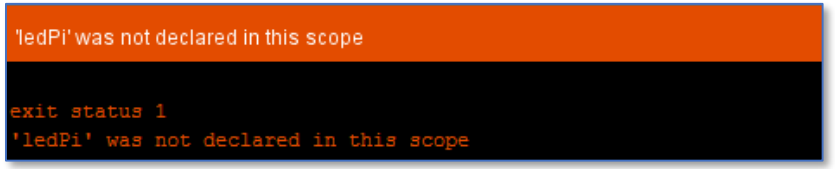
1. Wenn der Programm-Text eingegeben ist, „Überprüfen “ aufrufen
2. Im unteren Fenster zeigt die IDE das Resultat:
   - Grün: Prima, keine Fehler
   - Rot: Da stimmt etwas nicht. Fehlermeldung lesen, dann zurück zum Editor, das Programm korrigieren.
3. Wenn das Programm ohne Fehlermeldung kompiliert wurde, das Programm mit einem Klick auf „Hochladen“ auf den Arduino übertragen.
4. Wenn alles geklappt hat, kommt die Meldung „Hochladen abgeschlossen“.

Herzlichen Glückwunsch, das erste Arduino-Programm ist erstellt. Hoffentlich macht es, was es soll.

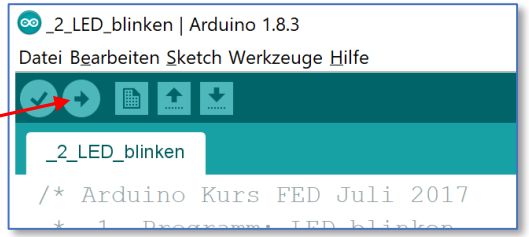
„Überprüfen“ anklicken

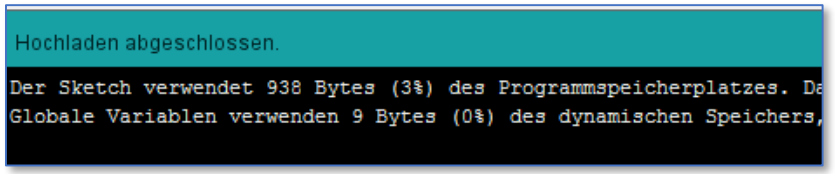






„Hochladen“ anklicken





Das Programm für Experiment 2

Alle Arduino-Programme haben (mindestens) zwei Funktionen:

- ***setup()*** für alle Einstellungen, die das Programm braucht (entspricht den Zutaten für ein Rezept)
- ***loop()*** mit den Arbeitsanweisungen (entspricht dem Arbeitsablauf eines Rezepts)

Vereinbarung, dass wir Pin 2 für die LED verwenden

Wir brauchen für diese Programm den Port *ledPin* (entspricht der Nummer 2 wie vereinbart). Der soll als Ausgang verwendet werden

Die Arbeitsweisungen sind ziemlich einfach:

- Setzte den Pegel an Port *ledPin* auf high (= 5V)
- Warte 1000 msec
- Setzte den Pegel an Port *ledPin* auf low (= 0V)
- Warte 1000 msec
- … und wieder von vorne

Programmieren in C

- Jede Anweisung wird mit einem Semikolon abgeschlossen.
- Die Funktion `pinMode(pin, value)` deklariert einen Pin am Controller als Input oder Output (oder als input-pullup).
- Die Funktion `digitalWrite(pin, status)` schaltet einen Pin am Controller auf HIGH (= 5V) oder LOW (= GND, 0V)
- Die Funktion `delay(msec)` bewirkt eine Pause in der Programmausführung
- Kommentare sind Notizen, die an den Programmierer oder andere Leser gerichtet sind. Die Kommentare werden vom Compiler ignoriert. Kommentare sind sehr hilfreich für das Verständnis der Programm-Logik und sollten häufig verwendet werden.
  - Jeder Text zwischen **/*** und ***/** wird als Kommentar gewertet. Der Kommentar kann über beliebig viele Zeile reichen.

    ```
    /* Arduino Kurs an der FES */
    ```

  - Zwei Schrägstriche // markieren den Rest der Zeile als Kommentar.

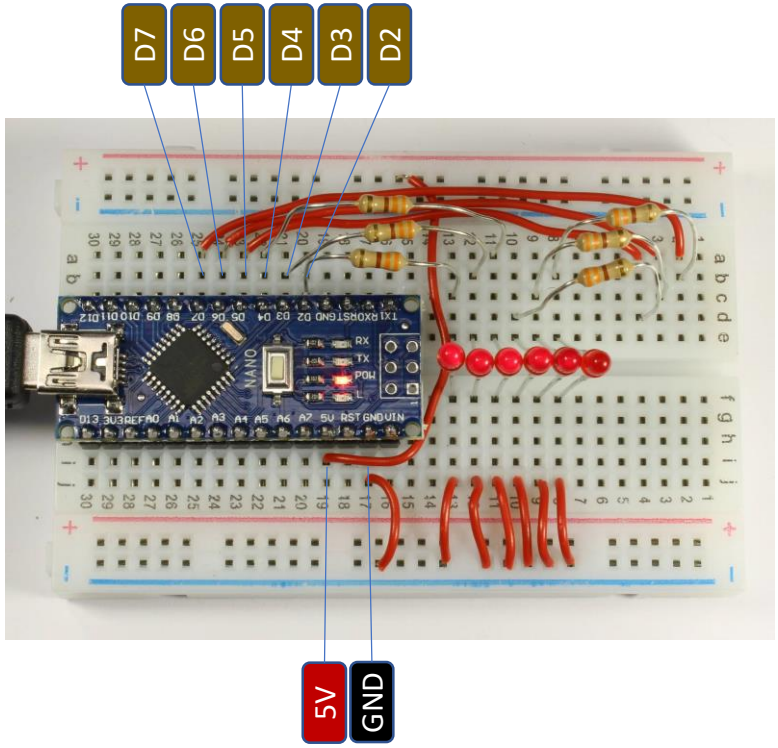
    ```
    digitalWrite(ledPin, HIGH);   // Dieser Befehl schaltet die LED ein
    ```

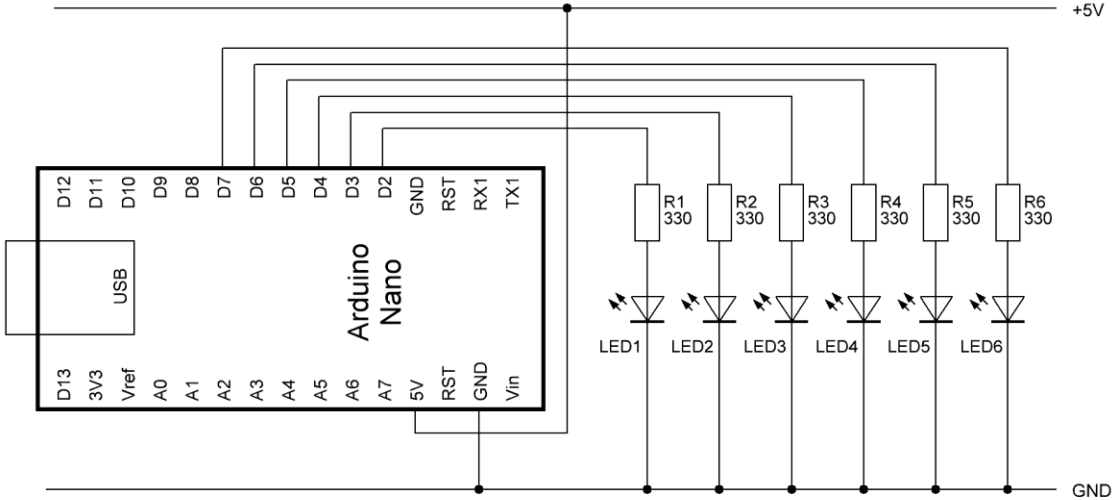
Experiment 2: Fragen und Anregungen

- Wie kann man die Geschwindigkeit des Blinkers langsamer oder schneller einstellen?
- Berechne die Blink-Frequenz. Ab welcher Frequenz kannst du das Blinken nicht mehr mit dem Auge erkennen?
- Wie kann man das Verhältnis von Ein-Zeit zu Aus-Zeit verändern? Was sind die kürzesten Blink-Phasen, die du noch sehen kannst?
- Versuche, die LED umzudrehen – Anode an GND, Kathode an den Widerstand. Was passiert?
- Was passiert, wenn du die LED mit Vorwiderstand zwischen Port und die +5V-Spannungsschine anschließt? Aber die LED niemals ohne Vorwiderstand in der Schaltung betreiben!
- Versuche, die LED an einem anderen Pin, z.B. D6 oder D7 anzuschließen. Was muss im Programm geändert werden?

Experiment 3: Lauflicht

Experiment 3: Aufbau und Schaltplan





Aus Platzgründen werden die Vorwiderstände von LED4 bis LED6 nach rechts verlegt, jeweils in eine eigene Kontaktreihe. Über Drähte werden die Kontaktreihen mit dem jeweiligen Port verbunden. **Wichtig: Jede der 5-poligen Kontaktreihen kann nur <u>eine</u> Verbindung zwischen Vorwiderstand und Draht aufnehmen!**

330 Ohm Farbcode:
- Orange = 3
- Orange = 3
- Braun = 1

Experiment 3: Programm

_3_Lauflicht

```
/* Arduino Kurs FES Juli 2017
 *  Programm: Lauflicht
 *
 *  Author: Stephan Laage-Witt
 *  Datum: 5. Juli 2017
 */

int active_pin = 2;       // Nummer der aktive LED (2 ... 7)

void setup() {
    pinMode(2, OUTPUT);   // Pin 2 bis Pin 7 werden Output
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
    pinMode(7, OUTPUT);
}

void loop() {
    digitalWrite(active_pin, LOW);    // vorherige LED ausschalten
    active_pin = active_pin + 1;      // nächste LED anwählen
    if (active_pin > 7) {             // wenn die Reihe zu Ende ist ...
      active_pin = 2;                 // ... zurück setzen auf die erste LED
    };
    digitalWrite(active_pin, HIGH);   // LED einschalten
    delay(50);                        // einen Moment warten
}
```

Programm 3a

Programmieren in C

Variablen

Die Pin-Nummer der aktiven LED wird als Variable definiert. Variablen sind vereinbarte Speicherstellen, die vom Programm gelesen und beschrieben werden können.

Variablen benötigen einen Datentyp, in diesem Fall `int`, was bedeutet, dass die Variable nur ganze Zahlen aufnehmen kann.

Andere Datentypen:

`boolean` kann nur zwei Werte annehmen: `true` oder `false`

`float` für beliebige Komma-Zahlen, z.B. 3.1415, -201.95, 12.4e-10

`char` für Buchstaben und Zeichen, z.B. `a`, `9`, `$`

Verzweigungen mit if … else

Das Schlüsselwort `if` leitet eine Abfrage ein.

Wenn das Ergebnis das Abfrage `true` ist, dann wird der folgende Programmteil abgearbeitet.

Wenn das Ergebnis der Abfrage `false` ist, dann wir der nächste Block übersprungen. Falls ein `else`-Block vorhanden ist, wird dieser abgearbeitet.

Blöcke mit { … }

Geschweifte Klammern fassen mehrere Anweisungen zu Blöcken zusammen.

Experiment 3: Fragen und Anregungen

- Wie kann man die Geschwindigkeit des Lauflichts schneller oder langsamer machen?
- Schreibe ein Programm, dass das Lauflicht von links nach rechts und wieder zurück laufen lässt.

Experiment 3: Programm-Variante mit Schleife

Lauflicht_v1

```
/* Lauflicht Version 1 */

int led_start = 2;                  // erster LED-Pin
int led_end = 7;                    // letzter LED-Pin
int active_led = 2;                 // Aktive LED

void setup() {
  for (int i = led_start; i <= led_end; ++i)
    pinMode(i, OUTPUT);             // alle Pins auf Output setzen
}

void loop() {
  digitalWrite(active_led, HIGH); // aktiven Pin einschalten
  delay(100);                       // einen Moment warten
  digitalWrite(active_led, LOW);  // aktiven Pin ausschalten
  ++active_led;                     // nächsten Pin anwählen
  if (active_led > led_end)         // am Ende der Reihe ...
    active_led = led_start;         // ... wieder von vorne
}
```

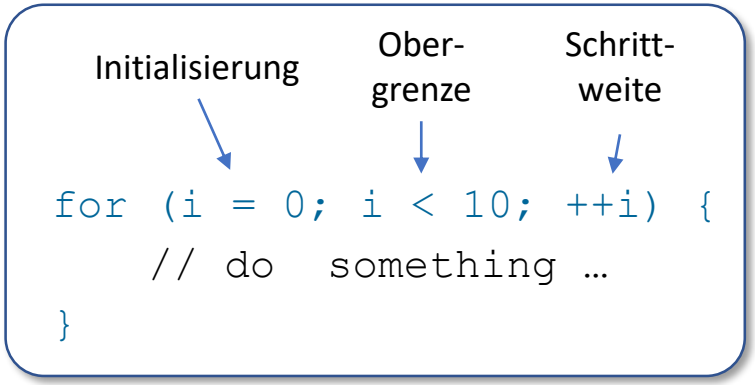
Programm 3b

Die for-Schleife in der Setup()-Funktion initialisiert alle LED-Pins.

For-Schleife

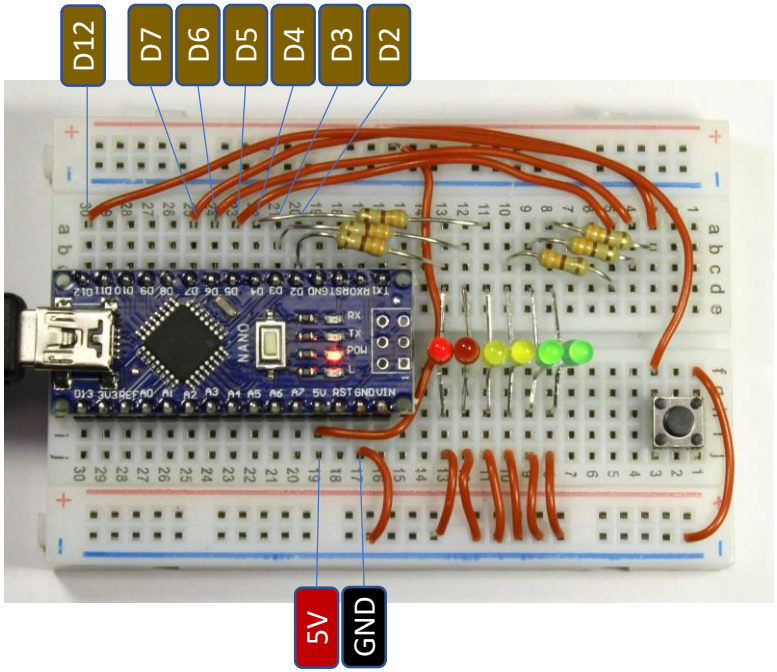
Programmierer vermeiden gerne jede Form von überflüssigem Tippen ...
Mit Schleifen geht es oft einfacher.

