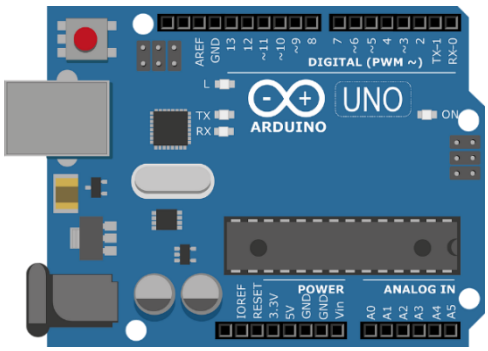
Steuern und Regeln

mit Arduino-Mikrocontrollern

mit Tinkercad



Mit diesem Lernheft lernst du:

- Wichtige Grundlagen zu Mikrocontrollern.
- Wie du mit C++ einen Arduino-Mikrocontroller programmierst

Das kannst du am Ende:

- Eigene Steuerungen oder Regelungen herstellen

Das brauchst du:

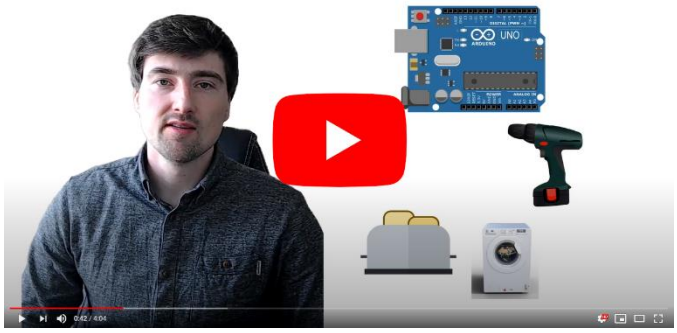
- ein Konto auf www.tinkercad.com
- oder einen Klassencode und einen Spitznamen von deiner Lehrkraft

Brauchst du einen Arduino, um in diesem Lernheft zu arbeiten? Nein!
Aber wenn du einen hast, brauchst du die Simulation nicht und kannst deine Projekte direkt auf dem richtigen Arduino ausführen. Die Schaltpläne sind die gleichen, wie in den Simulationen.

Inhaltsverzeichnis:

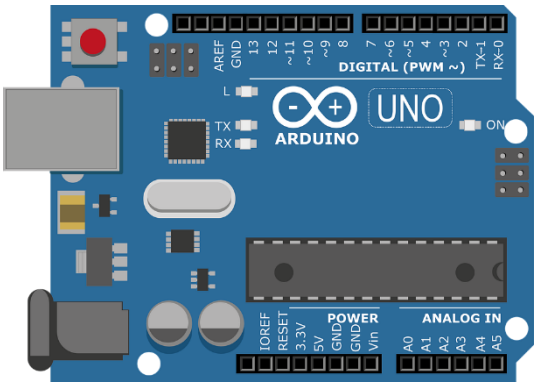
Was ist ein Mikrocontroller?

Aufgabe: Lies den Text oder schaue dieses Video.



Ein Mikrochip wird als Mikrocontroller bezeichnet, wenn er einen Prozessor und auch Peripheriefunktionen zugleich enthält. Meistens befinden sich auch der Arbeitsspeicher und der Programmspeicher auf demselben Chip. Er wird daher auch „Ein-Chip-Computersystem“ oder „System-on-a-Chip“ (kurz: SoC) genannt.

Mikrocontroller werden in vielen technischen Alltagsgegenständen verwendet. Diese Systeme werden als „eingebettete Systeme“ bezeichnet. Beispielsweise befinden sich Mikrocontroller in Waschmaschinen, Fernbedienungen, Bluetooth-Boxen, Schreibtischlampen, Kraftfahrzeugen (Steuergeräte für z. B. ABS, Airbag, Motor, Kombiinstrument, ESP usw.), Armbanduhren, Monitoren und vielen weiteren Geräten. Mikrocontroller werden so ausgewählt, dass ihre Leistung und Ausstattung den Anforderungen genügen. Sie haben ausreichend Leistung, die aber deutlich geringer ist als bei Computern und mobilen Endgeräten. Daher sind sie sehr günstig. Häufig kosten sie in großen Stückzahlen weniger als 1 EUR pro Stück.



Ein Arduino ist ein Board (eine bestückte Platine), auf dem ein Mikrocontroller ein paar weitere Bauteile und vor allem Anschlüsse zu finden sind. Die Anschlüsse sind Ein- oder Ausgänge für Signale. Der Arduino wird mit einer eigenen Software programmiert. Eine Software, in der programmiert wird, ist eine Entwicklungsumgebung (kurz: IDE). Die Arduino-Programmiersprache ähnelt der Sprache C++, ist jedoch stark vereinfacht. Ein „Sketch“ ist das geschriebene Programm. Es wird mit einem Compiler in Maschinensprache umgewandelt und per USB-auf den Speicher des Mikrocontrollers geladen. Mit dem Arduino können Werte mit Sensoren gemessen und Aktoren gesteuert werden. Aktoren sind beispielsweise Motoren, Leuchtdioden und Displays. Mit dem Arduino können zum Beispiel Temperaturwarner, Umwelt-Messstationen oder Roboter gebaut werden.

Hier geht’s zum Quiz:

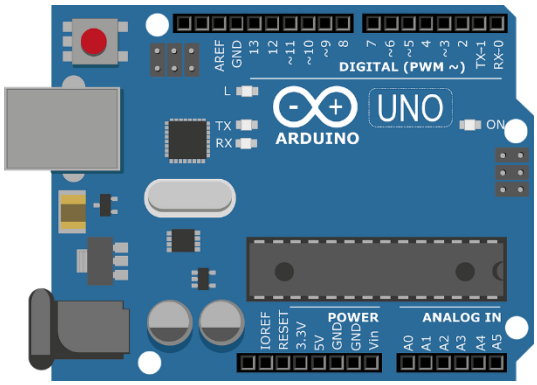
Quiz

Aufbau und Steckbrett

Arduino-Board: „Arduino Uno“

Das Arduino-Board „Uno“ verfügt über einen **USB-Anschluss**, über den der Mikrocontroller programmiert wird. Daneben befindet sich die **Stromversorgung**. Hier werden 5V-9V Gleichspannung angeschlossen. Die schwarzen Steckleisten sind in verschiedene Bereiche eingeteilt. Ein Bereich dient der **Spannungsversorgung**, einer ist für **analoge Eingänge** und einer für **digitale Ein-/Ausgänge**. Der **Mikrocontroller** (auch Prozessor genannt) sitzt in der Mitte. Er ist entweder fest verlötet oder steckbar. Dazu verfügt der „Arduino Uno“ über mehrere **Status-LED**. Soll der Arduino neu gestartet werden, kann ein **Reset-Taster** gedrückt werden.

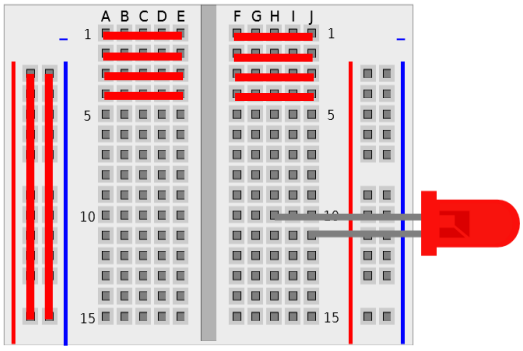
Aufgabe: Klicke auf den Arduino oder hier und beschrifte den Arduino mit den fettgedruckten Begriffen (Link: https://apps.zum.de/apps/arduino-uno-mikrocontroller)



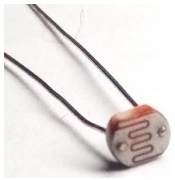
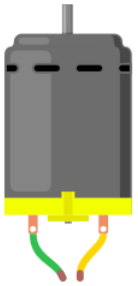
Das Steckbrett

Auf das Steckbrett (engl. Breadboard) werden die einzelnen Bauteile und Drähte gesteckt. Ohne zu wissen welche Stecklöcher miteinander verbunden sind, ist es nicht möglich eine Schaltung auf dem Steckbrett nachzubauen. Jede nummerierte Reihe (1–63) verbindet alle Stecklöcher der Reihe. Zwischen E und F ist diese Reihe aber unterbrochen. Es sind also immer die Stecklöcher A-E und F-J miteinander verbunden. Die Spannungsversorgung an der Seite verbindet alle Stecklöcher der Spalte. Diese Spalten werden mit 5V und GND auf dem Arduino-Board verbunden. So können viele Bauteile mit Spannung versorgt werden.

Eine LED muss daher, wie auf der Abbildung dargestellt im Steckbrett stecken. Es dürfen nicht beide Anschlüsse in einer Reihe stecken.



Sensoren und Aktoren

Sensor	Aktor
	

Sensoren dienen der Eingabe, sie erfassen Werte und werden auch Geber oder Fühler genannt.

Aktoren dienen der Ausgabe. Sie bewegen Dinge, senden Signale oder führen etwas aus.

Aufgabe 1: Lies den folgenden Text und markiere Sensoren und Aktoren.

Ein Auto ist längst nicht mehr nur ein Fortbewegungsmittel, das durch Treten des Gaspedals den Motor in Bewegung bringt und mit einem Lenkrad gesteuert wird. Neue Autos haben jede Menge moderne Technik verbaut. Neigungs- oder Erschütterungssensoren erkennen Einbruchversuche und lösen einen Alarm durch die Hupe und die Blinker aus. Näherungssensoren lösen einen Piepton aus einem Summer aus, wenn beim Einparken eine Kollision droht. Regnet es stark kann ein Regenfühler die Scheibenwischanlage automatisch starten lassen. Ein modernes Auto kann uns sogar davor warnen, wenn wir beim Fahren einzuschlafen drohen. Eine Kamera erkennt dabei unsere Augenliedstellung. Fallen uns die Augenlieder zu warnt uns eine Stimme aus dem Lautsprecher.

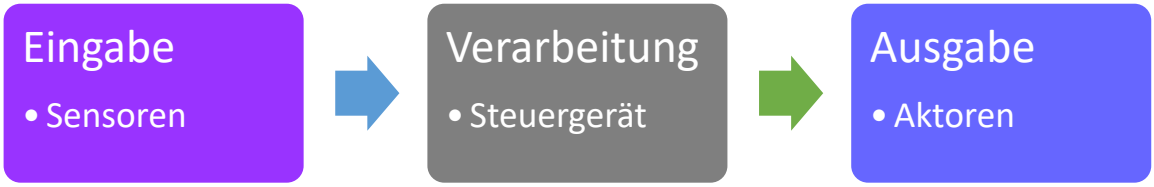
Aufgabe 2: Ziehe die Sensoren und Aktoren in die richtige Spalte der Tabelle. Hier geht es zur Challenge:

Challenge

Aufgabe 3: Kreuze an, ob es sich um einen Sensor oder Aktor handelt. Hier geht es zur Challenge:

Challenge

EVA-Prinzip



Die Eingabe erfolgt durch Sensoren. Diese werden häufig auch als Geber oder Fühler bezeichnet. Sie erfassen Werte, wie beispielsweise die Temperatur oder den Abstand und geben diese Information an das Steuergerät weiter. Das Steuergerät verarbeitet die Information. Ein Steuergerät enthält einen Mikrocontroller oder ein anderes Computersystem. Auf dem Steuergerät befindet sich ein Programm, welches die Informationen verarbeitet und nach einprogrammierten Bedingungen daraufhin Aktoren ansteuert. Aktoren werden auch Stellglieder genannt. Aktoren können zum Beispiel Motoren, Leuchtdioden oder Displays sein.