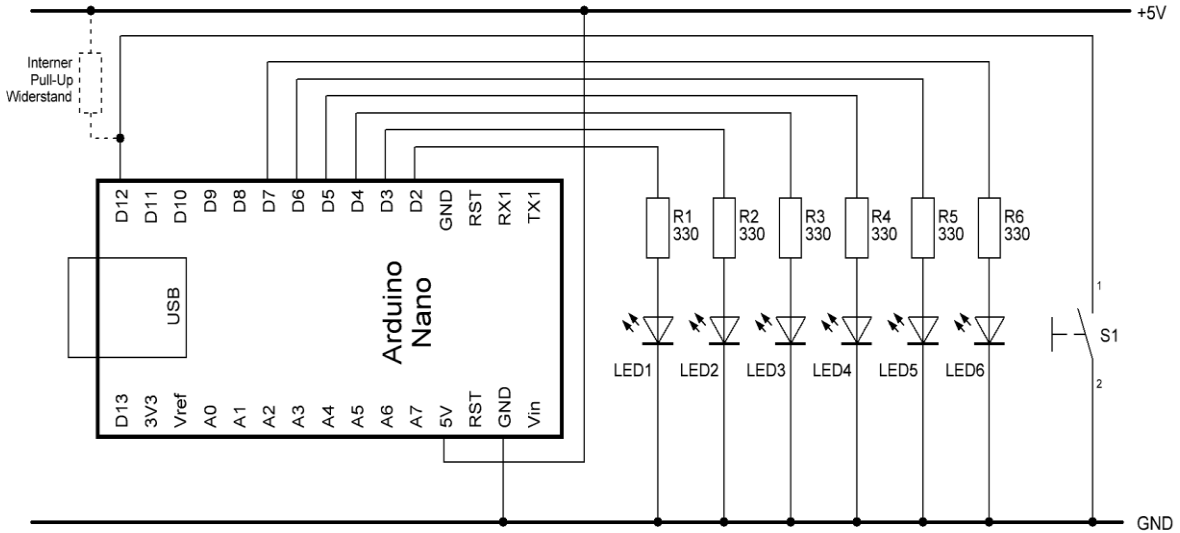
Die for-Schleife erlaubt die wiederholte Ausführung eines Blocks.



Experiment 4: Lauflicht mit Input-Taster

Experiment 4: Aufbau und Schaltplan





Pin 12 wird als digitaler Eingang verwendet. Mit dem Programm wird im Prozessor ein „Pull-Up“-Widerstand aktiviert, der die Spannung am Eingang auf `HIGH` (= 5V) setzt. Der Taster liegt an GND und zieht den Eingang auf `LOW` (= 0V).

Wichtig: Den Taster beim Einbau so drehen, dass die Längsstreifen auf der Unterseite senkrecht stehen!

330 Ohm Farbcode:
- Orange = 3
- Orange = 3
- Braun = 1

Experiment 4: Programm

_4a_Lauflicht_mit_Taster

```
/* Arduino Kurs FES Juli 2017
   Programm 4a: Lauflicht mit Taster
*/

int first_LED_pin = 2;                    // erste LED
int last_LED_pin = 7;                     // letzte LED
int button_pin = 12;                      // Taster an Pin 12
int active_LED = 2;                       // aktive LED

void setup() {
  int i;
  for (i = first_LED_pin; i <= last_LED_pin; ++i) {
    pinMode(i, OUTPUT);                   // Ausgang für die LEDs
  };
  pinMode(button_pin, INPUT_PULLUP);      // Eingang für den Taster
}

void loop() {
  if (digitalRead(button_pin) == LOW) {
    digitalWrite(active_LED, HIGH);       // aktive LED einschalten
    delay(100);                           // ... warten
    digitalWrite(active_LED, LOW);        // ... und wieder ausschalten
    ++active_LED;                         // die nächste LED ansteuern
    if (active_LED > last_LED_pin) {      // wenn das Ende der Reihe ...
      active_LED = first_LED_pin;         // erreicht ist, wieder ...
    };                                    // von vorne anfangen
  };
}
```

Programm 4a

Digitaler Eingang

Das Lauflicht läuft nur dann, wenn die Taste gedrückt ist.

In der setup()-Funktion wird Pin 12 als Eingang mit aktivem „Pull-Up“-Widerstand (`INPUT_PULLUP`) definiert.

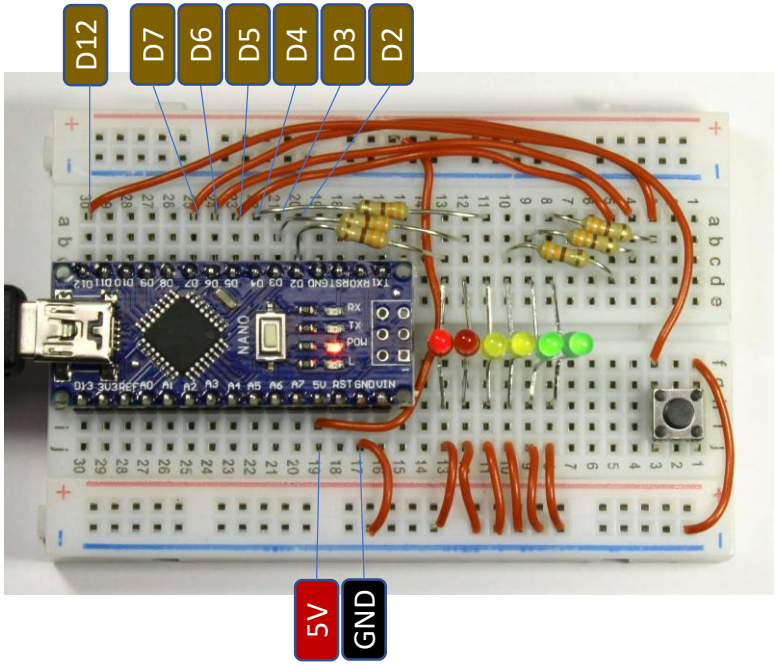
In der loop()-Funktion wird der Zustand der Taste mit `digitalRead()` abgefragt. Der Programm-Block für das Lauflicht wird nur abgearbeitet, wenn der Eingang auf `LOW` (= Taste gedrückt) steht.

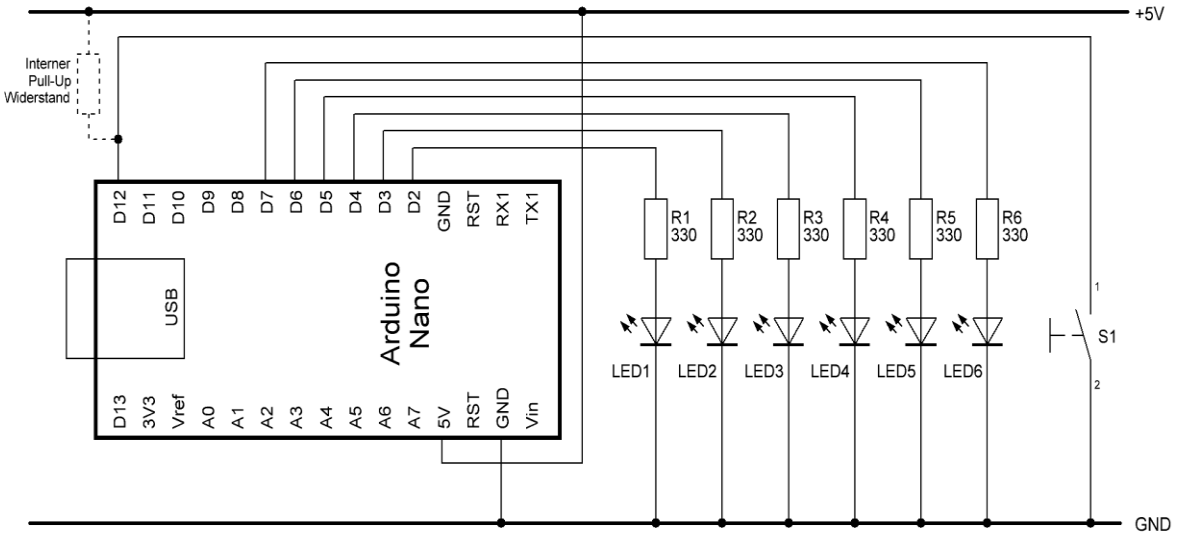
Experiment 4: Fragen und Anregungen

- Schreibe das Programm um, so dass das Lauflicht von alleine läuft, aber bei gedrücktem Taster stehen bleibt.

Experiment 5: Würfeln

Experiment 5: Aufbau und Schaltplan





Der Aufbau wird unverändert übernommen von Experiment 4.

Das Programm wird so umgebaut, dass ein Würfel entsteht. Die Taste löst einen Würfelvorgang aus, an dessen Ende die Anzahl der leuchtenden LEDs – 1 bis 6 - die Augenzahl des Würfels anzeigt. Dabei ist die Länge des Tastendrucks entscheidend für das Ergebnis. Es ist also ein Pseudo-Zufallsprozess.

330 Ohm Farbcode:
- Orange = 3
- Orange = 3
- Braun = 1

Experiment 5: Erstes Würfelprogramm

_5_Wuerfeln

```
/* Arduino Kurs FES Juli 2017
   Programm 5a: Wuerfeln
*/

int button_pin = 12;                    // Taster an Pin 12
boolean dice_rolling = true;            // zeigt an, ob der Würfel gerade rollt
int dice_score = 1;                     // aktuelle Augenzahl des Würfels
int delay_time = 5;                     // Zeit zwischen den Augenzahlen

void setup() {
  int i;
  for (i = 2; i < 8; ++i) {
    pinMode(i, OUTPUT);                   // Pin 2 bis Pin 7 werden Output
  };
  pinMode(button_pin, INPUT_PULLUP);      // Pin 12 wird Eingang für den Taster
}

void loop() {
  if (dice_rolling) {                     // Falls der Würfel rollt ...
     show_dice(dice_score);               // Zeige die aktuelle Augenzahl
     ++dice_score;                        // Erhöhe die Augenzahl
     if (dice_score > 6) dice_score = 1;  // Wenn 6 überschritten wird, wieder bei 1 anfangen
     if (digitalRead(button_pin) == HIGH) { // Wenn die Taste frei gegeben wurde ...
        delay_time = delay_time + 5;      // ... verlangsamere das Tempo
        if (delay_time > 200) {           // Wenn delay_time sehr gross wird ...
           dice_rolling = false;          // ... halte den Würfel an
        };
     };
  } else {                                // Falls der Würfel nicht rollt ...
    if (digitalRead(button_pin) == LOW) {  // ... checke, ob die Taste gedrückt ist.
      dice_rolling = true;                // In dem Fall starte den Würfel neu ...
      delay_time = 5;                     // ... und setze delay_time zurück
    };
  };
  delay(delay_time);                      // einen Moment warten
}
```

Programm 5a, Teil 1

Das Programm unterscheidet zwischen zwei Zuständen: Der Würfel rollt oder er rollt nicht. Dazu wird die boolesche Variable **`dice_rolling`** verwendet, die die Werte **`true`** oder **`false`** annehmen kann.

Wenn der Würfel rollt, wird die Augenzahl sehr schnell durchgezählt. Sobald die Taste losgelassen wird, wird die Geschwindigkeit verlangsamt, bis der Würfel nach einer Weile steht. Die Geschwindigkeit des Rollens wird mit der Variable **`delay_time`** kontrolliert.

Die Funktion **`show_dice()`** ist eine selbstgeschrieben Funtkion zur Anzeige der Augenzahl (siehe nächste Folie).

Funktion show_dice()

```
/* show_dice() ------------------------------------------------------------------------------
 *  Zeigt die aktuelle Augenzahl an
 */
void show_dice(int score) {                  // Funktion zur Anzeige der Augenzahl
  int i;
  for (i = 1; i < 7; ++i) {                  // alle LEDs, deren Nummer kleiner oder gleich ...
     if (i <= score)                         // ... der aktuellen Augenzahl ist, werden eingeschaltet
        digitalWrite(i + 1, HIGH);
     else                                    // alle anderen LEDs ...
        digitalWrite(i + 1, LOW);            // ... werden ausgeschaltet
  }
}
```

Programm 5a, Teil 2

Die Funktion **`show_dice()`** wird aus der **`loop()`**-Funktion aufgerufen und zeigt die aktuelle Augenzahl an. Dazu übergibt das aufrufende Programm die jeweilige Augenzahl als Parameter in den Klammern nach dem Funktionsnamen **`show_dice(dice_score);`**.

Entsprechend zeigt der Kopf der Funktions-Definition, dass ein Paramater erwartet wird. Der Name des Parameters heißt hier **`score`**. Wichtig ist, das die Datentypen zwischen Funktions-Definition und Aufruf gleich sind, in diesem Fall **`int`**. Der Aufruf `show_dice(dice_rolling);` würde also eine Fehlermeldung produzieren.

Die Funktion **`show_dice()`** läuft durch alle LEDs und schaltet sie ein oder aus, abhängig vom Wert.

Programmieren in C: Funktionen

Funktionen sind ausgelagerte Programmteile, die von einem übergeordneten Programm aufgerufen werden.

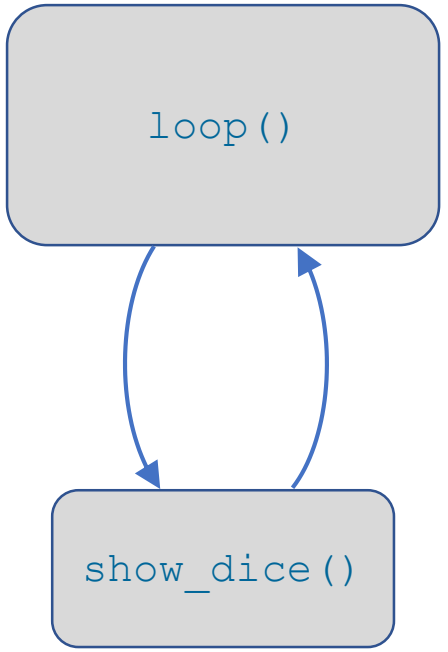
- Beispiel: Die Funktion `show_dice(int score)` zeigt die Augenzahl an. Die Funktion wird im übergeordneten Block `loop()` aufgerufen.

Was ist der Zweck von Funktionen?

- Funktionen verbessern die Übersichtlichkeit von Programmen, indem große Probleme in kleine Teilprobleme zerlegt werden.
- Funktionen können mehrfach aufgerufen werden, so dass Programmteile wieder verwendet werden.

Funktionen können über Parameter an die Bedürfnisse angepasst werden. In unserem Beispiel wird eine Augenzahl (`score`) als Parameter an die Funktion `show_dice()` übergeben.

Die Sprache C erlaubt die Übergabe von beliebig vielen Parametern. Es können auch Werte von der Funktion an das aufrufende Programm zurück gegeben werden. Beispiel dafür folgen später.

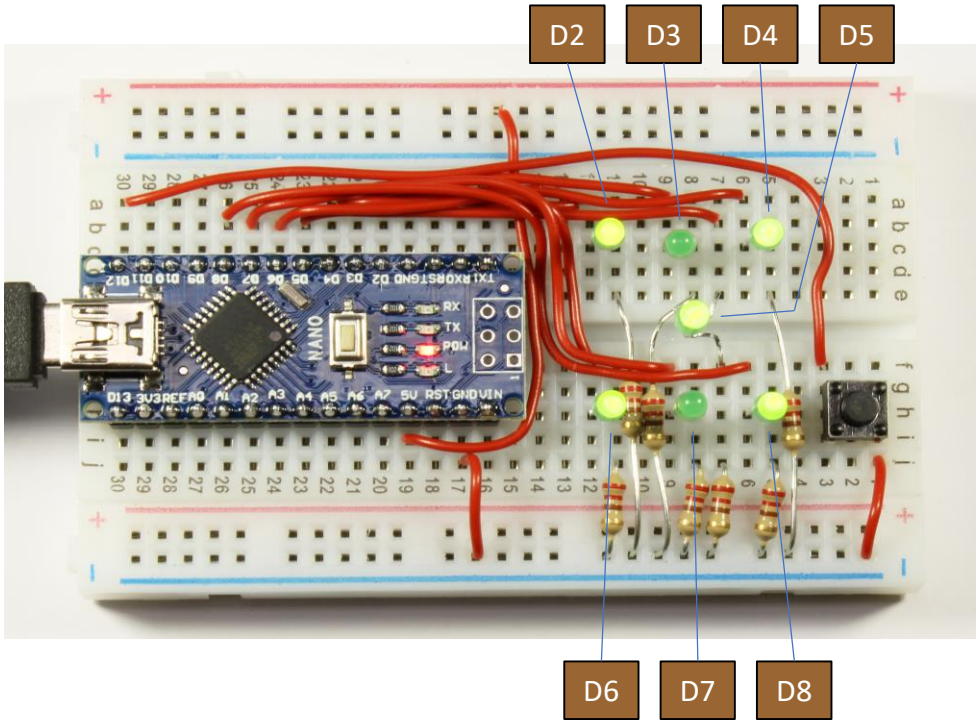


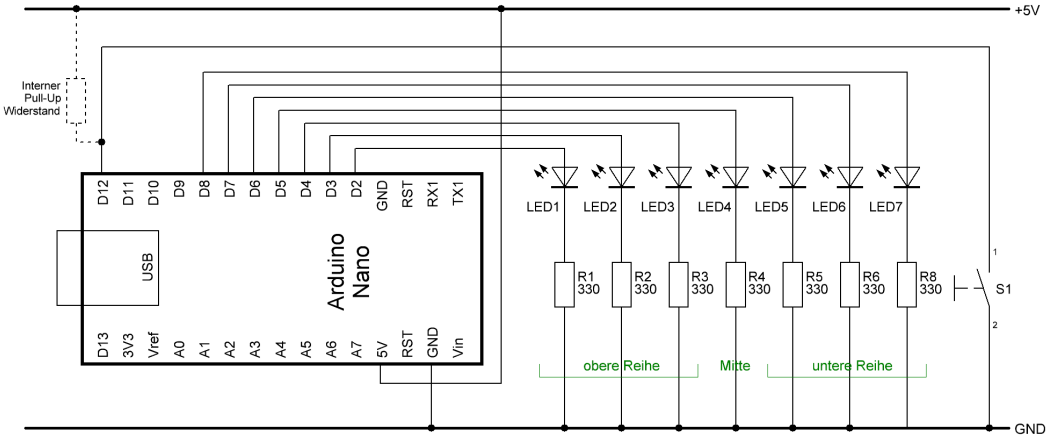
Verbesserter Würfel v2

Der Würfel funktioniert schon ganz gut. Schöner wäre es aber, die Augenzahl in einer Anordnung wie bei einem richtigen Würfel zu zeigen. Dazu muss der Aufbauabgeändert werden. Jetzt brauchen wir 7 LEDs, die über D2 bis D8 angesteuert werden.

Die Beine der mittleren LED an D5 müssen etwas zur Seite abgewinkelt werden, um die Position in der Mitte zu erreichen. Aus Platzgründen befinden sich die LED-Vorwiderstände sitzen in diesem Aufbau zwischen der LED und Ground.

Abgewinkelte Anschlussdrähte von LED4 (Port D5)





Es wird langsam eng auf dem Bread-Board. Wichtig ist, dass jede 5-polige Kontaktleiste nur eine Verbindung aufnehmen kann. Blanke Drähte dürfen sich nicht berühren.

Verbesserter Würfel: setup()-Funktion

Das Programm 5a muss an zwei Stellen verändert werden. Die erste Stelle ist einfach: Die setup()-Funktion muss jetzt die Ports D2 bis D8 für die 7 LEDs als Output initialisieren. Zur Vereinfachung benutzen wir die Variable `first_led_pin` und `last_led_pin`, um die Ports zu identifizieren.

_5b_Wuerfeln

```
/* Arduino Kurs FES Juli 2017
   Programm 5b: verbeserter Wuerfel
*/

int first_led_pin = 2;                  // erste LED
int last_led_pin = 8;                   // letzte LED
int button_pin = 12;                    // Taster an Pin 12
boolean dice_rolling = true;            // zeigt an, ob der Würfel gerade rollt
int dice_score = 1;                     // aktuelle Augenzahl des Würfels
int delay_time = 5;                     // Zeit zwischen den Augenzahlen

void setup() {
  int i;
  for (i = first_led_pin; i <= last_led_pin; ++i) {
    pinMode(i, OUTPUT);                    // Pin 2 bis Pin 7 werden Output
  };
  pinMode(button_pin, INPUT_PULLUP);       // Pin 12 wird Eingang für den Taster
}
```

Programm 5b, setup()-Funktion

Die loop()-Funktion bleibt unverändert zur vorherigen Version

Verbesserter Würfel: show_dice()-Funktion

Die zweite Veränderung bezieht sich auf die show_dice()-Funktion. Das Problem ist, dass zu jeder Augenzahl ein eigenes Leuchtmuster eghört. Wir müssen dem Prozessor mitteilen, welche LEDs er bei welcher Augenzahl aktivieren soll. Dazu benutzen wir eine Tabelle. In der Programmiersprache C ist das ein Array.

```
/* show_dice() ------------------------------------------------------------
 *  Zeigt die aktuelle Augenzahl an
 */
void show_dice(int score) {                  // Funktion zur Anzeige der Augenzahl
  boolean score_table[] = {
    // oben ---------------  Mitte  unten --------------
    // links, Mitte, rechts, Mitte, links, Mitte, rechts
       LOW,   LOW,   LOW,    HIGH,  LOW,   LOW,   LOW,        // 1
       HIGH,  LOW,   LOW,    LOW,   LOW,   LOW,   HIGH,       // 2
       HIGH,  LOW,   LOW,    HIGH,  LOW,   LOW,   HIGH,       // 3
       HIGH,  LOW,   HIGH,   LOW,   HIGH,  LOW,   HIGH,       // 4
       HIGH,  LOW,   HIGH,   HIGH,  HIGH,  LOW,   HIGH,       // 5
       HIGH,  HIGH,  HIGH,   LOW,   HIGH,  HIGH,  HIGH,       // 6
  };
  int i;

  for (i = 0; i < 8; ++i) {
    digitalWrite(first_led_pin + i, score_table[(score - 1) * 7 + i]);
  }
}
```

Programm 5b, show_dice()-Funktion

Die eckigen Klammern `[]` hinter dem Variablen-Namen zeigen an, dass es ein Array gebraucht wird. Das Array wird dann mit den Daten zwischen den geschweiften Klammern `{…}` initialisiert.

Der Index für Arrays startet immer bei 0. Mit der Berechnung `(score -1) * 7` wird die passende Zeile zur Augenzahl ausgewählt.

Programmieren in C: Arrays

Arrays

Ein Array ist ein Speicherbereich, in dem eine Reihe von Werten abgelegt wird. Arrays werden durch rechteckige Klammern `[]` angezeigt. Bei der Deklaration eines Arrays wird in der Klammer die Anzahl der Werte im Array angegeben, wodurch der Compiler den zugehörigen Speicherbedarf reserviert.

```
int temperatur[31]; // Tägliche Höchsttemperaturen im Monat May
```

Alternativ kann man die initialen Werte des Arrays in geschweiften Klammern `{…}` angeben. In dem Fall berechnet der Compiler den notwendigen Speicherplatz aus der Anzahl der Werte in den geschweiften Klammern. Als Beispiel dient die show_dice()-Funktion.

Nach der Deklaration können die einzelnen Werte im Array mit einem Index in der rechteckigen Klammer aufgerufen werden. Das Array beginnt immer mit dem Index 0.

```
temperatur[0] = 24;     // am 1. Mai war die Höchsttemperatur 24 Grad
temperatur[1] = 18;     // am 2. Mai war die Höchsttemperatur 18 Grad
…
temperatur[30] = 26;    // am 31. Mai war die Höchsttemperatur 26 Grad
```

Der Mittelwert über alle Temperaturwerte im Mai ist leicht zu berechnen:

```
summe = 0;
for (i = 0; i < 31; ++i)
   summe = summe + temperatur[i];
mittelwert = summe / 31;
```

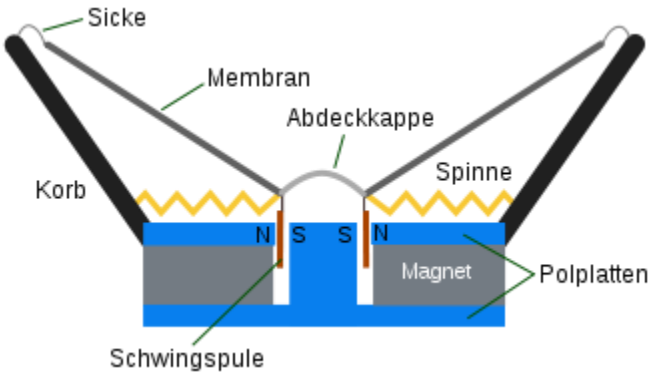
Vorsicht: Der Index darf niemals den definierten Bereich für ein Array übersteigen. In diesem Beispiel ist der höchste erlaubte Index 30, da der Indexbereich bei 0 anfängt. Zugriffe auf Daten außerhalb des Index-Bereiches sind häufige Fehler.

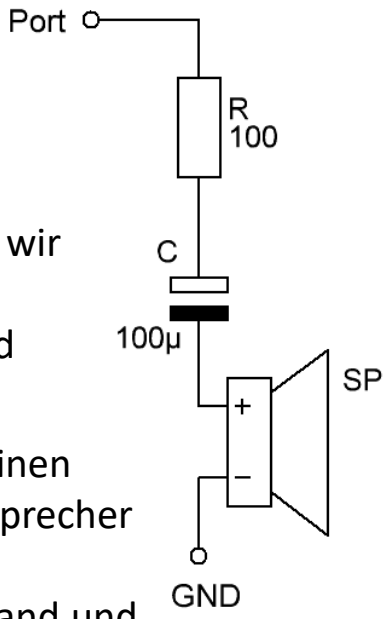
Teil 3: Frequenzen erzeugen, Pulse-Weiten-Modulation

Der Lautsprecher
Töne aus dem Lautsprecher
Mini-Orgel
PWM-Leistungsregelung von einer LED

Der Lautsprecher

Der Lautsprecher wandelt elektrische Impulse in Schall. Unter der Membran befindet sich eine Spule, die direkt in einen Permanentmagneten hineinragt. Wenn Strom durch die Spule fließt, dann erzeugt die Spule ein Magnetfeld, das sie je nach Flussrichtung entweder in den Magneten hinein zieht oder aus dem Magneten herausdrückt. Diese Bewegung überträgt sich auf die Membran und ist als Schall hörbar.





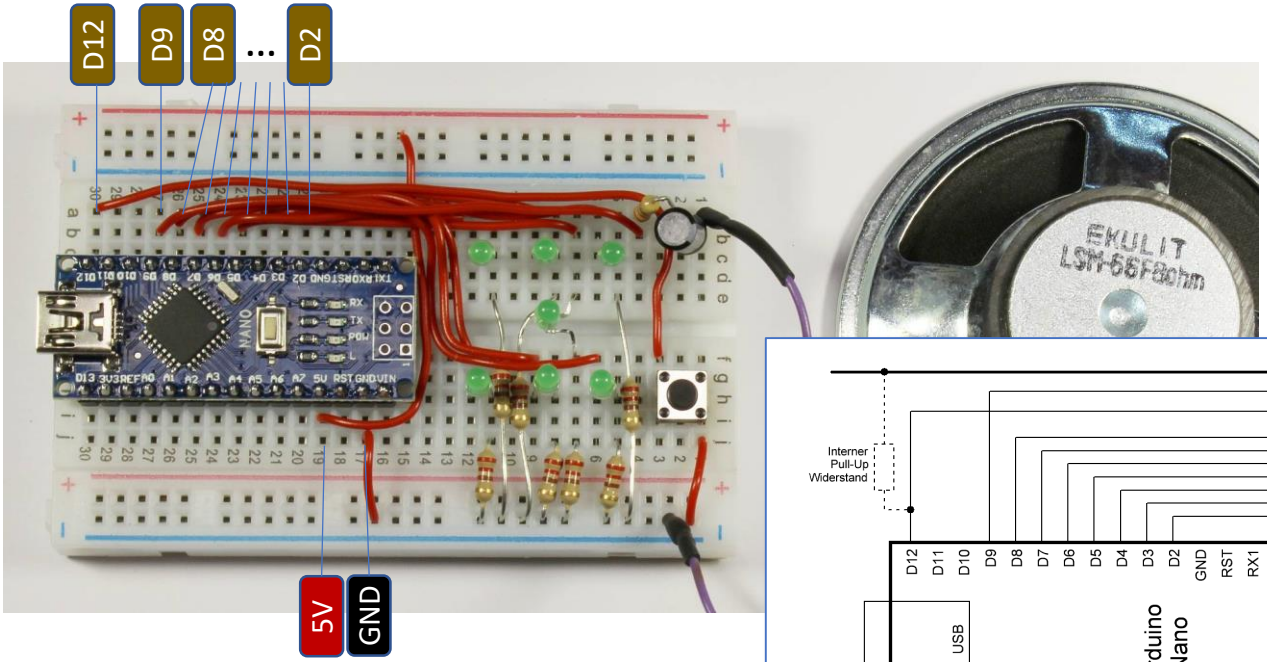
Für unsere Experimente verwenden wir den Lautsprecher immer in Reihenschaltung mit einem Widerstand und einem Kondensator.