Aufgabe: Ordne Sensoren, Aktoren, Einheit der Größe und Wert richtig den Steuerungen nach dem EVA-Prinzip zu.

Steuern & Regeln

Mit dem Arduino kann eine Steuerung oder eine Regelung programmiert werden. Was der Unterschied zwischen einer Steuerung und einer Regelung ist, lässt sich am Beispiel des Herdes und des Backofens erklären. Der Herd hat eine Steuerung. Für die Herdplatte kann eine Stufe zwischen 0 und 9 gewählt werden. 9 bedeutet „volle Leistung“. Stufe 4 ist dementsprechend nur ungefähr die halbe Leistung. Es kann nicht eine bestimmte Temperatur festgelegt werden. Dementsprechend muss für einen großen vollen Topf eine höhere Stufe gewählt werden, als für einen Kleinen mit wenig Inhalt. Beim Backofen ist das anders. Es kann eine Temperatur eingestellt werden. Ist der Backofen kälter als die eingestellte Temperatur, heizt er nach, bis die Temperatur erreicht ist. Ist er heißer als die eingestellte Temperatur, heizt er nicht weiter, bis die Temperatur wieder auf die eingestellte gefallen ist. Dabei ist es egal, wie voll der Backofen ist. Wurde der Backofen auf 200°C eingestellt, versucht er diese Temperatur zu halten, egal ob nur eine Pizza oder ein ganzer Braten im Ofen sind.

Aufgabe: Überlege, welche Systeme der „Challenge“ eine Steuerung und welche eine Regelung haben.

Den Arduino anschließen

WICHTIG!
Wenn du keinen Arduino besitzt, mache weiter auf der nächsten Seite!
Alle Codes kannst du auch simulieren.

ACHTUNG!
Schließe den Arduino niemals gleichzeitig per **USB-Kabel** und **9V-Batterie** an!
Das kann je nach Arduino-Board zu Schäden am PC oder am Arduino-Board führen.

Schritt 1.: Beachte folgendes Warnsymbol:

Das Warnsymbol bedeutet: **Achtung ESD-empfindliche Bauteile**
Lese den Text „Erklärung ESD“!
Um die elektrostatische Entladung unseres Körpers zu entladen „erde“ dich an der Heizung. So schützt du den Arduino vor Schäden durch ESD.

Erklärung ESD:
ESD = electrostatic discharge (deutsch: elektrostatische Entladung)
Elektrostatische Aufladung kennen wir aus dem Alltag. Wir bekommen einen „gewischt“. Dazu kommt es, wenn durch Reibung zum Beispiel von Socken auf einem Teppich wir uns statisch aufladen. Uns Menschen schaden diese kleinen Stromschläge nicht, einem elektronischen Bauteil schon. Elektronische Bauteile sind ESD-empfindlich. Grade im Winter, sind wir Menschen öfter aufgeladen. Trockene Haut und trockene Heizungsluft leiten Strom sehr schlecht und verhindern, dass die Ladung über die Haut und Luft entweichen kann. Fassen wir dann ein elektronisches Bauteil an, entladen wir uns an diesem. Der Strom kann für einen sehr kurzen Augenblick sehr groß sein. Dieser kurze Augenblick kann ausreichen, um ein Bauteil zu zerstören.

Schritt 2.: Schließe den Arduino per USB-Kabel am Computer an.

Schritt 3.: Überprüfe im „Geräte-Manager“, ob der Arduino erkannt wurde und der Treiber installiert ist.

Schritt 4.: Öffne die Arduino IDE (Entwicklungsumgebung, engl.: ***i**ntegrated **d**evelopment **e**nvironment*). Wenn du sie noch nicht installiert hast, klicke auf das Icon.

Schritt 5.: Stelle das richtige Arduino-Board unter *„Tools“* → *„Board“* ein. Der Arduino Uno wird meistens in Starter-Kits verwendet. Du hast ihn schon auf Seite 4 kennengelernt**.**

Schritt 6.: Eine USB-Schnittstelle eines Computers kann die Daten-Signale an verschiede serielle Ports weiterleiten. Deshalb müssen wir in der Arduino IDE festlegen, zu welchem Port die Signale gesendet werden. Ist der falsche Port eingestellt, kann keine Verbindung zum Arduino aufgebaut werden.
Daher wählen wir unter *„Tools“* → *„Port“* den richtigen Port. Welcher der richtige ist, können wir dem Geräte-Manager entnehmen.

Schritt 7.: Nun ist der Arduino bereit programmiert zu werden.

Erste Schritte mit Tinkercad

Schritt 1: Gehe auf www.tinkercad.com

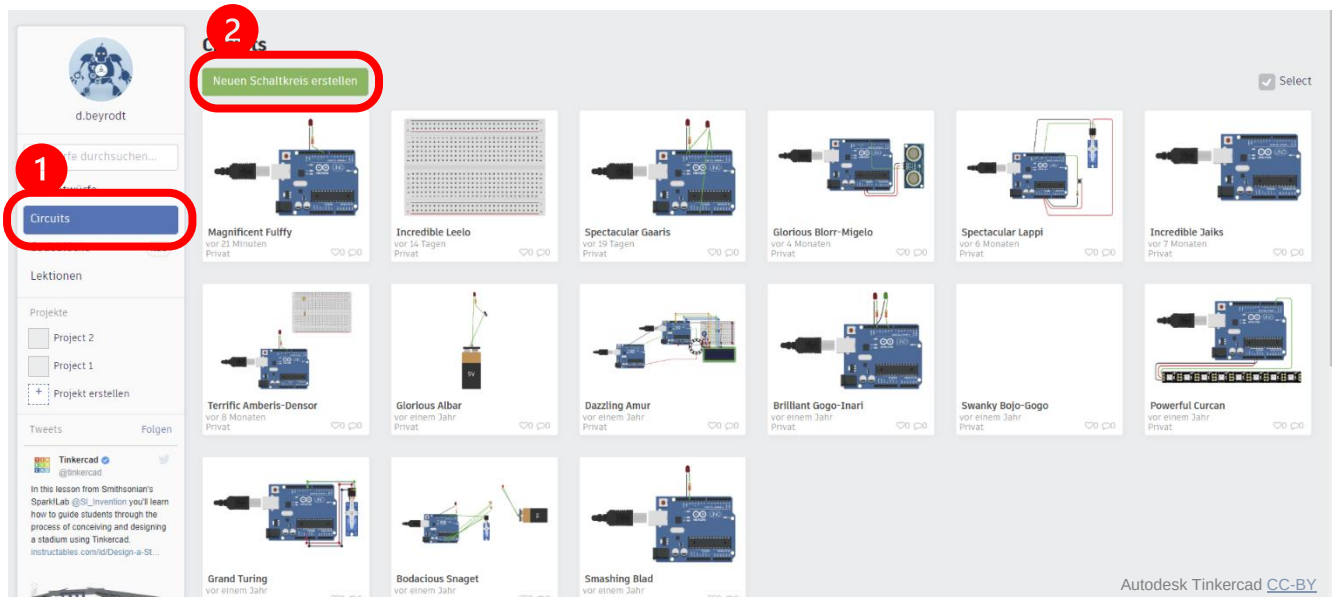
Schritt 2: Melde dich bei Tinkercad an. Wähle je nachdem was für dich zutrifft, eine der 3 Möglichkeiten:

a. Hast du schon einen Account bei Tinkercad oder möchtest du dich mit einem Account von Google, Microsoft oder Apple anmelden? → Klicke auch „Anmelden“.

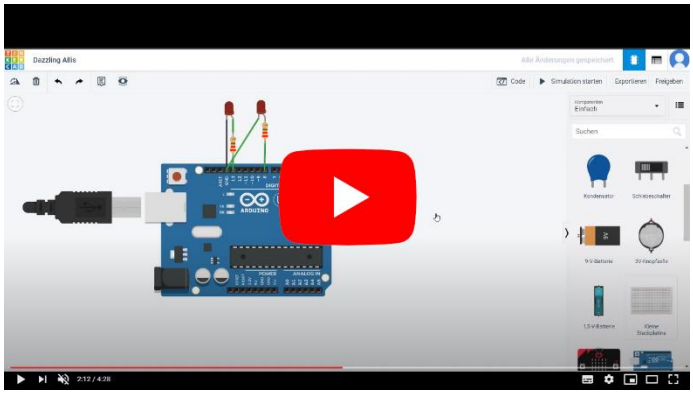
b. Deine Lehrkraft hat dir einen Klassencode und einen Spitznamen gegeben? → Scrolle runter und klicke „Klasse beitreten“.

c. Beides trifft nicht zu? → Klicke auf „Registrieren“ um ein Benutzerkonto zu erstellen. Achtung! Bist du unter 13 Jahren, müssen deine Eltern dich unterstützen.

Schritt 3: In Tinkercad kannst du neben dem Programmieren auch 3-dimensional Zeichnen (für 3D-Drucken) und vieles Weiteres machen. Um einen Arduino zu programmieren, klicke links im Menü auf „Circuits“ und dann auf „Neuen Schaltkreis erstellen“.

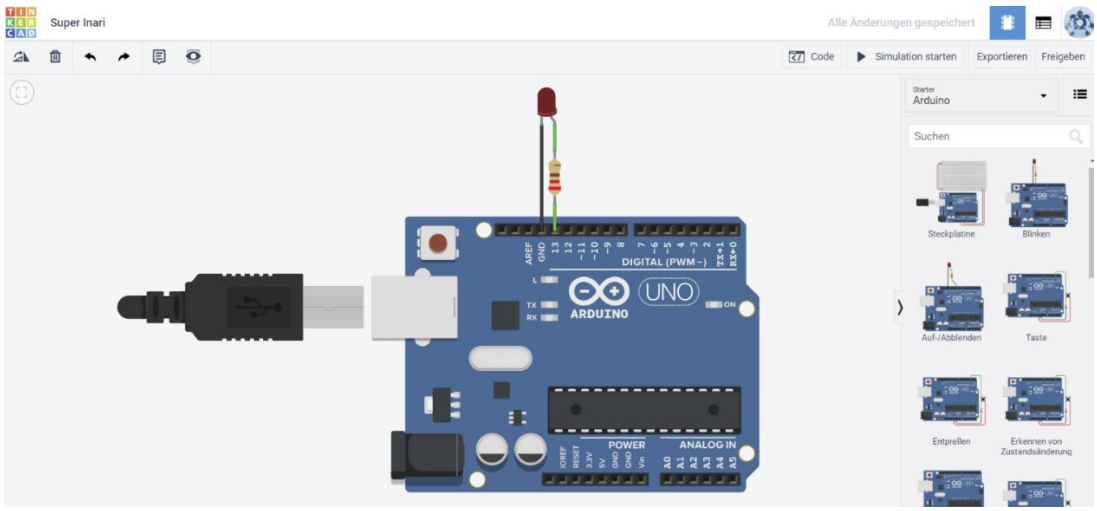


Schritt 4: Um die Bedienung von Tinkercad kennenzulernen, schaue dir dieses Tutorial an:



Eine LED (Leuchtdiode) mit dem Arduino steuern

Schritt 1: Öffne einen neuen Schaltkreis. Wähle in der Komponentenauswahl „Arduino“ aus und ziehe „Blinken“ auf die Fläche.