Die Schwingspule hat einen sehr kleinen Eigenwiderstand (8 Ohm). Der Lautsprecher und der Prozessor-Port würden bei direkten Anschluss ohne Vorwiderstand und Kondensator Schaden nehmen.

Experiment 6: Würfeln mit Lautsprecher

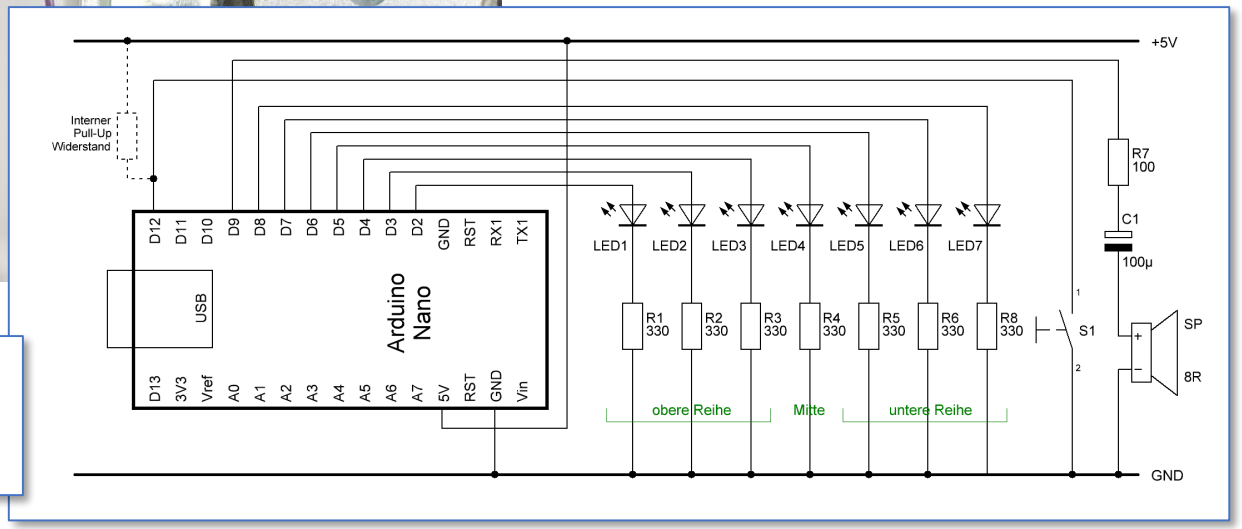
Experiment 6: Schaltplan

Die Würfelschaltung soll erweitert werden mit dem Lautsprecher, um Tick-Geräusche für den rollenden Würfel zu produzieren. Dazu verwenden wir die Reihenschaltung aus Kondensator, Widerstand und Lautsprecher an Port 9.



100 Ohm Farbcode:
- Braun = 1
- Schwarz = 0
- Braun = 1

330 Ohm Farbcode:
- Orange = 3
- Orange = 3
- Braun = 1



Experiment 6: setup()-Funktion

Im Programm gibt es nur zwei kleine Änderungen.

In der setup()-Funktion muss jetzt auch der Lautsprecher-Pin als Ausgang definiert werden.

_6_Wuerfeln_mit_Lautsprecher.ino §

```
/* Arduino Kurs FES Juli 2017
   Programm 6: Wuerfel mit Lautsprecher
*/

int first_led_pin = 2;                  // erste LED
int last_led_pin = 8;                   // letzte LED
int speaker_pin = 9;                    // Lautsprecher-Anschluss
int button_pin = 12;                    // Taster an Pin 12
boolean dice_rolling = true;            // zeigt an, ob der Würfel gerade rollt
int dice_score = 1;                     // aktuelle Augenzahl des Würfels
int delay_time = 5;                     // Zeit zwischen den Augenzahlen

void setup() {
  int i;
  for (i = first_led_pin; i <= last_led_pin; ++i) {
    pinMode(i, OUTPUT);                 // Pin 2 bis Pin 7 werden Output
  };
  pinMode(speaker_pin, OUTPUT);
  pinMode(button_pin, INPUT_PULLUP);    // Pin 12 wird Eingang für den Taster
}
```

Programm 6, setup()-Funktion

Zusätzlichen Anweisungen für den Lautsprecher.

Experiment 6: show_dice ()-Funktion

In der show_dice()Funktion wird jetzt bei jedem Aufruf der Lautsprecher-Port einmal umgeschaltet. Jede Umschaltung erzeigt einen „Klick“ im Lautsprecher

```
/* show_dice() ------------------------------------------------------------------------
 *  Zeigt die aktuelle Augenzahl an
 */
void show_dice(int score) {                 // Funktion zur Anzeige der Augenzahl
  boolean score_table[] = {
    // oben ---------------  Mitte  unten --------------
    // links, Mitte, rechts, Mitte, links, Mitte, rechts
       LOW,   LOW,   LOW,    HIGH,  LOW,   LOW,   LOW,        // 1
       HIGH,  LOW,   LOW,    LOW,   LOW,   LOW,   HIGH,       // 2
       HIGH,  LOW,   LOW,    HIGH,  LOW,   LOW,   HIGH,       // 3
       HIGH,  LOW,   HIGH,   LOW,   HIGH,  LOW,   HIGH,       // 4
       HIGH,  LOW,   HIGH,   HIGH,  HIGH,  LOW,   HIGH,       // 5
       HIGH,  HIGH,  HIGH,   LOW,   HIGH,  HIGH,  HIGH,       // 6
  };
  int i;

  for (i = 0; i < 8; ++i) {
    digitalWrite(first_led_pin + i, score_table[(score - 1) * 7 + i]);
  };
  digitalWrite(speaker_pin, !digitalRead(speaker_pin));   // Lautsprecher klicken lassen
}
```

Programm 6, show_dice()-Funktion

Zusätzlichen Anweisung für den Lautsprecher.

Programmieren in C

Ports umschalten mit dem NOT-Operator

Jeder Aufruf von show_dice() schaltet den Lautsprecher-Pin um, entweder von `HIGH` nach `LOW` oder von `LOW` nach `HIGH`. Das könnte man mit einer if-Abfrage implementieren, z.B.

```
if (digitalRead(speaker_pin) == LOW)
   digitalWrite(speaker_pin, HIGH);
else
   digitalWrite(speaker_pin, LOW);
```

Einfacher geht es mit dem NOT-Operator. Der NOT-Operator hat das Symbol **!** und verwandelt boolesche Werte (`false`, `true`) in das Gegenteil.

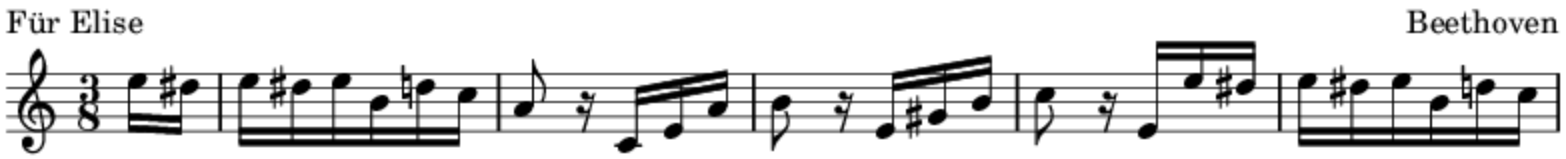
`!true` ergibt `false`, und `!false` ergibt `true`. Entsprechend verwandelt der NOT-Operator `HIGH` in `LOW`, und umgekehrt.

Die Anweisung

```
digitalWrite(speaker_pin, !digitalRead(speaker_pin));
```

schaltet den Lautsprecher-Pin aus der jeweiligen Position in das Gegenteil um.

Experiment 7: Melodie



Frequenzen erzeugen mit `tone()` und `noTone()`

Im vorherigen Beispiel haben wir Geräusche erzeugt, indem ein Portausgang an- und ausgeschaltet wurde. Diese Methode ist gut für einzelne Klicks. Um aber einen länger anhaltenden Ton zu produzieren, bietet Arduino die Funktionen `tone()` und `noTone()`. Nach einem einmaligen Aufruf schaltet der Prozessor den Ausgang ständig mit einer vorgegebenen Frequenz zwischen `HIGH` und `LOW`, ohne dass weitere Programmschritte nötig sind.

- Die Funktion `tone(pin, frequency)` erzeugt eine Rechteckschwingung an einem der digital Ports mit der gewünschten Frequenz.
- Die Funktion `noTone(pin)` schaltet die Frequenz wieder aus.

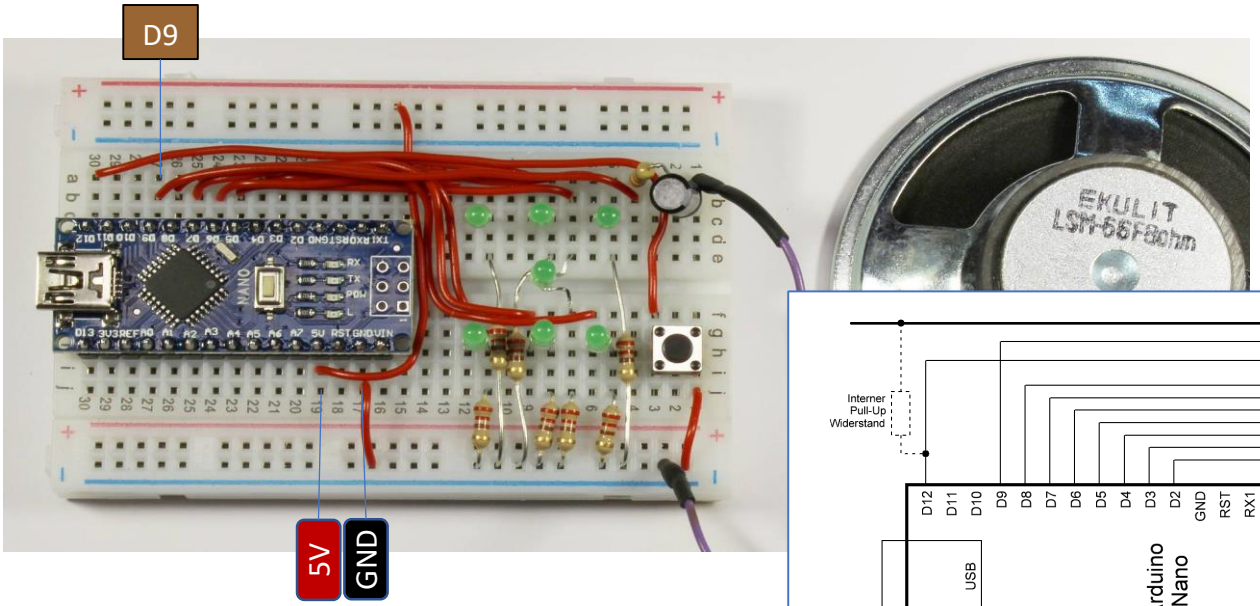
Beispiel: Dieses Programmsegment erzeugt einen Ton mit 440 Hz (Kammer-Ton A) an Pin 9 für 500 msec:

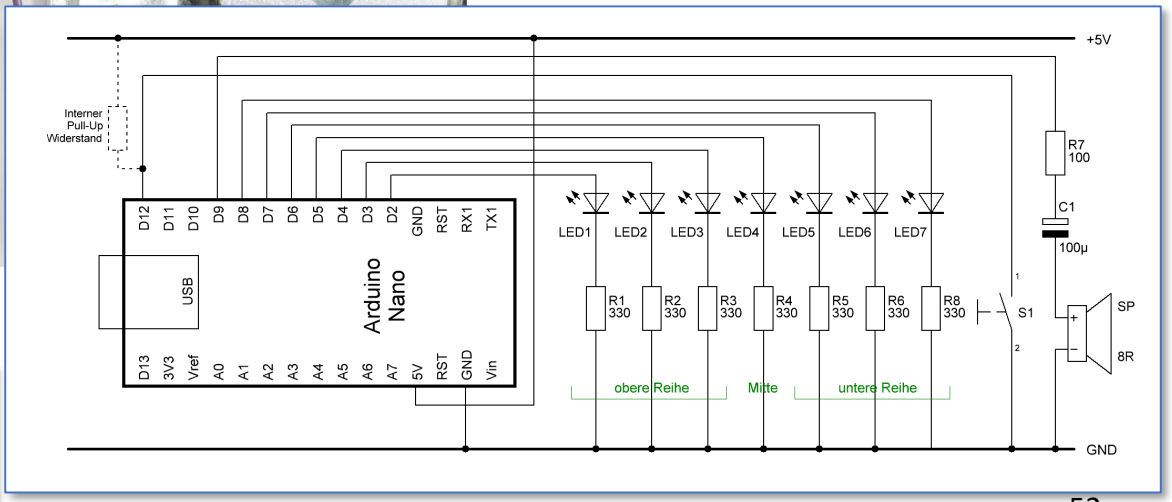
```
tone(9, 440);
delay(500);
noTone(9);
```

Experiment 7: Aufbau und Schaltplan

Für dieses Experiment brauchen wir den Taster an Pin 12 und den Lautsprecher mit Widerstand und Kondensator an Pin9.

Die LEDs on den anderen Pins können auf dem Bread-Board bleiben, werden hier aber nicht eingesetzt.





330 Ohm Farbcode:

- Orange = 3
- Orange = 3
- Braun = 1

Experiment 7: Programm

_8_Melodie §

```
/* Arduino Kurs FES Juli 2017
   Programm: Melodie
*/

// Frequenzen der Töne von 2 Oktaven
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494

int speaker_pin = 9;                    // Laustprecher an Pin 9
int button_pin = 12;                    // Taster an Pin 12

void setup() {
  pinMode(speaker_pin, OUTPUT);         // Lautsprecher am Ausgang
  pinMode(button_pin, INPUT_PULLUP);    // Taster am Eingang
}
```

```
void loop() {
  if (digitalRead(button_pin) == LOW) {     // warte auf einen Tastendruck
    play_melody(2500);                      // dann spiel die Melodie
  };
}

/* play_melody(): Funktion zum Abspielen eines Songs */
void play_melody(int song_speed) {
  int melody_tone[] = {                     // Definition der Melodie-Töne
    NOTE_E4, NOTE_DS4,                                          // Auftakt
    NOTE_E4, NOTE_DS4, NOTE_E4, NOTE_B3, NOTE_D4, NOTE_C4,      // Takt 1
    NOTE_A3, 0, NOTE_C3, NOTE_E3, NOTE_A3,                      // Takt 2
    NOTE_B3, 0, NOTE_E3, NOTE_GS3, NOTE_B3,                     // Takt 3
    NOTE_C4, 0, NOTE_C4, NOTE_A3 };                             // letzter Takt
  int melody_duration[] = {                 // Definition der Ton-Längen
    16, 16,                                                     // Auftakt
    16, 16, 16, 16, 16, 16,                                     // Takt 1
    8, 16, 16, 16, 16,                                          // Takt 2
    8, 16, 16, 16, 16,                                          // Takt 3
    8, 8, 16, 8};                                               // letzter Takt

  for (int i = 0; i < 22; ++i) {            // Schleife durch alle Melodie-Töne
    if (melody_tone[i] > 0)                 // 0 bedeutet Pause, alles andere sind Töne
      tone(speaker_pin, melody_tone[i]);    // starte den Ton
    delay(song_speed / melody_duration[i]); // warte die Länge der Note ab
    noTone(speaker_pin);                    // stopp die Ausgabe am Lautsprecher
    delay(song_speed / 100);                // mach eine kurze Pause vor dem nächsten Ton
  }
}
```

play_melody()-Funktion

Das Programm 7 enthält eine sehr kurze loop()-Funktion. Dort wird auf einen Tastendruck gewartet und dann die Funktion play_melody() aufgerufen, die eine Melodie abspielt.

play_melody() verwendet zwei Arrays: `melody_tune[]` enthält die Tonhöhen, und `melody_duration[]` die Tonlängen. Die Daten in den Arrays werden wieder mit geschweiften Klammern initialisiert. Um die Übersicht zu behalten, bekommt dabei jeder Takt aus dem Notenblatt eine eigene Zeile.

play_melody() hat einen Parameter `song_speed`, der die grundsätzliche Geschwindigkeit definiert. Der Wert von song_speed ist die Anzahl msec, die eine ganze Note ausmachen. Größere Werte bedeuten also ein langsameres Abspielen.

Nach diesen Deklarationen läuft play_melody() durch die beiden Arrays und spielt die Töne mit der Arduino-Funktion tone() nacheinander ab.

Programmieren in C: #define und #include

#define

In C benutzt man `#define`-Statements, um konstante Werte zu definieren. Die Liste der `#define`s am Anfang vom Programm bestimmt für jeden Ton die zugehörige Frequenz.

#include

Mit einem `#include`-Statement kann man eine externe Datei in die Kompilierung mit einschließen. Das ist nützlich, um häufig verwendete Definition auszulagern und hilft, die Übersichtlichkeit zu verbessern. Include-Dateien haben immer die Endung *.h*

In Melody v2 werden die Definitionen der Tonhöhen in eine include-Datei *pitches.h* ausgelagert. Diese Datei wird mit dem Statement

```
#include „pitches.h“
```

in das Programm eingebunden.

pitches.h

```
/*************************************************
 * Public Constants
 *************************************************/

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
```

Achtung: `#define` und `#include` sind Präprozessor-Statements, die nicht mit einem Semikolon abgeschlossen werden dürfen.

Melody v2: Hauptprogramm

_7b_Melodie_v2 | pitches.h

```
/* Arduino Kurs FES Juli 2017
   Programm: Melodie v2
*/

#include "pitches.h"                        // Definitionen der Tonhöhen

// Anschluss Pins
int speaker_pin = 9;                        // Laustprecher an Pin 9
int button_pin = 12;                        // Taster an Pin 12

void setup() {
  pinMode(speaker_pin, OUTPUT);             // Lautsprecher am Ausgang
  pinMode(button_pin, INPUT_PULLUP);        // Taster am Eingang
}

void loop() {
  play_melody_1(2500);                      // spiele Melodie 1
  while (digitalRead(button_pin) == HIGH);  // warte auf einen Tastendruck
  play_melody_2(1500);                      // dann spiel Melodie 2
  while (digitalRead(button_pin) == HIGH);  // Warte auf einen tastendruck
}
```

Version 2 des Melodie-Programms enthält zwei Melodien.

Die Datei *pitches.h* ist mit einem #include-Statement eingebunden.

Das Hauptprogramm wartet auf einen Tastendruck und spielt dann erst die eine und dann die andere Melodie ab.

Jede Melodie ist in eine eigene Funktion verpackt: `play_melody_1()` und `play_melody_2()`. In den beiden Funktionen sind die Tonhöhen und Tonlängen abgespeichert (siehe nächste Seite).

Melody v2: Melody-Funktionen

```
/* play_melody_1: Funktion zum Abspielen der ersten Melodie */
void play_melody_1(int song_speed) {
  int melody_tone[22] = {                    // Melodie-Töne
    NOTE_E4, NOTE_DS4,                       // Auftakt
    NOTE_E4, NOTE_DS4, NOTE_E4, NOTE_B3, NOTE_D4, NOTE_C4, // Takt 1
    NOTE_A3, 0, NOTE_C3, NOTE_E3, NOTE_A3,   // Takt 2
    NOTE_B3, 0, NOTE_E3, NOTE_GS3, NOTE_B3,  // Takt 3
    NOTE_C4, 0, NOTE_C4, NOTE_A3 };          // letzter Takt
  int melody_duration[22] = {   // Ton-Längen
    16, 16,          16, 16, 16, 16, 16, 16,  // 4 für vi
    8, 16, 16, 16, 16,    8, 16, 16, 16, 16,  // Takt 2 u
    8, 8, 16, 8};                            // letzter

  for (int i = 0; i < 22; ++i) {     // Schleife durch al
    if (melody_tone[i] > 0)                  // 0 bedeut
      tone(speaker_pin, melody_tone[i]);     // starte d
    delay(song_speed / melody_duration[i]);  // warte di
    noTone(speaker_pin);                     // stopp di
    delay(song_speed / 70);                  // mach ein
  }
}
```

Die beiden Funktionen `play_melody_1()` und `play_melody_2()` unterscheiden sich nur in den Definitionen der Tonhöhen und Tonlängen.

```
/* play_melody_2: Funktion zum Abspielen der zweiten Melodie */
void play_melody_2(int song_speed) {
  int melody_tone[38] = {                    // Melodie-Töne
    NOTE_D4, NOTE_F4, NOTE_F4, NOTE_A4,      // Auftakt
    NOTE_C3, NOTE_C3, NOTE_C3, NOTE_D3,      // Takt 1
    0, NOTE_D4, NOTE_F4, NOTE_F4, NOTE_C5,   // Takt 2
    NOTE_A4, NOTE_C3, NOTE_C3, NOTE_C3, NOTE_D3,  // Takt 3
    0, NOTE_A4, NOTE_C5, NOTE_A4, NOTE_A4,   // Takt 4
    NOTE_G4, NOTE_D3, NOTE_F3, NOTE_D3, NOTE_E3,  // Takt 5
    0, NOTE_A4, NOTE_C5, NOTE_A4, NOTE_A4,   // Takt 6
    NOTE_G4, NOTE_D3, NOTE_F3, NOTE_D3, NOTE_E3}; // Takt 7
  int melody_duration[38] = {   // Ton-Längen
    4, 8, 8, 4,      4, 8, 4, 3,             // Auftakt und Takt 1
    8, 4, 8, 8, 8,   4, 4, 8, 4, 3,          // Takt 2 und Takt 3
    8, 8, 8, 8, 8,   2, 8, 8, 8, 2,          // Takt 4 und Takt 5
    8, 8, 8, 8, 8,   2, 8, 8, 8, 4   };      // Takt 6 und Takt 7

  for (int i = 0; i < 38; ++i) {     // Schleife durch alle Melodie-Töne
    if (melody_tone[i] > 0)                  // 0 bedeutet Pause, alles andere sind Töne
      tone(speaker_pin, melody_tone[i]);     // starte den Ton
    delay(song_speed / melody_duration[i]);  // warte die Länge der Note ab
    noTone(speaker_pin);                     // stopp die Ausgabe am Lautsprecher
    delay(song_speed / 70);                  // mach eine kurze Pause vor dem nächsten Ton
  }
}
```

Die for-Schleife ist bei beiden Funktionen identisch.

Experiment 7: Fragen und Anregungen

- Kombiniere das Programm mit dem Würfel, so dass am Ende des Würfelrollens eine kurze Melodie gespielt wird. Man kann auch unterschiedliche Melodien spielen abhängig vom Punktestand.

Experiment 8: LED dimmen mit Pulsweiten-Modulation

Analoge Ausgabe mit PWM (1)

Die digitalen Ausgänge die wir bisher benutzt haben, kennen nur zwei Zustände: `HIGH` (= 5V) oder `LOW` (= GND). Oft möchte man aber Spannungswerte erzeugen, die dazwischen liegen. Eine häufige Anwendung ist das Regeln von LED-Beleuchtung, elektrischen Motoren, usw.

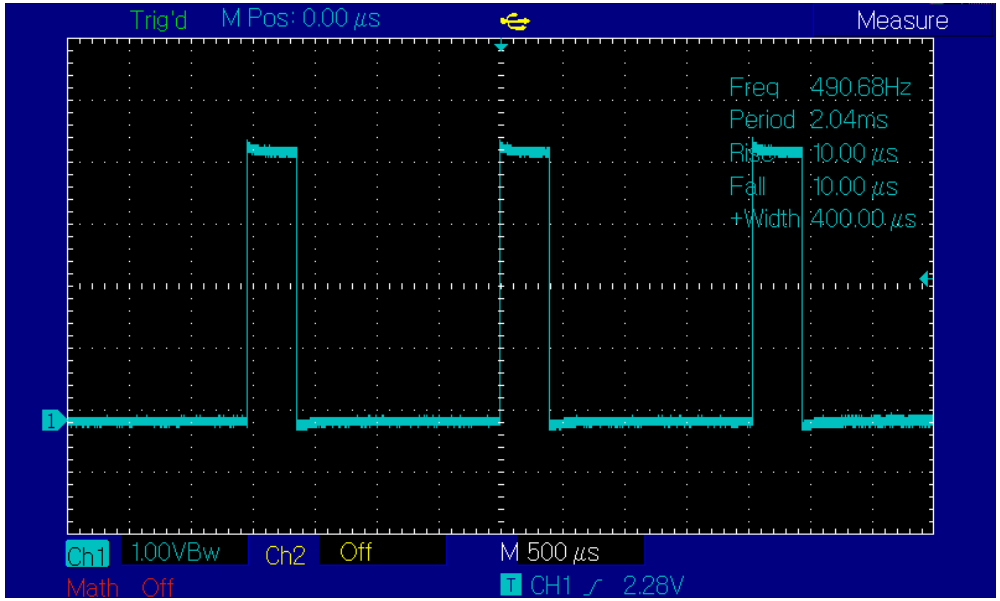
Der Arduino kann nicht direkt andere Spannungswerte als `HIGH` und `LOW` produzieren. Er hat aber die Möglichkeit, ein PWM-Signal zu produzieren, das über die Zeit im Mittel andere Spannungswerte zulässt.

Ein PWM-Signal besteht aus einem schnellen Wechsel zwischen `HIGH` und `LOW`, bei dem der Anteil der HIGH-Phase – der sogenannte Duty-Cycle - grösser oder kleiner gesetzt werden kann. Im zeitlichen Mittel erlaubt diese Methode, beliebige Spannungswerte zwischen `LOW` und `HIGH` zu erzeugen.

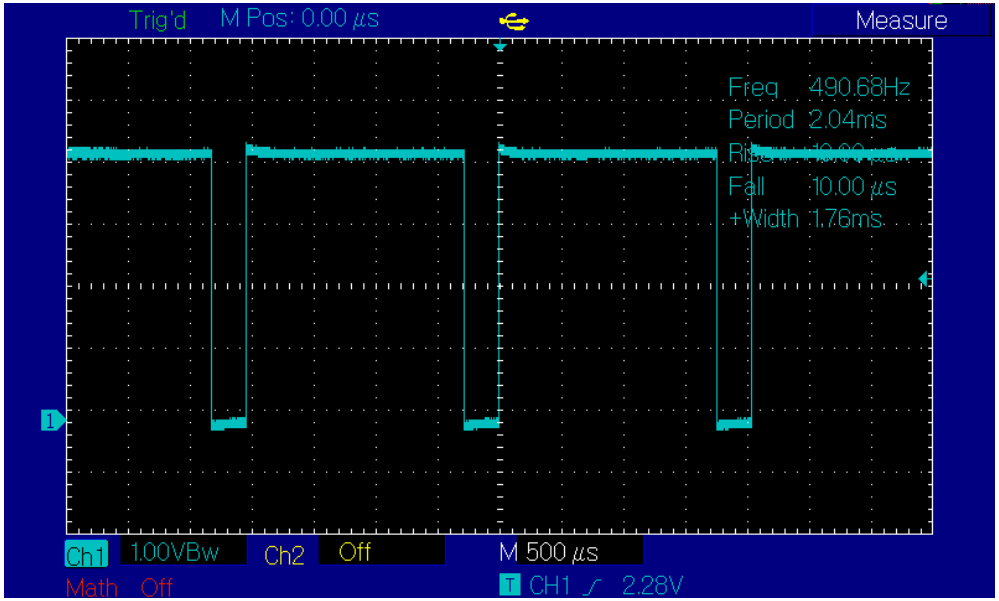
Analoge Ausgabe mit PWM (2)

Mit der Funktion `analogWrite(pin, value)` kann man ein PWM-Signal auf einen der Anschlüsse am Mikrocontroller ausgeben. Der Wert `value` bestimmt den Duty-Cycle. Dieses ist ein 8-Bit-Wert, der also Werte zwischen 0 (entsprich vollständig aus) und 255 (entspricht dauerhaft an) annehmen kann.