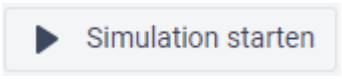
Schritt 2: Klicke auf „Code“ und wähle „Text“ aus. Du siehst jetzt folgenden Code.

Autodesk Tinkercad CC-BY

Text | 1 (Arduino Uno R3)

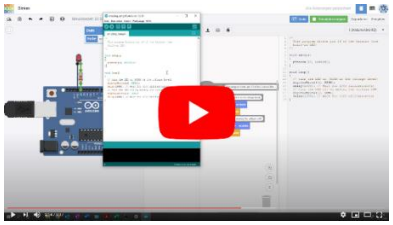
```
/*
  This program blinks pin 13 of the Arduino (the
  built-in LED)
*/

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

void loop()
{
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(13, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  // turn the LED off by making the voltage LOW
  digitalWrite(13, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Schritt 3: Klicke auf Simulation starten und das Programm wird ausgeführt. Beobachte, wie der Arduino die LED blinken lässt.



Schritt 4: Versuche durch „Try&Error“ die Geschwindigkeit des Blinkens zu verändern.

Wenn du einen Arduino zur Verfügung hast: Lade dein Programm auf den Arduino. Wie das geht, wird in folgendem Video erklärt:



Eine LED mit dem Arduino steuern

Hinweis zur LED: Eine LED braucht immer einen **Vorwiderstand**, wenn die Spannung der Spannungsquelle zu hoch ist. Der Arduino hat an den digitalen Ausgängen immer eine Spannung von 5V, wenn der Ausgang auf „HIGH“ und 0V wenn er auf „LOW“ steht. Die meisten LED können nur Spannungen von ungefähr 2V ab. Ein Vorwiderstand in Reihe zur LED geschaltet nimmt die restlichen 3V auf. Wie groß der Vorwiderstand sein muss, kann genau berechnet werden. Beim Arduino werden Vorwiderstände von 220Ω oder 330Ω verwendet. Einen genaueren Vorwiderstand zu verwenden ist nicht unbedingt notwendig für unsere Zwecke. Wir wollen ja nur nicht, dass die LED sofort durchbrennt. Teste gerne die Simulation ohne Vorwiderstand 😉
Der Wert des Widerstandes kann mit einer App sehr leicht bestimmt werden. Suche nach Apps mit „Widerstand“ oder „Resistor“. Es gibt unzählige Apps, zum Bestimmen der Widerstandswerte.
Die zweite wichtige Sache, um eine LED richtig anzuschließen, ist die Polung. Eine LED lässt den Strom nur in eine Richtung durch. Die „Anode“ ist der Pluspol (langes Beinchen), die Kathode ist der Minuspol (kurzes Beinchen).

Jetzt werden wir uns anschauen, wie die Befehle aussehen und was sie bezwecken.

Orangene Befehle = Kommentare

In jedem Sketch werden Kommentare hinzugefügt. In der Informatik gibt es sogar aufwendige Regeln, wie Kommentare aussehen müssen. So kann ein anderer Programmierer den Sketch nachvollziehen.

Du kannst auch Kommentare erstellen, indem du „//“ vor den Kommentar setzt.

Violette Befehle = Funktionen

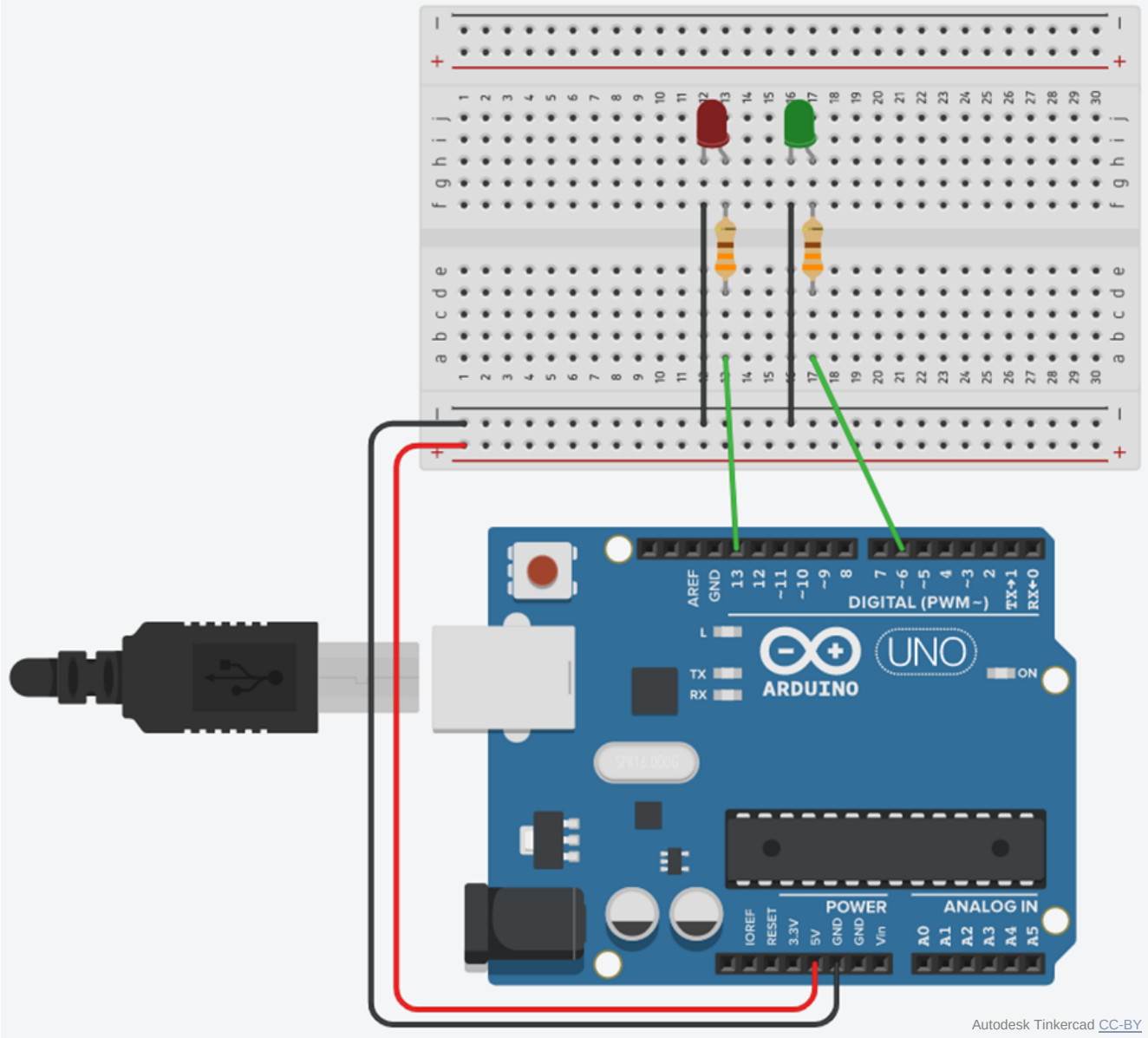
Eine Funktion beinhaltet verschiedene Befehle. Die Funktionen strukturieren den Sketch. Bei der Programmierung von Arduino-Mikrocontrollern verwendet man immer mindestens 2 Funktionen. Die „void setup()“ und die „void loop()“. Die „void setup()“ wird nur einmal ausgeführt. Die „void loop()“ danach in Dauerschleife. Eine Funktion wird durch eine geschweifte Klammer geöffnet und auch wieder geschlossen. Geschweifte Klammern erzeugst du auf der Tastatur mit „alt gr“ + 8 oder 9.

In dieser Abbildung wurde das „Blinken“ übersichtlicher dargestellt:

Text ▾ 1 (Arduino Uno R3)

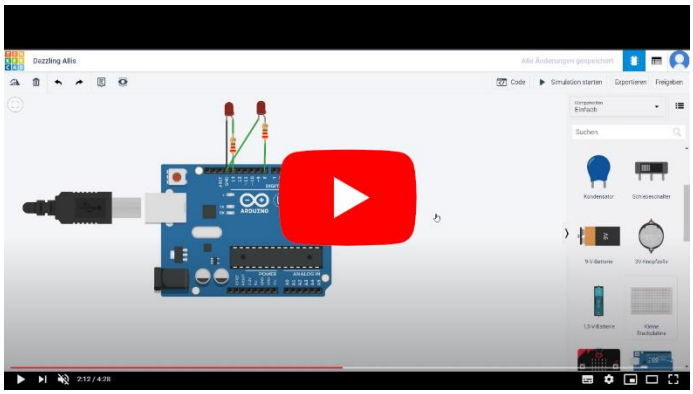
```
1
2  void setup()                  //startet Funktion "setup". Die Klammer zeigt, dass die kommenden Befehle
3  {                             //zur Funktion "setup" gehören.
4    pinMode(13, OUTPUT);        //Der Pin 13 wird als Ausgang definiert. Das Semikolon beendet den Befehl.
5  }                             //Die "setup"-Funktion wird geschlossen.
6
7
8  void loop()                   //startet Funktion "setup".
9  {
10   digitalWrite(13, HIGH);     //setzt den Ausgangspin 13 auf High, also 5V
11   delay(1000);                //wartet 1000ms, also 1 Sekunde
12   digitalWrite(13, LOW);      //setzt den Ausgangspin 13 auf Low, also 0V
13   delay(1000);                //wartet 1000ms, also 1 Sekunde
14 }                             //Die "loop"-Funktion wird geschlossen.
```

Aufgabe: Schließe eine weitere LED nach dem Schaltplan an und lasse die LED's abwechselnd blinken. Kommentiere deinen Sketch. (Hinweis: der Vorwiderstand beträgt 330 Ohm, beachte die Polung der LED)



Hinweis: GND steht für „Ground". Übersetzt heißt das „Masse". Es ist gängige Praxis, dass der Minuspol auf die Masse gelegt wird. Die Masse hat 0V. So werden die 0V für unseren Minuspol festgelegt.

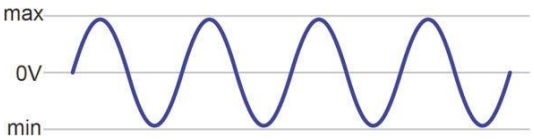
Du brauchst Hilfe, bei dieser Aufgabe? Schaue dir folgendes Video an:



Digitale und analoge Signale

„Analog“ kommt vom griechischen Wort analogos („übereinstimmend“) und bedeutet entsprechend oder vergleichend. „Digital“ kommt von lat. digitus (der Finger) und bedeutet zählend (1, 2, 3, … wie mit Fingern). In der Digitaltechnik wird daher gezählt. Und zwar keine Finger, sondern elektrische Impulse also Einsen.

Digitaler Signalverlauf



Analoger Signalverlauf

Ein digitales Signal kann nur 2 verschiedene Werte annehmen. In unserem Fall entweder 0V oder 5V. 0V nennt man auch LOW. 5V nennt man auch „1“ oder HIGH. Ein analoges Signal ist ein Spannungswert in einem bestimmten Bereich, in unserem Beispiel zwischen 0V und 5V also beispielsweise 2,8V.

Auch Anzeigen werden als digital oder analog bezeichnet. Dabei kann eine analoge Anzeige manchmal aber auch mit einem digitalen Signal angesteuert werden, eine digitale Anzeige aber nie mit einem analogen Signal.

analoge Anzeige

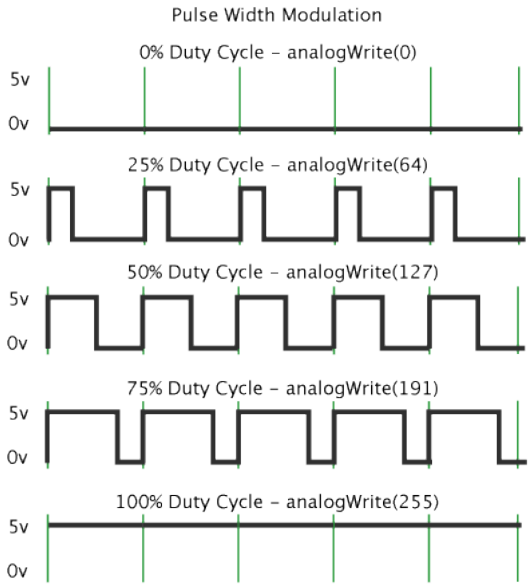
digitale Anzeige

Aufgabe 1: Ordne in der Challenge die Bauteile den beiden Signalen zu.

Challenge

Pulsweitenmodulation (PWM)

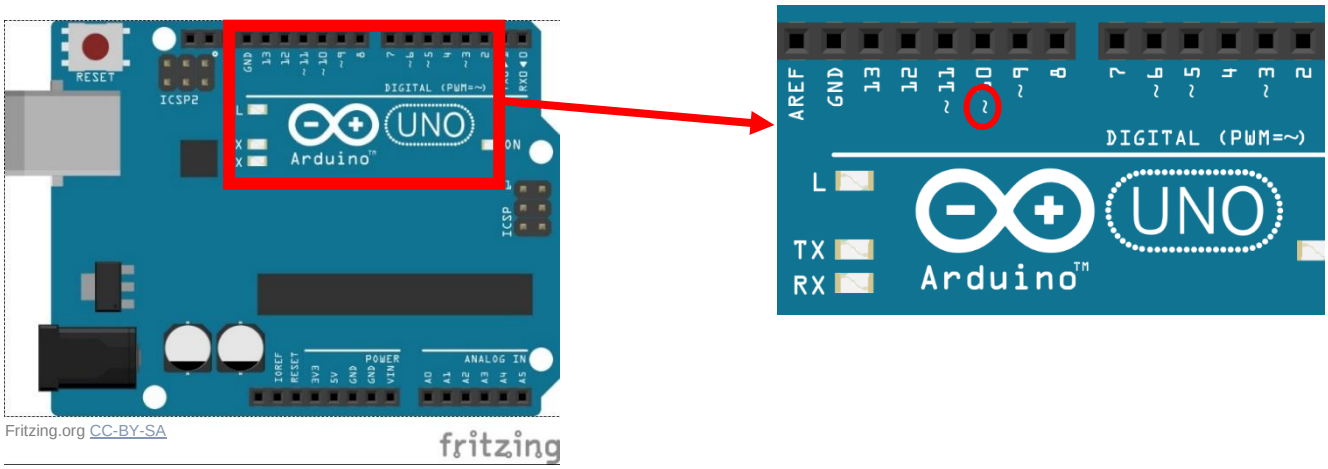
Da ein Mikrocontroller nicht selbst ein analoges Signal ausgeben kann, wird ein PWM-Signal verwendet. Bei einem PWM-Signal ist der Signalpegel eine bestimmte Zeit auf „HIGH“, also 5V. PWM steht für Pulsweitenmodulation. Rechts in der Grafik ist zu sehen, wie die Pulsweite verändert wird. Eine Zeile ist nur 1 Sekunde lang. Der Ausgangspin wird also 5x pro Sekunde an- und ausgeschaltet. Mit höherer Zahl im analogWrite Befehl, werden die Pulse immer breiter, in der der Pegel auf „HIGH“ steht. Das bedeutet, die „LOW“-Zeiten werden kürzer und die „HIGH“-Zeiten länger. Das schnelle An- und Ausschalten erzeugt eine Durchschnittsspannung. Sind „HIGH“- und „LOW“-Zeiten gleich breit, wird statt 5V eine Durchschnittsspannung von 2,5V erreicht. Auf der nächsten Seite wird das PWM-Signal genutzt, um eine LED zu dimmen.



The arduino.cc team - http://arduino.cc/it/Tutorial/PWM; CC BY-SA 3.0

LED dimmen mit PWM-Signal

Mit dem PWM-Signal kann eine LED gedimmt werden. Dazu wird eine LED an einen der „PWM-Pins“ angeschlossen. Das sind alle Pins mit einer kleinen Welle davor, wie die Abbildung zeigt (Pin 3, 5, 6, 9, 10, 11). Die Helligkeit der LED kann in 256 Schritten gesteuert werden, denn das PWM-Signal ist ein 8-bit Signal. 8-bit bedeutet 2^8 oder $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$ Einsen und Nullen.



Fritzing.org CC-BY-SA

Um eine LED per PWM-Signal zu steuern, wird nicht der Befehl „digitalWrite“ benutzt, sondern „analogWrite“. In der Klammer steht wie gewohnt zuerst der Pin. Anstelle des „HIGH“ oder „LOW“ wird bei „analogWrite“ eine Zahl zwischen 0 (kleinster Wert) und 255 (größter Wert) eingetragen.

Bei einer am Pin 3 angeschlossenen LED könnte der Sketch folgendermaßen aussehen:

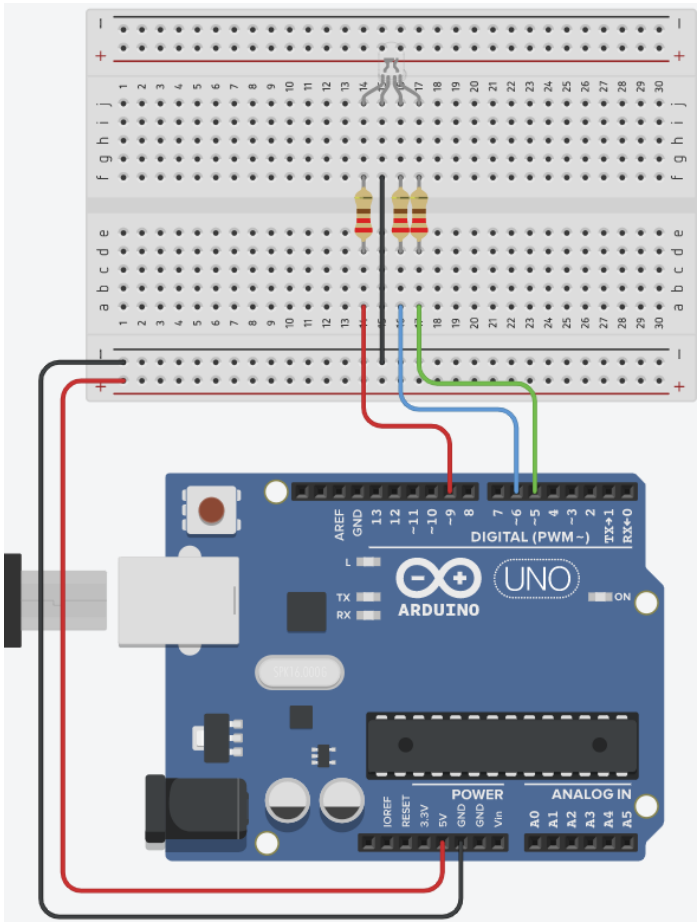
Text

```
1
2  void setup() {
3
4    pinMode(3, OUTPUT);
5  }
6
7  void loop() {
8    analogWrite(3, 255);
9    delay(300);
10   analogWrite(3, 200);
11   delay(300);
12   analogWrite(3, 150);
13   delay(300);
14   analogWrite(3, 100);
15   delay(300);
16   analogWrite(3, 50);
17   delay(300);
18   analogWrite(3, 0);
19   delay(300);
20 }
```

Aufgabe: Schließe eine LED an einem PWM-Pin an und programmiere einen Sketch, welcher die LED sanft heller und dunkler werden lässt.

Die RGB-LED

Eine RGB-LED ist eine LED, die im Prinzip 3 einzelne LED enthält. Und zwar LED in den Farben Rot, Grün und Blau. Sie hat 4 Anschlüsse. Dabei hat jede LED ihren eigenen „Pluspol“. Der vierte Anschluss ist der gemeinsame Minuspol. Wird die LED wie in der Abbildung gezeigt angeschlossen, kann eine Farbe aus den drei einzelnen (RGB)-Farben gemixt werden. Wichtig ist, dass alle 3 LED an PWM-Pins angeschlossen sind. Der längste Anschluss ist der Minuspol.



Autodesk Tinkercad CC-BY

```
Text

1  void setup() {
2    pinMode(5, OUTPUT);
3    pinMode(6, OUTPUT);
4    pinMode(9, OUTPUT);
5  }
6
7  void loop() {
8    analogWrite(5, 255);
9    analogWrite(6, 0);
10   analogWrite(9, 0);
11   delay(300);
12   analogWrite(5, 0);
13   analogWrite(6, 255);
14   analogWrite(9, 0);
15   delay(300);
16   analogWrite(5, 0);
17   analogWrite(6, 0);
18   analogWrite(9, 255);
19   delay(300);
20 }
```

Aufgabe 1: Schließe die RGB-LED, wie in der Abbildung gezeigt, an und teste sie mit dem Beispielsketch.

Aufgabe 2: Mixe die 3 Farben und programmiere einen sanften Farbwechsel.

Variablen

Infotext: Variablen sind in der Programmierung „Behälter“ für eine Größe, meistens also eine Zahl. Dieser Behälter ist ein Speicherbereich auf dem Mikrochip. Er kann mit einer Zahl oder auch mit Buchstaben beschrieben werden. Der Wert bekommt zusätzlich eine Adresse und kann so von der Maschine wieder aufgerufen oder beschrieben werden.

Um eine Variable zu nutzen, müssen wir sie „deklarieren“. Deklarieren bedeutet in der Sprache der Informatik das „Festlegen“ von verschiedenen Eigenschaften.

Es gibt verschiedene Typen von Variablen. Denn für einen digitalen Status braucht man nur eine 0 oder eine 1 abzuspeichern. Es wäre nicht sinnvoll für diesen Wert einen „Behälter“ zu wählen in den eine große Zahl wie beispielsweise „1456123678,123567“ passt.

Aufgabe: Recherchiere was für Werte die Variablen abspeichern können und vervollständige die Tabelle in der Challenge. (Hinweis Suche: „Arduino Variablen“)

Deklaration einer Variablen in Arduino:

Den Variablentyp, den wir am häufigsten für Arduino nutzen ist „int“. Die Deklaration kann innerhalb einer Funktion erfolgen, dann ist die Variable aber auch nur in dieser Funktion vorhanden. In komplizierter Software ist dies gut, um einen Überblick über die Variablen zu behalten. Variablen können auch von Funktion zu Funktion übergeben werden. Bei Arduino-Sketchen brauchen wir meistens nur wenige Variablen, daher deklarieren wir sie als „global“. Dafür müssen wir sie im Sketch einfach vor der „void setup“ definieren.

Die Deklaration sieht wie gefolgt aus:

```
int LEDwert = 255;
```

(**ACHTUNG!** Gib der Variable einen Namen ohne Leer- oder Sonderzeichen, ansonsten kann es in Tinkercad zu Fehlern kommen.)

Der Wert einer Variablen kann leicht verändert werden:

```
LEDwert = 100;
```
oder
```
LEDwert = LEDwert=LEDwert+10;
```

Nun kann sie in einem Befehl verwendet werden:

```
analogWrite(9, LEDwert);
```

Aufgabe: Programmiere ein Sketch unter Verwendung einer globalen Variablen.
(Beispiel: Die Helligkeit der LED nimmt alle 100 Millisekunden um den Wert 1 zu.)

serielle Schnittstelle

Infotext: Die serielle Schnittstelle ermöglicht die Kommunikation zwischen PC und Arduino via USB-Anschluss. Seriell bedeutet, die Bits (1-en und 0-en) werden nacheinander übertragen. Diese Kommunikation wird auf dem Arduino ermöglicht, da sie im Bootloader einprogrammiert ist. Der Bootloader ist das Programm, welches als erstes gestartet wird. Beim Arduino ist der Bootloader ab Werk einprogrammiert. Kauft man aber den Mikrochip einzeln, muss man den Bootloader per Programmiergerät auf den Mikrochip laden.

Die serielle Schnittstelle sorgt dafür, dass wir ohne Programmiergerät ein Programm auf den Arduino laden können.

Des Weiteren können wir über die serielle Schnittstelle Daten an den PC senden. Dazu müssen wir die Serielle Kommunikation starten und unter Tools in der IDE den seriellen Monitor öffnen. In Tinkercad wird der serielle Monitor direkt unter unserem Code geöffnet.

Hier ein Beispielsketch:

Text

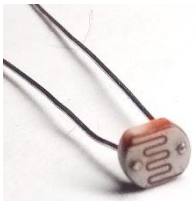
```
1  int LEDwert = 0;
2
3  void setup() {
4    Serial.begin(9600);           //startet die serielle Kommunikation
5    pinMode(9, OUTPUT);
6  }
7
8  void loop() {
9    analogWrite(9, LEDwert);
10   Serial.print("LEDWert: ");     //sende Text
11   Serial.println(LEDwert);      //sende Variable
12   LEDwert = LEDwert + 10;       //erhöht den Wert der Variablen um 10
13   delay(1000);
```

Aufgabe: Verwende die serielle Schnittstelle, um dir Werte am PC auszugeben.

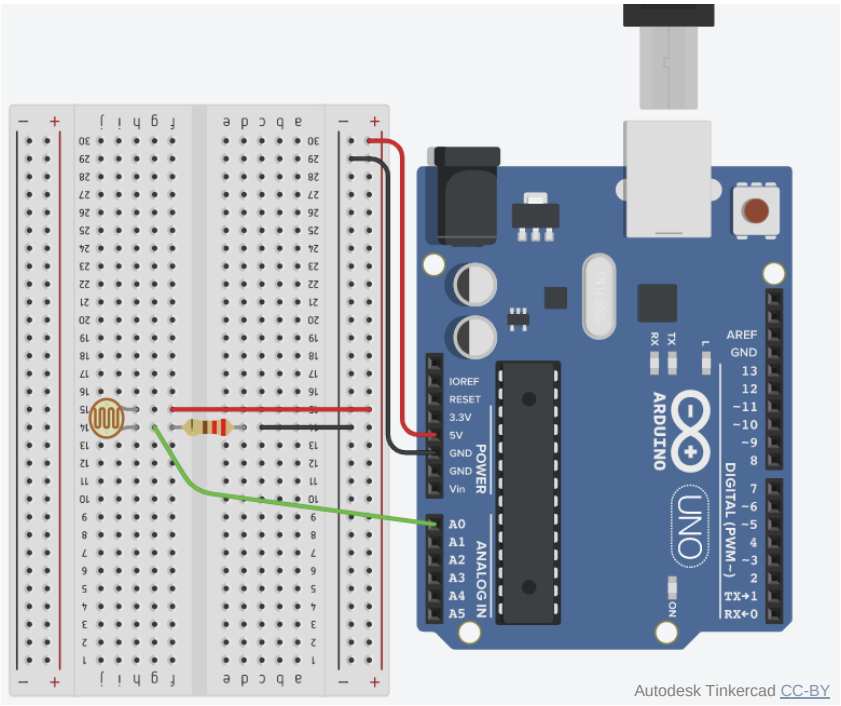
analoge Sensoren

Infotext: Analoge Sensoren können an die Pins bei „Analog In“ angeschlossen werden. Diese Ports können nur als Eingänge für analoge Sensoren genutzt werden. Sie funktionieren wie ein Multimeter, das eine Spannung misst. Viele analoge Sensoren funktionieren so, dass die physikalische Größe den elektrischen Widerstand verändert. Wird ein weiterer Festwiderstand in Reihe zum analogen Sensor geschaltet (Reihenschaltung = Spannungsteiler), so ändert sich das Verhältnis der anliegenden Spannungen bei Veränderungen des Widerstandswertes des Sensors. Diese Spannungsänderung können wir „auslesen“.

Ein Beispiel: Der Fotowiderstand (auch „LDR“ = light dependent resistor) leitet bei hoher Helligkeit besser den Strom. Die physikalische Größe „Helligkeit“ ändert den elektrischen Widerstand des Fotowiderstands. Wird es heller, leitet der LDR besser. Sein elektrischer Widerstand sinkt. Die Spannung am Fotowiderstand sinkt, die am Festwiderstand steigt.



Aufgabe 1: Baue mit dem Arduino-Kasten folgenden Schaltplan auf.



Aufgabe 2: Lese den Wert des Fotowiderstands alle 200ms aus und gebe ihn an die serielle Schnittstelle weiter. Im Folgenden werden die notwendigen Befehle genannt:

Deklaration einer Variablen:

```
int sensorWert = 0;
```

„Auslesen“ des Sensors:

```
sensorWert = analogRead(A0);
```

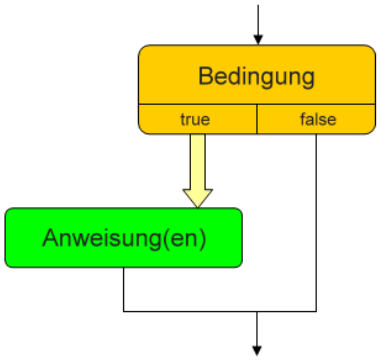
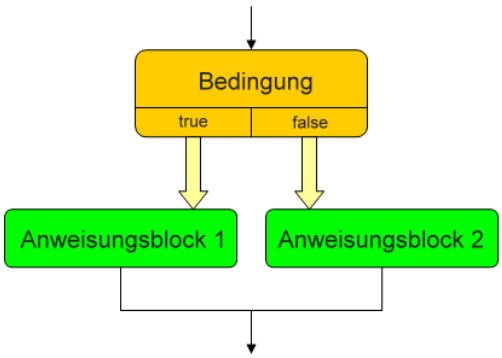
Ausgeben des Werts an die serielle Schnittstelle:

```
Serial.println(sensorWert);
```

Hinweis: Klicke in der Simulation auf den Fotowiderstand, um die Beleuchtung einzustellen.

bedingte Anweisung und Verzweigung

Infotext: Mit einer bedingten Anweisung werden ein oder mehrere Befehle des Sketches nur ausgeführt, falls die Bedingung erfüllt ist. Ein Beispiel: Falls der sensorWert ≥ 150 ist, schalte die LED an. Eine Verzweigung ist ähnlich der bedingten Anweisung. Zusätzlich werden aber noch bestimmte Befehle ausgeführt, falls die Bedingung nicht zutrifft. Im Beispiel vorhin, würde die LED immer an bleiben, sobald der sensorWert einmal ≥ 150 war. Jetzt können wir noch sagen, dass die LED ausgehen soll, wenn der Wert < 150 ist.

Bedingte Anweisung: **if**	Verzweigung **if … else**
	

```
if (sensorWert >= 150){
  digitalWrite(13, HIGH);
}
```

```
if (sensorWert >= 150){
  digitalWrite(13, HIGH);
}
else{
  digitalWrite(13, LOW);
}
```

Aufgabe: Programmiere selbst eine bedingte Anweisung oder Verzweigung.
Wenn du nicht weißt, wie du das machen sollst, hier ein Beispiel. Denke daran die „150“ an die Werte deines Fotowiderstands anzupassen. Deine Werte kannst du dir am seriellen Monitor anzeigen lassen.

Text

```
1  int sensorWert = 0;              //Die Variable für den Fotowiderstand wird deklariert.
2
3  void setup() {
4
5    pinMode(13, OUTPUT);           //Der Pin 13 wird als Ausgang definiert.
6    Serial.begin(9600);            //Der serielle Monitor wird gestartet.
7  }
8
9  void loop() {
10
11   sensorWert = analogRead(A0);   //Der Fotowiderstand wird ausgelesen.
12   Serial.println(sensorWert);    //Der Wert wird an den seriellen Monitor gesendet.
13
14   if (sensorWert >= 150){        //Bedingung für die folgende Anweisung
15     digitalWrite(13, HIGH);      //setzt den Ausgangspin 13 auf High, also 5V
16   }
17   else{                          //falls Bedingung nicht erfolgt (andere Abzweigung)
18     digitalWrite(13, LOW);       //setzt den Ausgangspin 13 auf Low, also 0V
19   }
20   delay(500);                    //Verzögerung zur Stabilisierung des Programms.
21 }
```

for-Schleife

Infotext: Soll ein bestimmter Befehl häufiger ausgeführt werden, zum Beispiel soll eine LED 10-mal ein- und ausgeschaltet werden, eignen sich dafür Schleifen. In der Programmierung gibt es verschiedene Arten von Schleifen: Kopfgesteuerte, Fußgesteuerte, Zählschleifen und Mengenschleifen. Für die Arduino-Programmierung wird die Zählschleife sehr oft verwendet. Sie wird auch for-Schleife genannt. Sie kann Anweisungen, die in ihr stehen, eine bestimmte Anzahl oft wiederholen.

Sie ist folgendermaßen aufgebaut:

```
for (Initialisierung; Bedingung; Anweisung){
  //Anweisungen
}
```

So funktioniert die Initialisierung:

In der Initialisierung wird eine Variable deklariert und ihr Startwert festgelegt. Sie ist der Zähler. Die Variable bleibt nach der Durchführung der Schleife nicht erhalten. Häufig wird die Variable einfach „i“ genannt. Der Fachbegriff für „Zähler“ lautet „Iterator“. Daher kommt das „i“.

Beispiel:

```
int i=1;
```

So funktioniert die Bedingung:

Die Bedingung legt fest, wann die Schleife ausgeführt wird. Wenn die Schleife beispielsweise 9-mal ausgeführt werden soll, lautet die Bedingung:

```
i<=9;
```

Das Symbol „<=“ ist das gleiche, wie: „≤“. Es bedeutet kleiner oder gleich. Die Schleife wird daher ausgeführt, falls die Variable (unser Zähler) „i“ eine Zahl ist, die entweder kleiner als 9 oder gleich 9 ist.

So funktioniert die Anweisung:

Die Anweisung funktioniert so, wie auch sonst Anweisungen im Programmcode.

Im Falle der for-Schleife dient die Anweisung dazu, den Zähler „i“ hochzusetzen, Sodass dieser irgendwann die Bedingung nicht mehr erfüllt und die Schleife beendet wird. Die Anweisung kann also lauten:

```
i = i+1;
```

In der Programmierung wird aber eine schönere Variante verwendet. Für dieselbe Anweisung kann Folgendes geschrieben werden:

```
i = i++;
```

WICHTIGE INFO:

= : In der Programmierung wird bei = einer Variablen ein Wert zugeordnet. Das ist eine „Zuweisung“.

== : Zwei Gleichheitszeichen stellen einen Test dar, ob 2 Werte übereinstimmen.

Beispiel für eine for-Schleife:

Im folgenden Beispiel wird eine LED 9-mal hintereinander blinken gelassen. Danach sind 3 Sekunden Pause, damit man erkennen kann, dass die Schleife beendet wurde. Denn die „void loop ()“ startet die Schleife ja automatisch wieder von vorne.

Text

```
1  void setup() {                    //startet Funktion "setup".
2   pinMode(13, OUTPUT);             //Der Pin 13 wird als Ausgang definiert.
3  }                                 //Die "setup"-Funktion wird geschlossen.
4
5  void loop() {                     //startet Funktion "loop".
6    for(int i=1; i<=9; i++){        //for-Schleife beginnt
7     digitalWrite(13, HIGH);        //setzt den Ausgangspin 13 auf High, also 5V
8     delay(200);                    //wartet 200ms
9     digitalWrite(13, LOW);         //setzt den Ausgangspin 13 auf Low, also 0V
10    delay(200);                    //wartet 200ms
11   }                               //schließt die for-Schleife
12
13   delay(3000);                    //wartet 3 Sekunden, damit zu erkennen ist,
14                                   //wann die Schleife beendet ist.
15 }                                 //Die "loop"-Funktion wird geschlossen.
```

Achtung Endlosschleife!

Eine falsch definierte Schleife, kann zu einer Endlosschleife führen. Der Arduino würde keinen Fehler melden. Der Sketch hinge lediglich in dieser Schleife fest. Ein Beispiel:

```
for(int i=1; i<=9; i=2){
    //Anweisungen
}
```

Challenge! Analysiere, welche der for-Schleifen in der Challenge eine Endlosschleife sind.

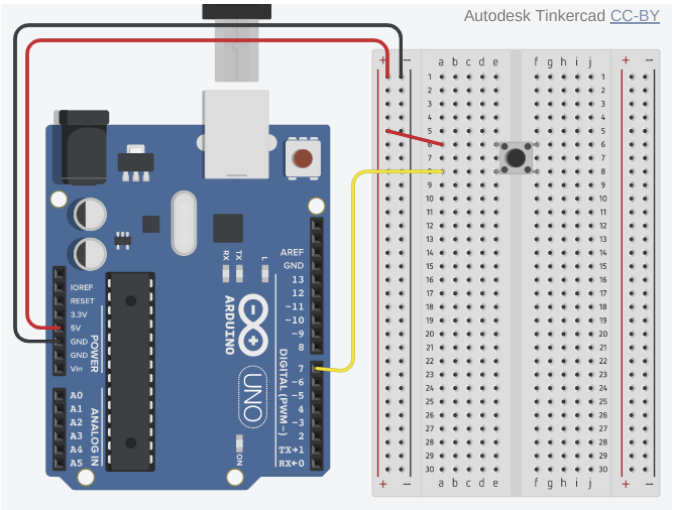
Challenge

Aufgabe: Programmiere selbst eine **for-Schleife**.

digitale Sensoren – der Taster

Infotext: Digitale Sensoren werden an die digitalen I/O Pins angeschlossen. Sie senden an den Arduino eine Zeichenfolge an Nullen und Einsen, beispielsweise so: 0110 1110 1001 1111. Diese Zeichenfolge kann dann gleich mehrere Daten enthalten. Beispielsweise Temperatur und Luftfeuchtigkeit.

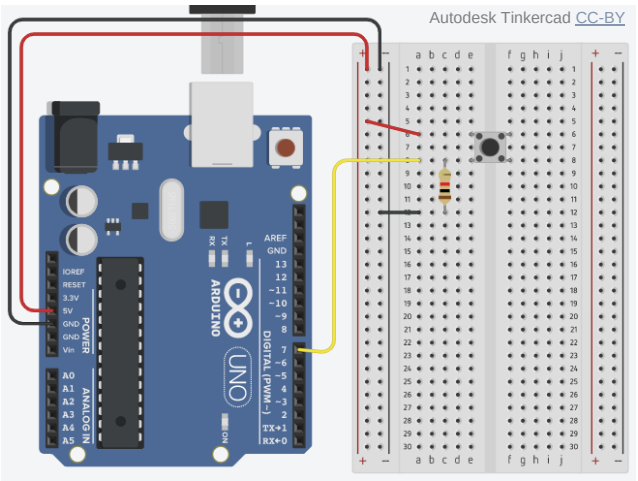
Der Taster ist ein digitaler Sensor, der den Wert 1 oder 0 haben kann (also „HIGH“ oder „LOW“). Wird der Taster gedrückt, liegt am Pin eine Spannung von 5V an. Die Schaltung sieht wie gefolgt aus:



Doch diese Schaltung hat noch 2 **Probleme**.

1.) Ist der Taster gedrückt, haben wir einen Kurzschluss, da der Taster kein „Verbraucher“ ist.
2.) Durch die elektrostatische Ladung befinden sich am Pin 7 noch Elektronen, wenn der Taster nicht mehr gedrückt ist. Der Arduino erkennt dies, als wäre der Taster noch gedrückt.

Lösung: Erdung durch einen Widerstand sodass die Elektronen abgeleitet werden. Der Widerstand muss groß sein, damit an diesem nur wenig Spannung abfällt (Denke dran: Reihenschaltung = Spannungsteiler). Da der Widerstand den Pin immer runter auf 0V „zieht“, wird er **„PULLDOWN“-Widerstand** genannt. Ein Widerstand von 1 kΩ ist ausreichend. Zudem ist dieser groß genug, damit die Schaltung keinen Kurzschluss erzeugt.
Die Schaltung sieht wie gefolgt aus:



Aufgabe: Baue die Schaltung eines Tasters auf und lassen dir den Wert des Tasters per LED oder seriellem Monitor (oder beidem) ausgeben.
Hinweis: du brauchst den Befehl in der void setup(): `pinMode(7, INPUT);`

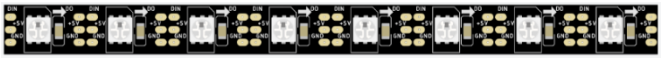
Bibliothek (Library) und Neopixel

Infotext: Eine Bibliothek (oder „library“) erweitert den Funktionsumfang in der Arduino-Software. Es werden neue Befehle geschaffen, die die Arbeit vereinfachen. Für die Nutzung eines Temperatur- und Feuchtigkeitssensors (DHT11), müsste eine lange folge Bits (Einsen und Nullen) ausgelesen und umgerechnet werden. Denn die Bitfolge gibt in einer binären Zeichenfolge Temperatur und Luftfeuchtigkeit an. Für das Auslesen und Umrechnen wird viel Code benötigt. Damit nicht jeder Nutzer diesen Code selbst schreiben braucht, ist er in einer Library hinterlegt. Für den LED-Pixel-Streifen, das Display und den Temperatur- und Feuchtigkeitssensor (DHT11) brauchen wir eine Library.

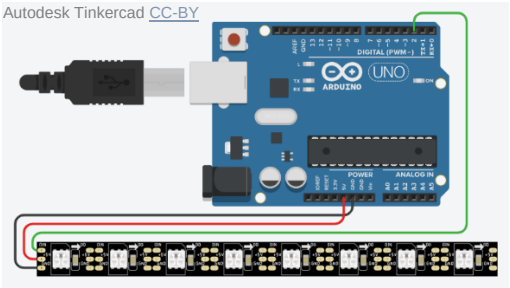
Librarys werden meistens von den Herstellern selbst veröffentlicht. Sie können in der IDE unter „Sketch“ → „Library importieren“ installiert werden. Um sie im Sketch einzubinden, kann auf die gewünschte Library geklickt, oder der Befehl selbst in den Sketch eintragen werden. Dieser gehört noch vor die Deklaration der Variablen.

Im Folgenden wird gezeigt, wie der RGB-LED-Streifen verwendet werden kann. Die Library ist schon auf dem PC installiert. Der Befehl zum Einbinden der Bibliothek lautet:

```
#include <Adafruit_NeoPixel.h>
```



Die RGB-LED-Pixel mit dem Controller „WS2812b“ werden auch Neopixel genannt. Jede LED (jeder Pixel) enthält einen Controller „WS2812b“. Dadurch kann jede LED einzeln angesteuert werden. Bei „normalen“ LED-Streifen ohne diesen Controller kann jede LED immer nur die gleiche Farbe und Helligkeit annehmen, wie alle anderen. Die Neopixel sind durch den Controller „adressierbar“. Der Streifen wird folgendermaßen angeschlossen:



ACHTUNG VORWIDERSTAND!

Wenn am Anfang des LED-Streifens kein Widerstand „R1“ aufgelötet ist, muss ein 150Ω in Reihe vor den Anschluss der Datenleitung geschaltet werden!

Folgende Neopixel-spezifische Befehle können nun verwendet werden:

Initialisierung des LED-Streifens vor der void setup() mit 8 Pixel und an Pin 2 angeschlossen:

```
Adafruit_NeoPixel LED_Streifen = Adafruit_NeoPixel(8,2, NEO_RGB + NEO_KHZ800);
```

In der void setup() wird der LED-Streifen gestartet:

```
LED_Streifen.begin();
LED_Streifen.show();
```

Nun kann jeder Pixel einzeln angesteuert werden. Für „pixel“ wird die Nummer des Pixels beginnend bei 0 (also eine Zahl zwischen 0 und 7) eingesetzt. Die drei Zahlen sind die RGB-Werte (Zahl zwischen 0 und 255)

```
LED_Streifen.setPixelColor(pixel,255 ,255 ,0 );
```

Der folgende Befehl stellt prozentual die Helligkeit ein:

```
LED_Streifen.setBrightness(100);
```

Nach jeder Veränderung der Farbe oder Helligkeit, muss folgender Befehl erfolgen:

```
LED_Streifen.show();
```

Aufgabe 1: Programmiere einen Sketch, der dem Beispiel ähnlich ist. Verändere die Farben der einzelnen Pixel nach deinen Wünschen.

Text

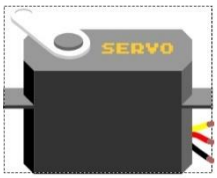
```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel LED_Streifen = Adafruit_NeoPixel(8,2, NEO_RGB + NEO_KHZ800);

void setup(){
  LED_Streifen.begin();
  LED_Streifen.show();
}

void loop(){
  LED_Streifen.setPixelColor(0, 255, 0, 0);
  LED_Streifen.setPixelColor(1, 255, 255, 0);
  LED_Streifen.setPixelColor(2, 0, 0, 255);
  LED_Streifen.setPixelColor(3, 255, 100, 100);
  LED_Streifen.setPixelColor(4, 30, 80, 0);
  LED_Streifen.show();
  delay(500);
  LED_Streifen.setPixelColor(0, 0, 255, 0);
  LED_Streifen.setPixelColor(1, 255, 100, 0);
  LED_Streifen.setPixelColor(2, 0, 0, 255);
  LED_Streifen.setPixelColor(3, 0, 100, 100);
  LED_Streifen.setPixelColor(4, 30, 80, 255);
  LED_Streifen.show();
  delay(500);
}
```

Aufgabe 1: Programmiere einen Sketch mit for-Schleifen, um die Farben für alle Pixel einfacher zu verändern.

Text