Oszilloskop-Ansicht eines PWM-Signals

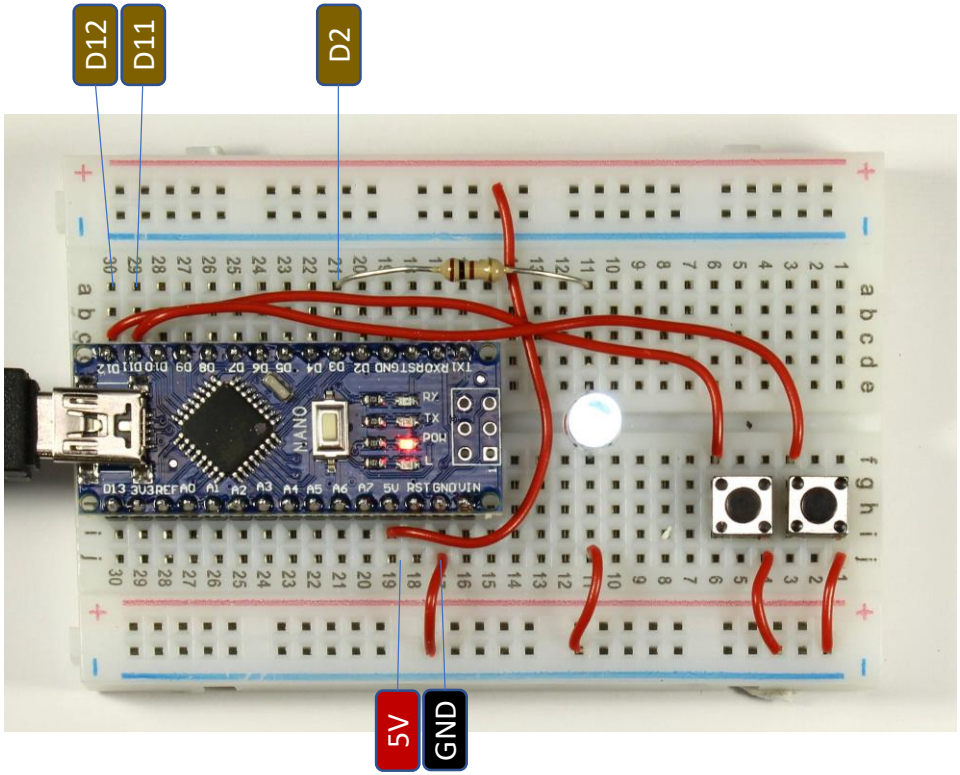


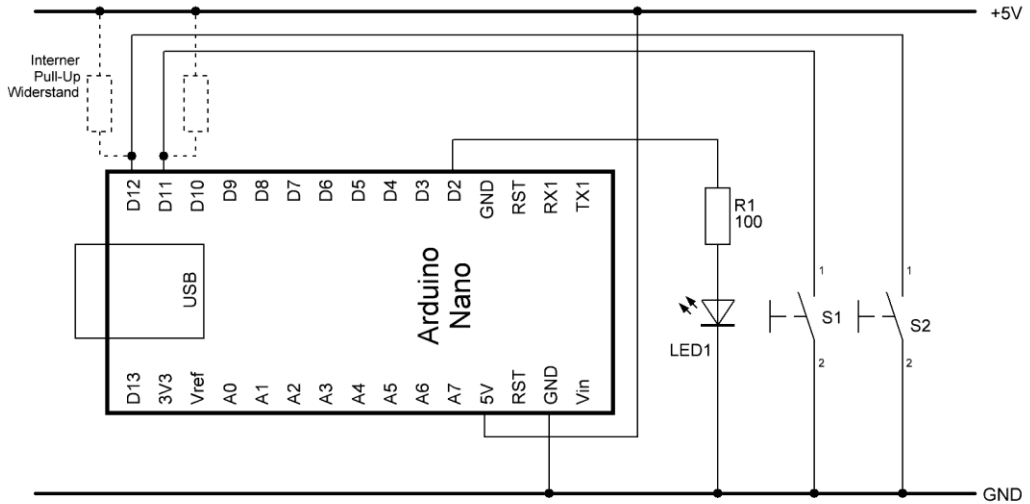
Duty Cycle = 50



Duty Cycle = 220

Experiment 8: Aufbau und Schaltung





100 Ohm Farbcode:
- Braun = 1
- Schwarz = 0
- Braun = 1

Experiment 8: Programm

_8_LED-Dimmer

```
/* Arduino Kurs FES Juli 2017
 *  Programm: LED-Dimmer
 */

/* I/O Pins */
int button_low = 12;
int button_high = 11;
int led_pin = 3;

int duty_cycle = 100;                    // Anfangswert nach dem Einschalten

void setup() {
  pinMode(button_low, INPUT_PULLUP);
  pinMode(button_high, INPUT_PULLUP);
}

void loop() {
    if (digitalRead(button_low) == LOW) {   // wenn button_low gedrückt ist, ...
      if (duty_cycle> 0)                    // ... duty_cycle verkleinern
        --duty_cycle;
    };
    if (digitalRead(button_high) == LOW) {  // wenn button_high gedrückt ist, ...
      if (duty_cycle < 255)                 // duty_cycle vergrössern
        ++duty_cycle;
    };
    analogWrite(led_pin, duty_cycle);       // duty_cycle setzen
    delay(50);                              // einen Moment warten
}
```

Die Variable `duty_cycle` bewegt sich zwischen 0 und 255.
Wenn eine der beiden Tasten gedrückt ist, wird `duty_cycle` um 1 herauf oder herunter gezählt, bis das Ende des zulässigen Bereichs erreicht ist.

Logarithmische Leuchtstärke

Das Programm von Experiment 8 funktioniert zwar, hat aber den Nachtteil, dass sich die Helligkeit bei geringer Leuchtstärke sehr schnell und bei großer Leuchtstärke nur langsam ändert. Das liegt daran, dass eine Änderung zwischen geringen Werten, z.B. der Schritt 2 auf 3 vom Auge als eine große Helligkeitssteigerung wahrgenommen wird, während eine Änderung von z.B. 253 auf 254 mit dem Auge fast nicht sichtbar ist.

Wie lässt sich das Problem lösen? Für eine gleichmäßige Helligkeitssteigerung brauchen wir Schrittweiten, die am Anfang sehr gering und gegen Ende sehr viel größer sind. Dafür eignet sich eine logarithmische Skala.

Die folgende Tabelle zeigt 31 Helligkeitswerte entsprechend einer angenäherten logarithmischen Skala. Diese Werte ergeben gleichmäßige Schrittweiten zwischen dunkel und hell.

```
// Tabelle mit logarithmischen PWM-Werten für gleichmässige
Leuchtabstufung
int duty_cycle_table[31] =
{0, 1, 2, 3, 5, 8, 12, 15, 20, 25, 30, 36, 43, 50, 58, 66, 75, 84,
 94, 104, 115, 127, 139, 152, 165, 178, 193, 208, 223, 239, 255};
```

LED-Dimmer Version 2

_8_LED-Dimmer_v2

```
/* Arduino Kurs FES Juli 2017
 *  Programm: LED-Dimmer
 */

// Tabelle mit logarithmischen PWM-Werten für gleichmässige Leuchtabstufung
int duty_cycle_table[31] =
{0, 1, 2, 3, 5, 8, 12, 15, 20, 25, 30,
 36, 43, 50, 58, 66, 75, 84, 94, 104, 115,
 127, 139, 152, 165, 178, 193, 208, 223, 239, 255};

/* I/O Pins */
int button_low = 12;
int button_high = 11;
int led_pin = 3;

int light_level = 15;

void setup() {
  pinMode(button_low, INPUT_PULLUP);
  pinMode(button_high, INPUT_PULLUP);
}

void loop() {
    if (digitalRead(button_low) == LOW) {                 // wenn button_low gedrückt ist, ...
      if (light_level > 0)                                // ... light_level verkleinern
        --light_level;
    };
    if (digitalRead(button_high) == LOW) {                // wenn button_high gedrückt ist, ...
      if (light_level < 30)                               // light_level vergrössern
        ++light_level;
    };
    analogWrite(led_pin, duty_cycle_table[light_level]);  // duty_cycle setzen
    delay(100);                                           // einen Moment warten
}
```

Bei dieser Version wird die Variable `light_level` zwischen den 31 Helligkeitswerten hin und her bewegt.

Der PWM-Duty-Cycle wird mit dem Aufruf von `analogWrite()` gesetzt. Dabei wird der zu `light_level` gehörende Tabellenwert verwendet.

Experiment 8: Fragen und Anregungen

- Verwende den Lautsprecher, um mit einem kurzen Ton darauf hinzuweisen, wenn der Anwender das untere oder obere Ende des Dimm-Bereiches erreicht hat.

Teil 4: Analog Digital Converter (ADC)

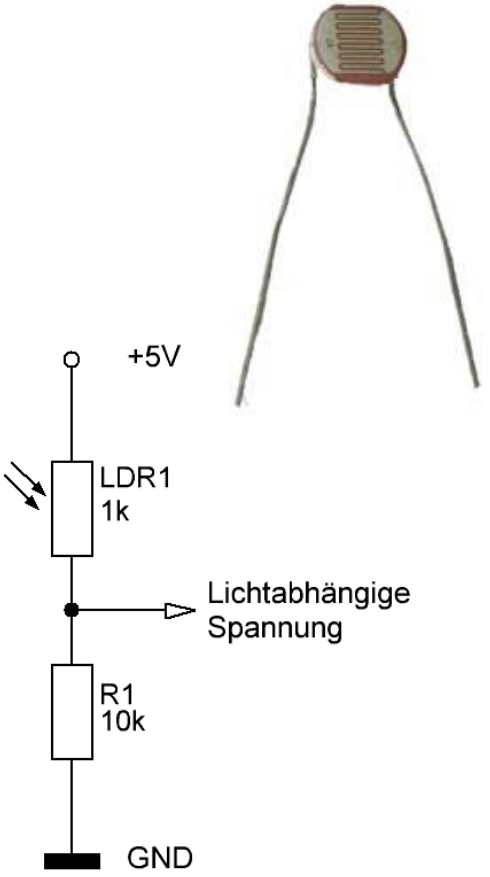
Helligkeiten als Zahlenwert einlesen
Kommunikation zwischen Arduino und Computer-Terminal
Lichtschranke
Licht-gesteuerte Orgel

Licht-abhängiger Widerstand (LDR)

Ein LDR (auch Fotowiderstand genannt) ist ein Widerstand, der seinen Wert in Abhängigkeit von der Beleuchtung verändert. Ein LDR hat normalerweise einen sehr weiten Widerstandsbereich von wenigen 100 Ohm bis zu mehreren MOhm, folgt dem Lichtwechsel allerdings relativ langsam.

Für unsere Experimente verwenden wir den LDR mit einem Widerstand nach Masse (GND). So erhalten wir Spannungswerte, die Licht-abhängig sind:

Hohe Lichtstärke -> hohe Spannungswerte (max. 5V)
Niedrige Lichtstärke -> niedrige Spannungswerte (min. 0V)



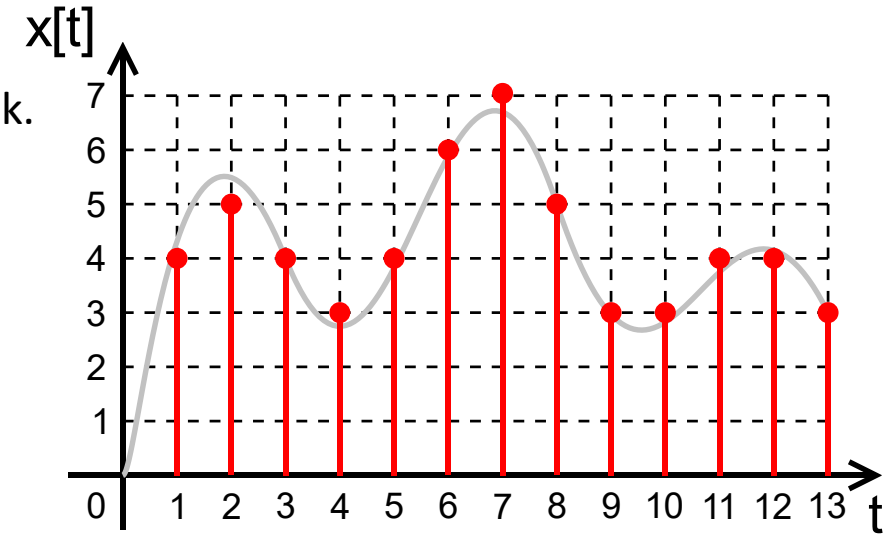
Analog-Digital Converter ADC

Der Mikrocontroller verfügt über einen ADC, der einen externen Spannungswert zwischen 0 und 5V in einen Zahlenwert umwandelt. Dieser Zahlenwert kann dann als Variable im Programm verwendet werden, um Aktionen auszulösen.

Der ADC des Arduino Nano hat 8 Kanäle entsprechend den analogen Pins A0 bis A7.
Die Funktion `analogRead(pin)` liest einen Spannungswert an einem Pin aus und liefert einen Ganzzahl-Wert (`int`) an das Programm zurück.

Der ADC des Arduino Nano ist ein 10-Bit Wandler. Damit werden 2^{10} = 1024 verschiedene Werte möglich. Der Wertebereich ist 0 also bis 1023.

Das Einlesen eines externen Spannungsgwertes braucht etwas Zeit. Der ADC benötigt 0.1 msec für eine Konvertierung.



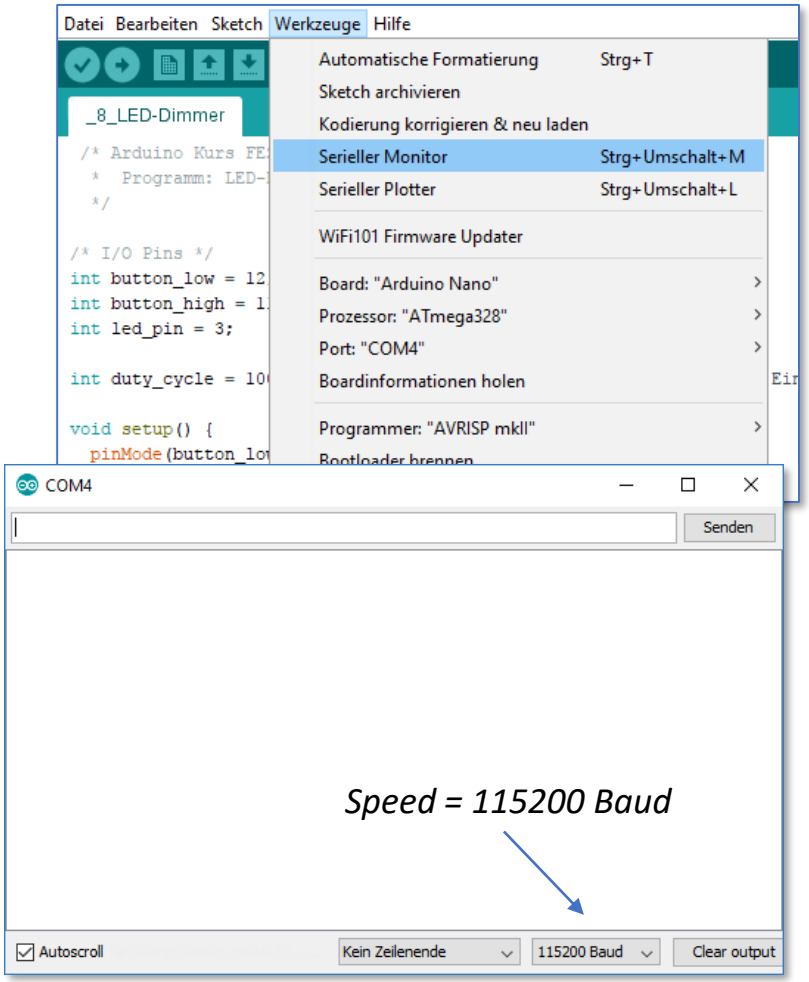
Kommunikation mit dem Computer

Die Arduino-Plattform enthält eine einfache Möglichkeit, Daten mit dem PC auszutauschen. Damit ist es möglich, Ausgaben vom Arduino auf den PC zu geben.

Für die Kommunikation wird eine serielle Schnittstelle über den USB-Anschluss verwendet. Die Funktion `Serial.begin(speed)` öffnet einen Kanal zum PC. Die Funktionen `Serial.print()` gibt einen Wert an den PC aus. Die Funktion `Serial.println()` fügt zusätzlich eine neue Zeile in die Ausgabe ein.

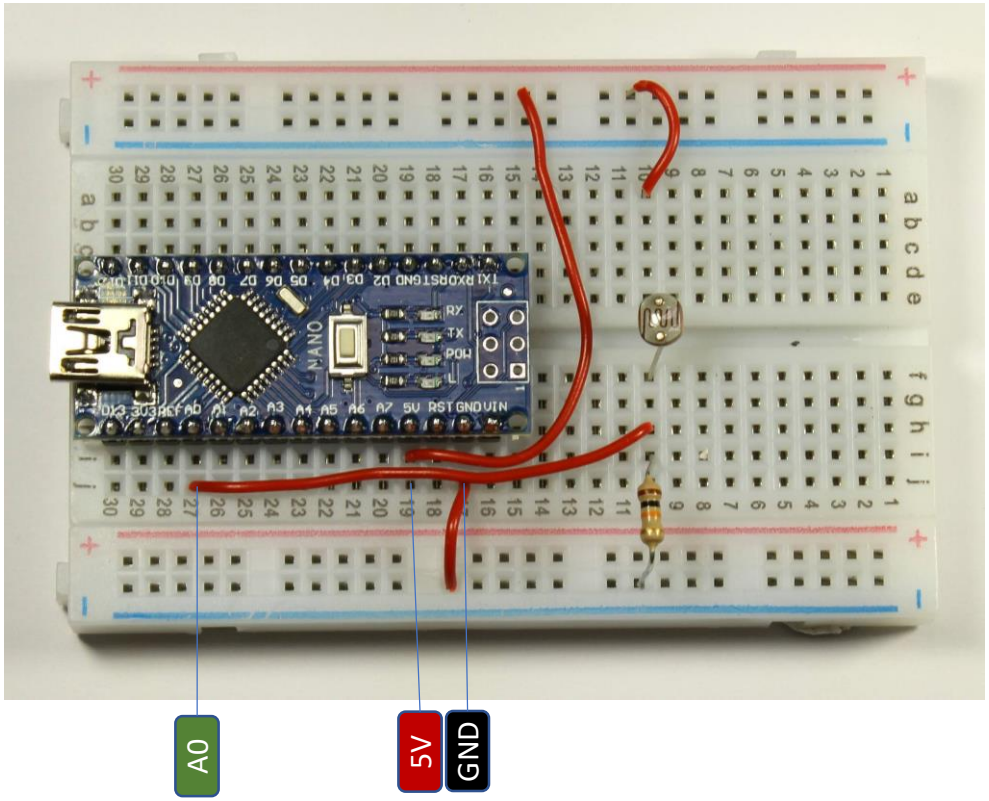
Auf dem PC muss dazu das Werkzeug „Serieller Monitor“ geöffnet werden, um die Daten vom Arduino entgegen zu nehmen.

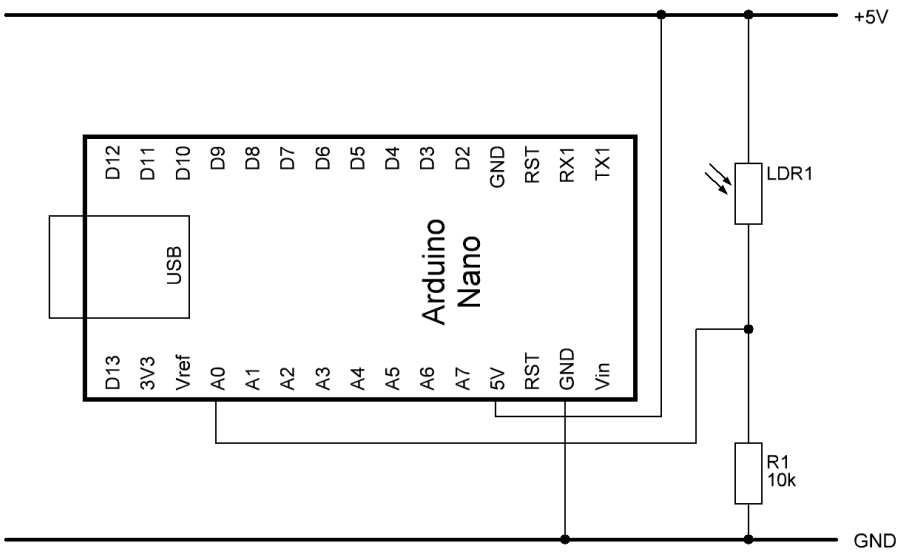
Wichtig: Die eingestellte Geschwindigkeit (Baud) muss mit dem Speed-Wert von `Serial.begin()` übereinstimmen. Wir verwenden 115200 Baud.



Experiment 9: LDR mit dem ADC auslesen

Experiment 10: Aufbau und Schaltplan





10 kOhm Farbcode:

- Braun = 1
- Schwarz = 0
- Orange = 3

Experiment 10: Programm

_9_LDR

```
/* Arduino Kurs FES Juli 2017
 *  Programm: LDR mit dem ADC auslesen
 */

// I/O-Pins
int adc_ldr_pin = 0;

int adc_value;

void setup() {
  Serial.begin(115200);
}

void loop() {
    adc_value = analogRead(adc_ldr_pin);    // ADC auslesen
    Serial.println(adc_value);              // Ergebnis an den Computer senden
    delay(250);                             // einen Moment warten
}
```

Experiment 9: Fragen und Anregungen

- Probiere andere Widerstandswerte aus. Was passiert, wenn der Widerstand nach GND kleiner oder grösser wird?

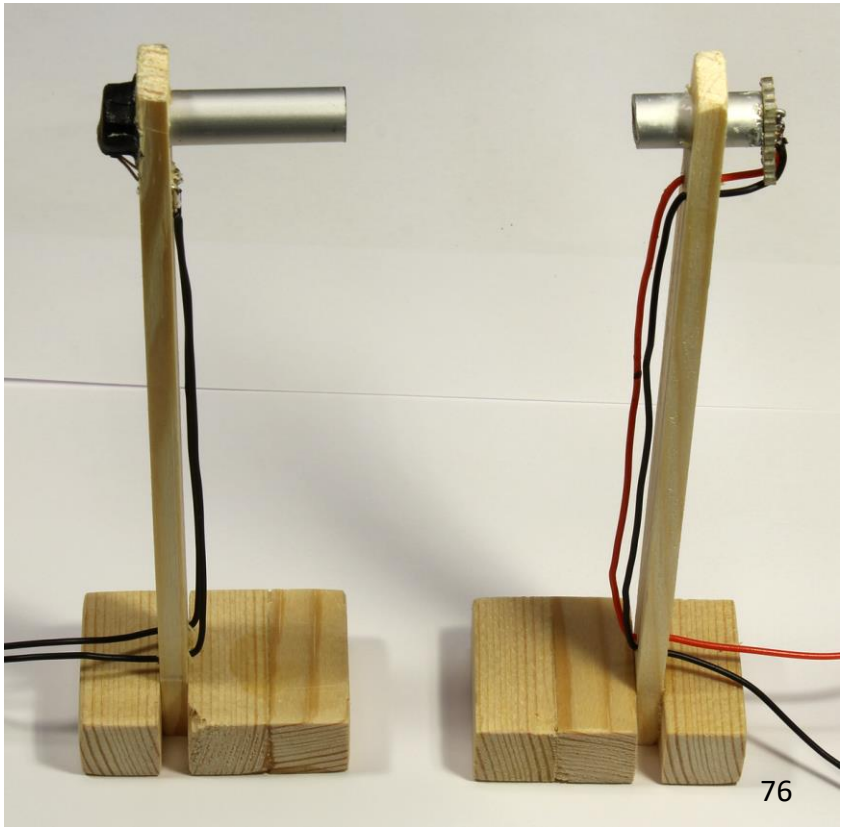
Experiment 10: Lichtschranke

Lichtschranke: Sender und Empfänger

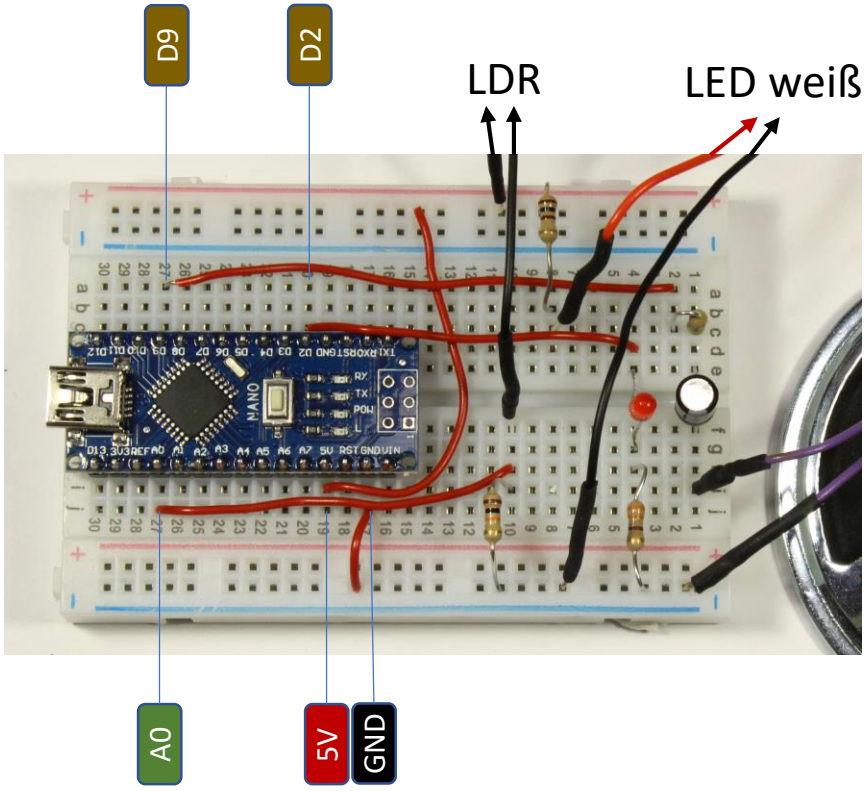
Lichtschranken werden in der Technik für vielfältige Aufgabe eingesetzt zur Erfassung von Ereignissen, deren Zählung, etc. Jeder kennt sie von der Unterbrechung der Türschließung an Aufzügen, aber auch Personenerfassung, Drehzahlmessung, usw.

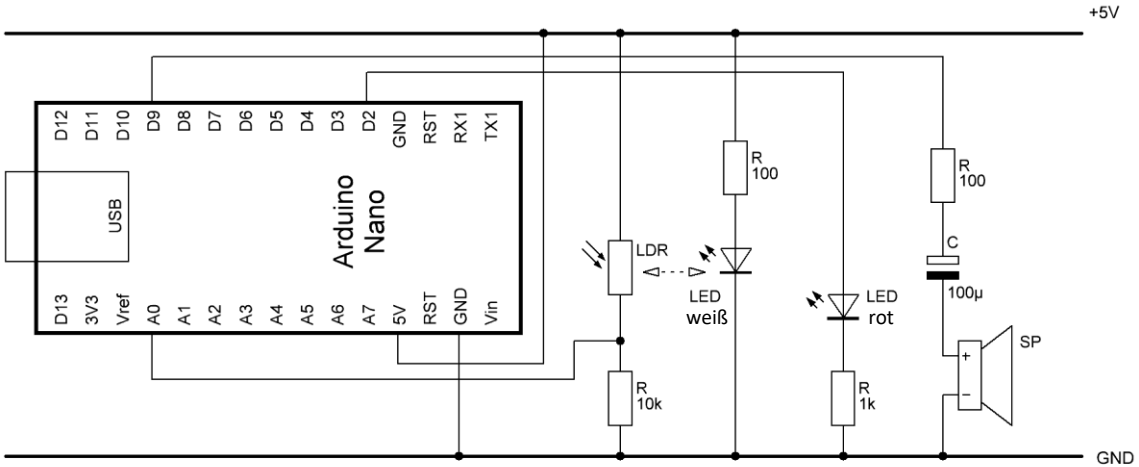
Zu einer Lichtschranke gehören zwei Elemente: Ein Sender, hier die weiße LED, und ein Empfänger, hier der LDR.

Der LDR hat keine Richtungsempfindlichkeit. Deshalb muss er in einer geeigneten Röhre verpackt werden, die die Empfindlichkeit auf nur eine Richtung beschränkt. Wir verwenden ein Rohr mit etwa 8mm Durchmesser. Es ist von Vorteil, wenn das innere des Rohres schwarz ist, um Streulicht zu reduzieren.



Experiment 10: Aufbau und Schaltplan





Experiment 10: Programm

_10_Lichtschranke § | pitches.h

```
/* Arduino Kurs FES Juli 2017
 *  Programm: Lichtschranke
 */

#include "pitches.h"

// I/O-Pins
int adc_ldr_pin = 0;
int red_led_pin = 2;
int speaker_pin = 9;

int adc_value;                          // aktueller ADC-Wert vom LDR
int threshold = 500;                    // Schwellwert für die Erkennung der Unterbrechung
int count = 0;                          // Zähler der Unterbrechungen
boolean ldr_status = HIGH;              // aktueller LDR-Status (HIGH oder LOW)

void setup() {
  pinMode(red_led_pin, OUTPUT);
  Serial.begin(115200);
}

void loop() {
    adc_value = analogRead(adc_ldr_pin);    // ADC auslesen
    if (ldr_status == HIGH) {               // falls der aktuelle Status HIGH ist ...
      if (adc_value < threshold) {          // ... testen, ob der ADC-Wert unter dem Schwellwert liegt
        ldr_status = LOW;                   // In dem Fall den Status ändern
        digitalWrite(red_led_pin, HIGH);    // Rote LED einschalten
        tone(speaker_pin, NOTE_D3, 50);     // Statusänderung mit einem kurzen Ton anzeigen
      };
    } else {
      if (adc_value > threshold) {          // ... sonst testen, ob der ADC-Wert über dem Schwellwert liegt
        ldr_status = HIGH;                  // In dem Fall den Status ändern
        digitalWrite(red_led_pin, LOW);     // Rote LED ausschalten
        tone(speaker_pin, NOTE_A3, 50);     // Statusänderung mit einem kurzen Ton anzeigen
        ++count;                            // Unterbrechung zählen
        Serial.println(count);              // Den neuen Zähler an den PC ausgeben
      };
    };
}
```

Das Programm benutzt die boolesche Variable `ldr_status`, um den aktuellen Status des LDRs festzuhalten. Im Normalzustand ist der LDR beleuchtet und der Status `HIGH`. Wenn der Lichtstrahl unterbrochen wird, wechselt der Status auf `LOW`. Erst bei einem erneuten Wechsel zu `HIGH` wird die Unterbrechung als Ereignis gezählt.

Experiment 10: Fragen und Anregungen

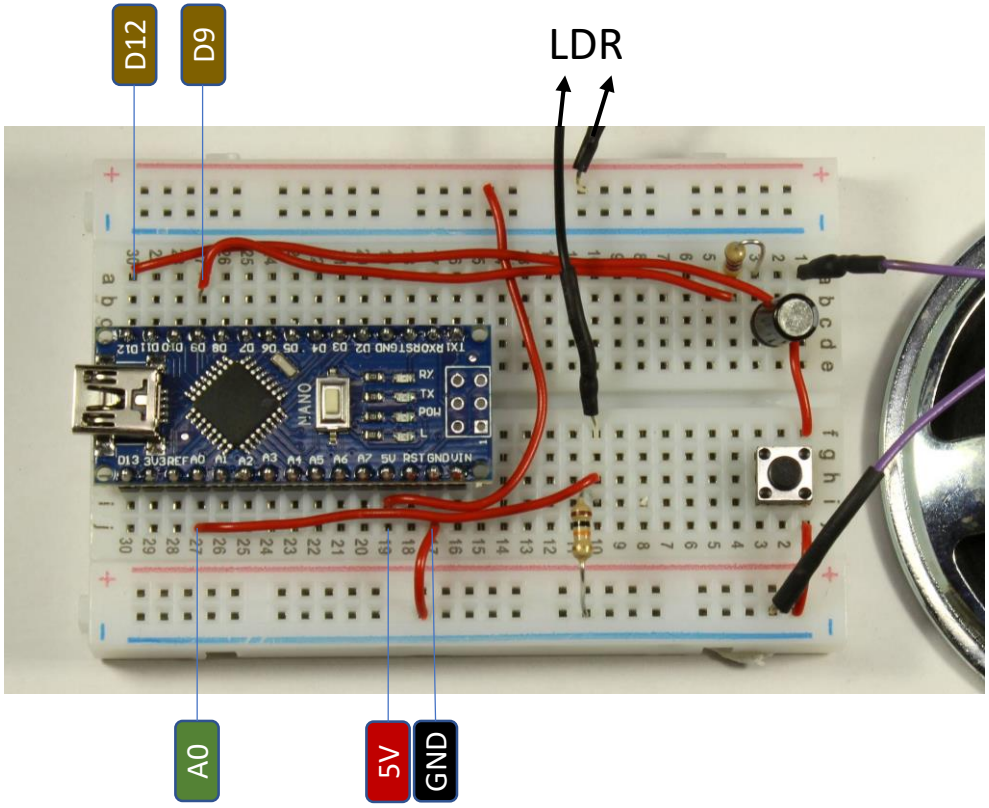
- Die Funktion `tone()` wird hier in einer anderen Form verwendet als bei der Mini-Orgel. Die Funktion gibt es in zwei verschiedenen Version:
  - `Tone(pin, frequency)`
  - `Tone(pin, freqeuency, duration)`

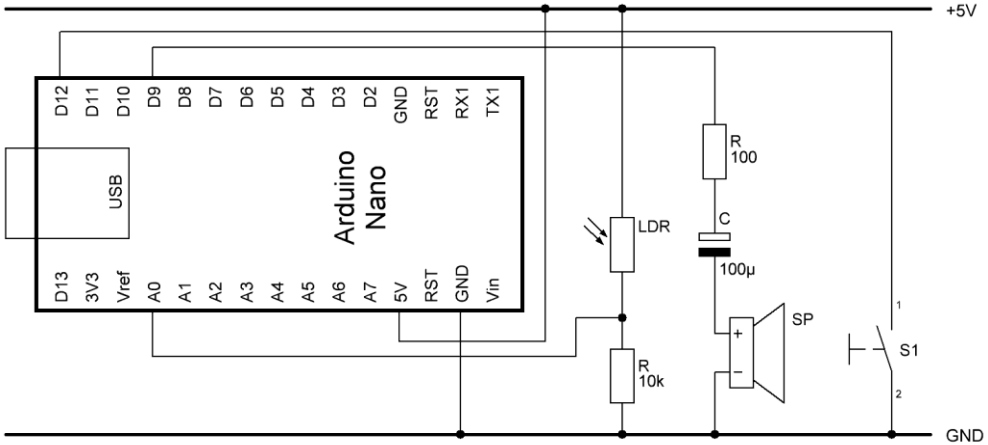
  In der zweiten hier verwendeten Version wird die Länge des Tons als dritter Parameter übergeben. Dadurch ist der Aufruf von `noTone()` nicht mehr notwendig (siehe https://www.arduino.cc/en/Reference/Tone)
- Wie weit kann man Sender und Empfänger von einander entfernen, um immer noch eine zuverlässige Messung zu erhalten?
- Teste die Geschwindigkeit aus, mit der Unterbrechungen erfasst werden.
- Was bräuchte man für eine schnellere Erfassung, z.B. einen Drehzahl-Messer?

Experiment 11: Lichtorgel

Zum Abschluss kombinieren wir den LDR mit der Ausgabe von Tönen und versuchen, die Tonhöhe über die Beleuchtungsstärke zu steuern. So entsteht ein neues Musikinstrument.

Experiment 11: Aufbau und Schaltung





Experiment 11: Programm

_11_Lichtorgel | pitches.h

```
/* Arduino Kurs FES Juli 2017
   Programm: Lichtorgel
*/

int ldr_value;                          // eingelesener ADC-Wert vom LDR

// Anschluss Pins
int speaker_pin = 9;                    // Laustprecher an Pin 9
int button_pin = 12;                    // Taster an Pin 12
int ldr_pin = 0;                        // LDR and ADC-Pin 0

void setup() {
  pinMode(speaker_pin, OUTPUT);         // Lautsprecher am Ausgang
  pinMode(button_pin, INPUT_PULLUP);    // Taster am Eingang
}

void loop() {
  if (digitalRead(button_pin) == LOW) {     // wenn die Taste gedrückt ist ...
    ldr_value = analogRead(ldr_pin);        // LDR-Wert vom ADC abholen
    tone(speaker_pin, ldr_value * 2 + 31);  // den LRD-Wert als Tonhöhe ausgeben
  } else {
    noTone(speaker_pin);                    // ... sonst den Ton abstellen
  };
}
```

Das Programm liest den aktuellen Wert vom LDR ein und gibt ihn als Tonhöhe aus, solange die Taste gedrückt ist.

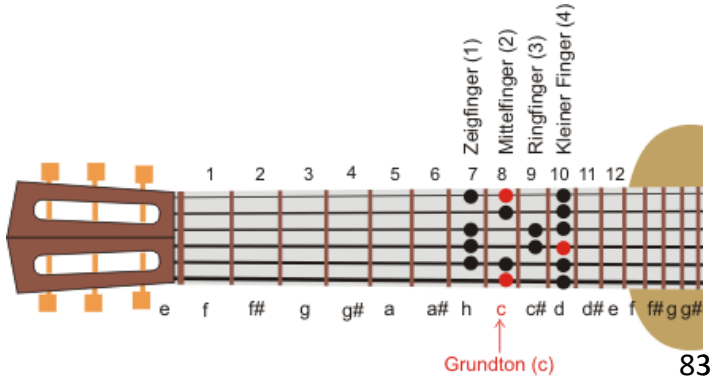
Der LDR-Wert wird verdoppelt, um einen größeren Tonumfang zu erhalten. Da die Funktion `tone()` keine Werte kleiner als 31 akzeptiert, wird zusätzlich 31 addiert. Selbst wenn der LDR-Wert 0 wäre, wird der Wert 31 an `tone()` übergeben.

Experiment 11: Lichtorgel Version 2

_11_Lichtorgel_v2 | pitches.h

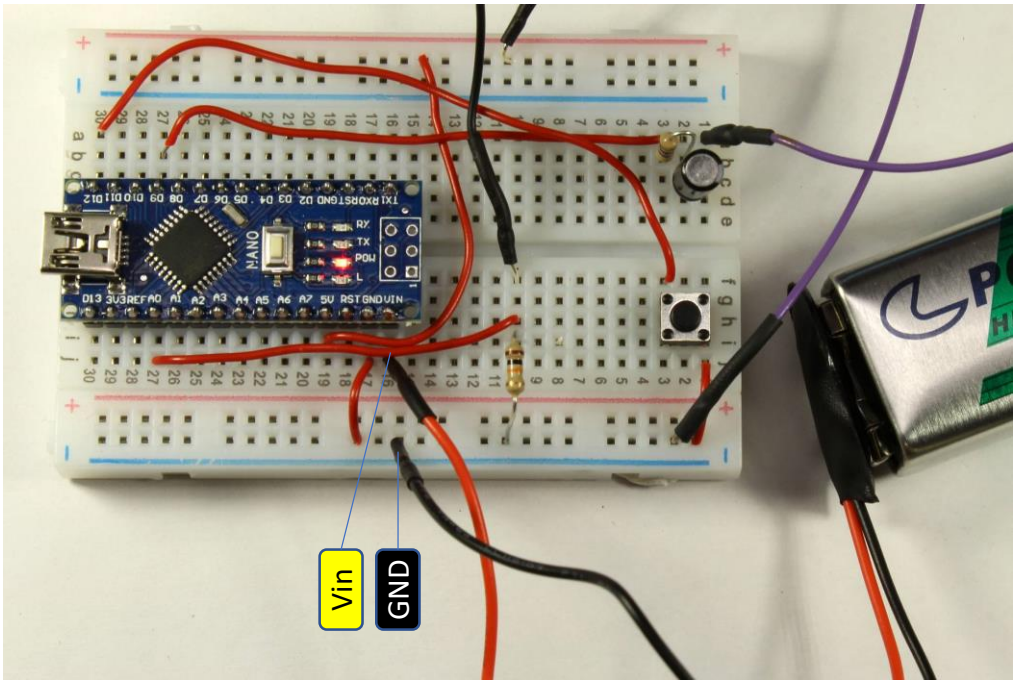
```
/* Arduino Kurs FES Juli 2017
   Programm: Lichtorgel v2
*/

// Tabelle der Frequenzen von Tönen über 4 Oktaven
int pitches[36] = {131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233, 247,
                   262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494,
                   523, 554, 587, 622, 659, 698, 740, 784, 831, 880, 932, 988};
int ldr_value;                          // eingelesener ADC-Wert vom LDR

// Anschluss Pins
int speaker_pin = 9;                    // Laustprecher an Pin 9
int button_pin = 12;                    // Taster an Pin 12
int ldr_pin = 0;                        // LDR and ADC-Pin 0

void setup() {
  pinMode(speaker_pin, OUTPUT);         // Lautsprecher am Ausgang
  pinMode(button_pin, INPUT_PULLUP);    // Taster am Eingang
}

void loop() {
  if (digitalRead(button_pin) == LOW) {  // wenn die Taste gedrückt ist ...
    // LDR-Wert vom ADC holen
    ldr_value = analogRead(ldr_pin);
    ldr_value = ldr_value * 2;          // LDR-Wert verdoppeln für weiteren Tonumfang
    // suche den nächsten passenden Ton aus der Pitch-Tabelle
    int i = 0;
    while ((pitches[i] < ldr_value) && (i < 35))  // solange die Pitch-Werte kleiner ...
      ++i;                              // ... sind als der LDR-Wert, weiter suchen
    // Ton ausgeben
    tone(speaker_pin, pitches[i]);      // spiel den passenden Wert aus der Pitch-Tabelle
    delay(50);                          // warte einen Moment
  } else {
    noTone(speaker_pin);                // ... sonst den Ton abstellen
  };
}
```

Das vorherige Programm war noch nicht sehr musikalisch. Diese Version des Programms enthält eine Tabelle der korrekten Tonhöhen über 4 Oktaven. In der while-Schleife wird aus dieser Tabelle der nächste passende Wert herausgesucht, so dass nur „richtige“ Töne ausgegeben werden. Die while-Schleife funktioniert hier ähnliche den Bünden einer Gitarre.



Zum Abschluss: Stromversorgung mit Batterie

Für unsere Experiment haben wir den Arduino immer am USB-Port betrieben und so die notwendige Betriebsspannung bekommen.
Wenn das Programm übertragen ist, kann man einen Arduino aber auch mit anderen Stromquellen betreiben. Zum Beispiel eignet sich eine 9V-Blockbatterie. Der Pluspol der Batterie muss an den Vin-Anschluss gelegt werden, und der Minuspol an GND.



Achtung: Höhere Spannungen als 5V dürfen nur mit dem Vin-Anschluss verbunden werden. Wenn z.B. 9V Spannung den 5V-Anschluss oder einen der Ports berührt, führt das zur Zerstörung des Arduinos.

Wir wünschen viel Spaß
und jede Menge
Kreativität beim Basteln
mit dem Arduino