```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel LED_Streifen = Adafruit_NeoPixel(8,2, NEO_RGB + NEO_KHZ800);

int pixel = 0 ;
int zeit = 200 ;

void setup(){
  LED_Streifen.begin();
  LED_Streifen.show();
}

void loop(){

  pixel = 0 ;
  for (int i=1; i<=8; i++ ){
    LED_Streifen.setPixelColor(pixel,255 ,255 ,0 );
    LED_Streifen.setBrightness(100);
    LED_Streifen.show();
    pixel++;
    delay( zeit );
  }

  pixel = 0 ;
  for (int i=1; i<=8; i++ ){
    LED_Streifen.setPixelColor(pixel,0 ,0 ,255 );
    LED_Streifen.setBrightness(100);
    LED_Streifen.show();
    pixel++;
    delay( zeit );
  }
}
```

Servomotoren mit PWM ansteuern



Fritzing.org CC-BY-SA

Servomotoren (kurz: Servos) sind Elektromotoren, die sich je nach gewünschter Winkelposition einstellen. Dreht man am Lenkrad eines Kraftfahrzeugs, stellt sich der Winkel der Reifen ein. Aber nur, wenn das Auto eine Servolenkung hat. Heutzutage ist das der Standard. Sehr alte Autos haben keine Servolenkung. Bei ihnen wird viel Kraft benötigt, um die Position der Räder allein durch Muskelkraft einzustellen.

Beispiele für die Servomotoren verwendet werden:

1) automatisch einklappende Außenspiegel am Auto
2) Autofokus an der Digitalkamera
3) elektrische Türverriegelung

Servomotoren können auch vom Arduino gesteuert werden. Sie werden an +5V und GND angeschlossen und an einem digitalen Pin. Über diesen wird ein PWM-Signal an den Servo gesandt. Je nach Wert des PWM-Signals wird der Winkel eingestellt. Um einen Servo anzusteuern, wird die Bibliothek „Servo“ genutzt:

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

Anschließend wird ein Servo deklariert, da die Bibliothek auch mehrere Servos ansprechen könnte:

```
Servo Servo1;
```

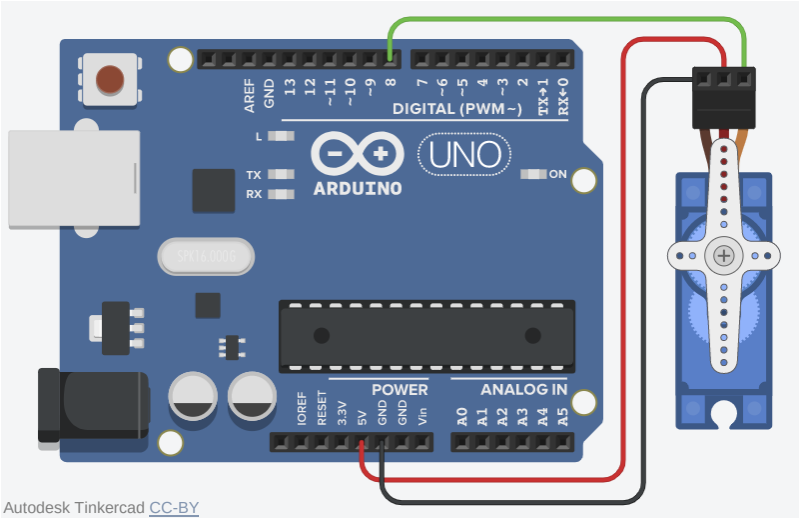
Anschließend wird in der „void setup()“ festgelegt, an welchem Pin der „Servo1“ angeschlossen ist. In diesem Fall ist er an Pin 8 angeschlossen:

```
void setup() {
Servo1.attach(8);
}
```

Nun kann der Winkel zwischen 0° und 180° in der „void loop()“ eingestellt werden. Hier ein Beispiel:

```
void loop() {
Servo1.write(0);
delay(1000);
Servo1.write(180);
delay(1000);
}
```

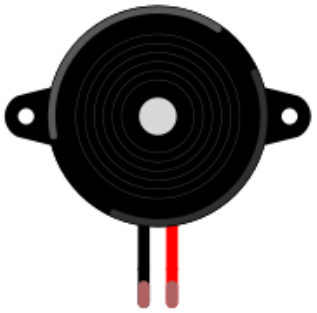
Angeschlossen wird der Servo wie in der Abbildung rechts.



Autodesk Tinkercad CC-BY

Aufgabe 2: Schließe den Servo am Arduino an und steuere ihn mit einem Sketch.

Zusatz: Ton ausgeben mit dem passiven „Buzzer“



Fritzing.org CC-BY-SA

Es gibt zwei verschiedene Bauteile, mit denen wir ganz einfach einen Ton mit dem Arduino erzeugen können: „aktiver Buzzer“ und „passiver Buzzer“. Ein aktiver Buzzer besitzt eine Elektronik, die einen Ton erzeugt. Er kann mit digitalWrite ein- oder ausgeschaltet werden. Seine Tonhöhe kann nicht verändert werden. Ein passiver Buzzer „braucht“ eine Frequenz, das bedeutet ein Signal ähnlich wie ein PWM-Signal. Je schneller die Frequenz schwingt, desto höher wird der Ton. Um den passiven Buzzer ansteuern zu können, muss er an GND und einem digitalen Pin angeschlossen werden (pinMode nicht vergessen). Um ihn zu programmieren gibt es einen speziellen Befehl:

```
tone(Pin, Tonhöhe, Dauer in ms);
```

In der Klammer wird also einmal der digitale Pin, die Tonhöhe und die Dauer des Tons angegeben. Tipp: Teste Tonhöhen zwischen 100 und 1000.

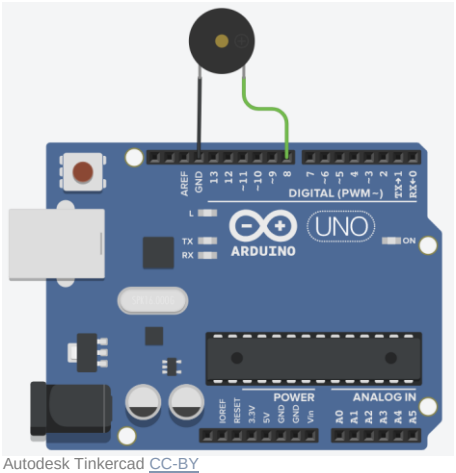
Um den Buzzer wieder auszuschalten, wird folgender Befehl genutzt:

```
noTone(Pin);
```

Aufgaben:
Erledige die Aufgaben in beliebiger Reihenfolge, oder denke dir eigene Projekte aus.

1) Programmiere eine Sirene
2) Programmiere ein Keyboard mit 3 Tastern
3) Programmiere eine Alarmanlage mit einem LDR (Schrankwächter)

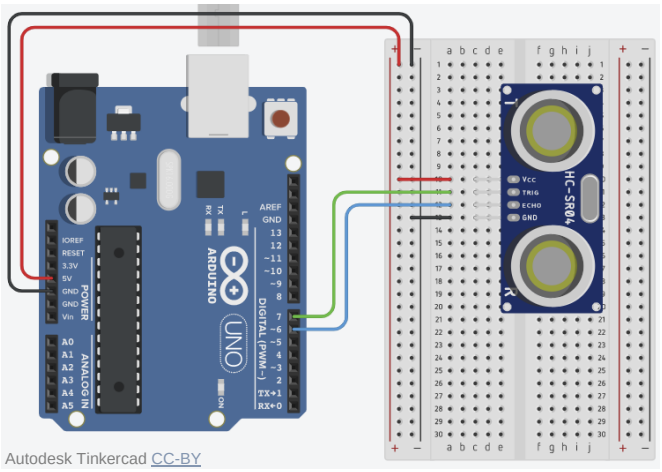
Schaltplan:



Autodesk Tinkercad CC-BY

Zusatz: HC-SR04: Ultraschallsensor (Entfernung messen)

Mit dem Sensor „HC-SR04“ kann per Ultraschallsignal die Entfernung gemessen werden. Dazu wird über den Pin „Trigger“ ein Ultraschallpegel gesandt. Diese Schallwelle trifft irgendwann auf ein Hindernis. Ein Teil der Welle prallt ab und fliegt wieder in Richtung des Sensors: das sogenannte Echo. Der Pin „Echo“ meldet uns den ankommenden Echo. Die Zeit zwischen dem Senden und dem Empfangen wird gemessen. Mit dieser Dauer können wir die Entfernung messen, da wir wissen, wie schnell die Schallwelle ist.
„Echo“ und „Trigger“ des Sensors werden an je einen digitalen Pin angeschlossen. Dazu werden noch +5V und GND an den Sensor angeschlossen.



Autodesk Tinkercad CC-BY

Programmierung:

Für den HC-SR04 legen wir zuerst Variablen für trigger und echo fest, damit es leichter wird, die Signale zu senden und zu empfangen. Dann ist es noch sinnvoll gleich 2 große Variablen für die Dauer und die Entfernung festzulegen. Dies geschieht vor dem Setup:

```
int trigger=7;
int echo=6;
long dauer=0;
long entfernung=0;
```

Im Setup werden die pinModes gesetzt und die serielle Schnittstelle gestartet:

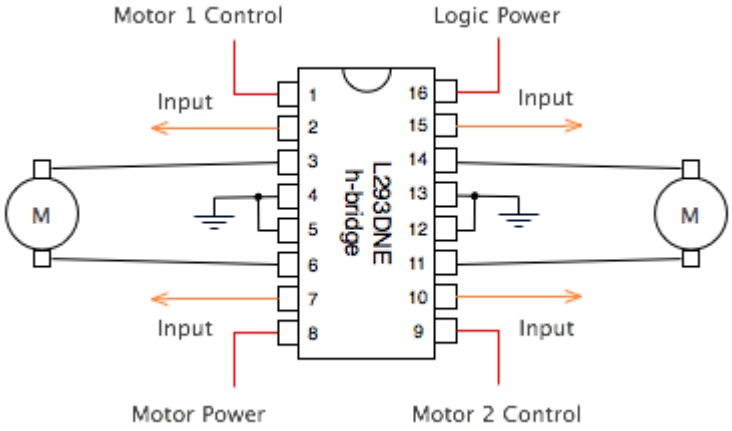
```
void setup(){
Serial.begin (9600);
pinMode(trigger, OUTPUT);
pinMode(echo, INPUT);
}
```

Anschließend wird in der Loop der Trigger kurz ausgeführt. Anschließend die Dauer des ankommenden Echos ausgelesen. Mit einer Berechnung wird die Entfernung errechnet. Danach wird die Entfernung an die serielle Schnittstelle gesandt. Ist die Entfernung zu weit, gibt es fehlerhafte Messwerte. Auch dies wird an die serielle Schnittstelle gesandt.

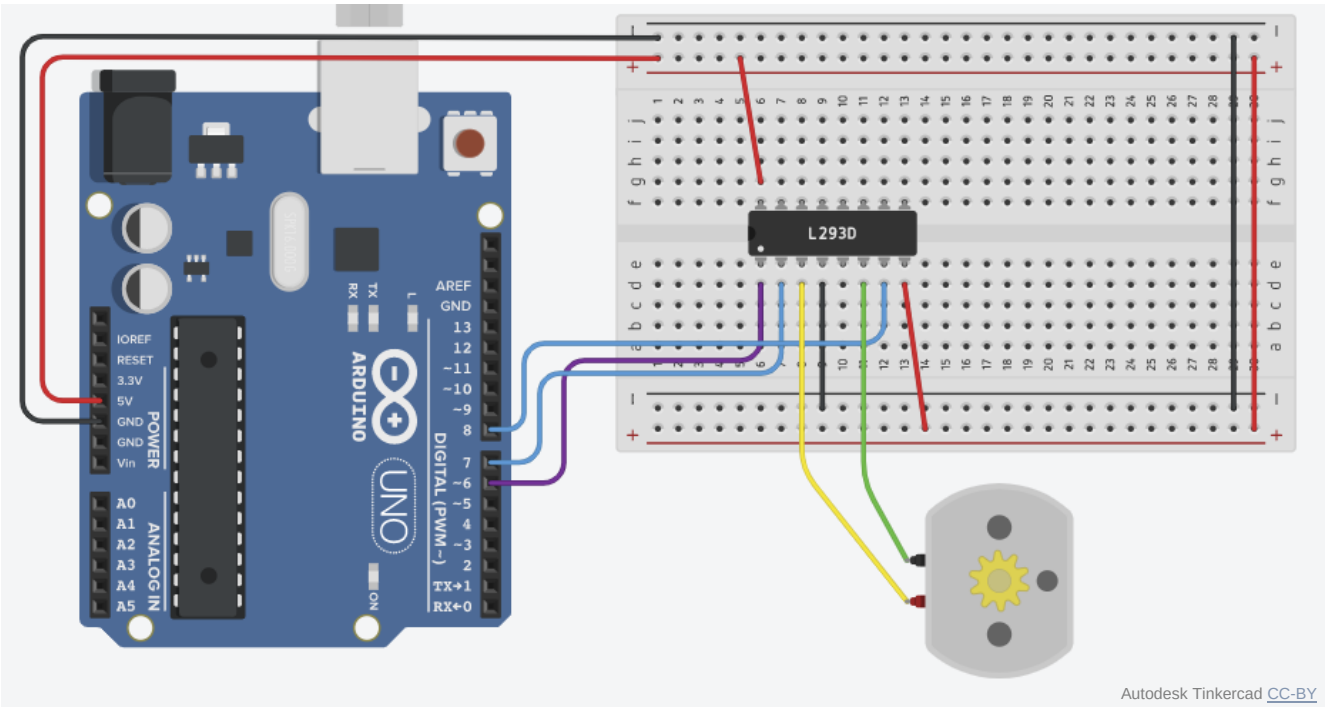
```
void loop(){
digitalWrite(trigger, LOW);
delay(5);
digitalWrite(trigger, HIGH);
delay(10);
digitalWrite(trigger, LOW);
dauer = pulseIn(echo, HIGH);
entfernung = (dauer/2) * 0.03432;
if (entfernung >= 500 || entfernung <= 0) {
    Serial.println("Kein Messwert");
}
else {
    Serial.print(entfernung);
    Serial.println(" cm");
}
delay(1000);
}
```

Zusatz: DC-Motor mit L293D Motortreiber

Um einen kleinen Gleichstrommotor oder DC-Motor (DC = direct current = Gleichstrom) am Arduino zu betreiben, ist eine H-Brücke als Motortreiber notwendig, um die Drehzahl und Drehrichtung des Motors zu steuern. Die H-Brücke „L293D“ schützt den Arduino zusätzlich vor der den Magnetfeldern, die in einem Elektromotor entstehen und zu hohen Spannungsspitzen oder Störsignalen führen können. An die H-Brücke können 2 Motoren angeschlossen werden, wie der folgende Schaltplan zeigt.



Um einen Motor an der H-Brücke „L293D“ anzuschließen, müssen „Logic Power“ und „Motor Power“ mit 5V versorgt werden. Auch muss die H-Brücke mit mindestens einem Anschluss mit GND verbunden werden. Die beiden Output-Pins 3 und 6 werden mit dem Motor verbunden. Die Polung muss nicht beachtet werden, sie entscheidet nur die Drehrichtung. Um den L293D steuern zu können, gehen 3 Anschlüsse zum Arduino. Die beiden Inputs brauchen je einen digitalen Pin. Der Anschluss „Motor 1 Control“ muss an einem digitalen PWM-Pin angeschlossen werden. Hier ein beispielhafter Schaltplan:



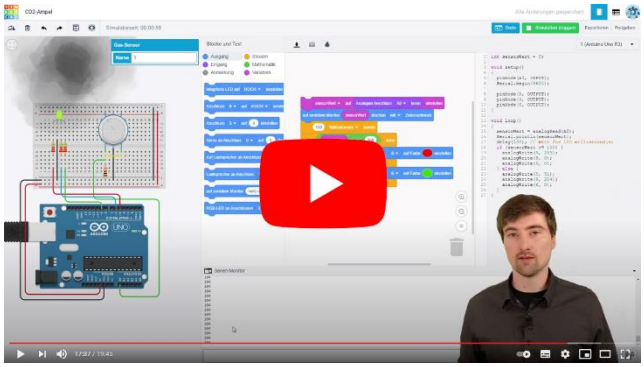
Programmierung:

Den Motor mit dem Arduino anzusteuern ist recht einfach. Per analogWrite wird die Drehzahl eingestellt. Dieser Wert ist allerdings recht ungenau. Die Drehzahl hängt nämlich auch von der Last ab. 255 bedeutet aber volle Leistung. Zudem müssen die beiden Inputs am L293D geschaltet werden. Dass heißt, sie sind Outputs am Arduino (siehe void setup). Einer der beiden Inputs wird auf HIGH, der andere auf LOW geschaltet. Vertauscht man HIGH und LOW, ändert sich die Drehrichtung. Hier ein Beispielsketch:

Text

```
int motorA=7;                         //Pin: Motor Input 1 des L293DNE
int motorB=8;                         //Pin: Motor Input 2 des L293DNE
int motorSpeed=6;                     //Pin: Motor Control

void setup(){
  pinMode(motorA,OUTPUT);
  pinMode(motorB,OUTPUT);
  pinMode(motorSpeed,OUTPUT);
}

void loop(){

  analogWrite(motorSpeed, 255);     //Motordrehzahl volle Leistung
  //Ein Motorpin muss auf HIGH und einer auf LOW damit der Motor sich dreht.
  //Werden HIGH und LOW getauscht, dreht der Motor anders herum.
  //Sind beide Pins auf LOW, dreht der Motor nicht.
  digitalWrite(motorA, HIGH);       //5V am Motorpin 1
  digitalWrite(motorB, LOW);        //0V am Motorpin 2
  delay(5000);

  analogWrite(motorSpeed, 100);
  digitalWrite(motorA, HIGH);
  digitalWrite(motorB, LOW);
  delay(5000);

  analogWrite(motorSpeed, 255);
  digitalWrite(motorA, LOW);
  digitalWrite(motorB, HIGH);
  delay(5000);

}
```

Zusatz: CO_2-Ampel mit MQ-135 Gassensor

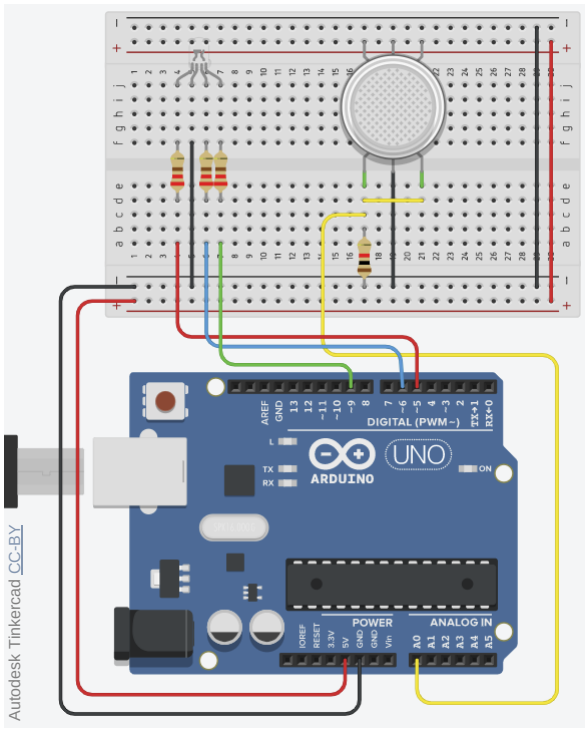


Zur folgenden Anleitung kannst du dir auch das Youtube-Tutorial anschauen.

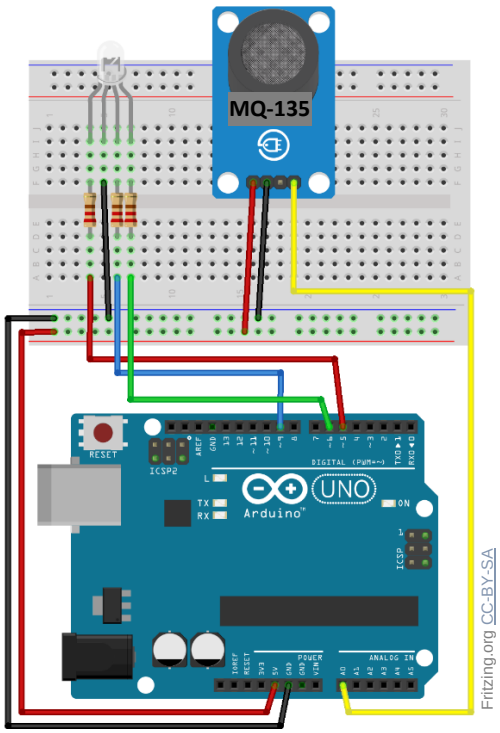
Was ist eine CO_2-Ampel? Eine CO_2-Ampel, dient dem Erkennen, wann ein Raum gelüftet werden sollte. Der CO_2-Gehalt ist dabei ein Indikator für ausgeatmete Luft. Zu viel ausgeatmete Luft ist aufgrund des geringeren Sauerstoffgehalts, Bakterien und Viren ungesund beziehungsweise unhygienisch. Es gibt Sensoren, die ein bestimmtes Gas messen. Sensoren, die nur CO_2 messen sind recht teuer. Die Verunreinigung kann aber auch mit günstigen VOC-Sensoren gemessen werden. Diese messen dann viele verschiedene Gase in der Luft. Einer dieser Sensoren ist der MQ-135. Für eine CO_2-Ampel in einem Wohnraum ist dieser ausreichend, da es kaum andere Gasquellen gibt. Eine LED kann uns anzeigen, wann die Luftverunreinigung zu hoch ist.

So wird der MQ-135 angeschlossen:
Der MQ-135 muss (wie jeder analoge Sensor) mit einem Festwiderstand in Reihe geschaltet werden. Die Reihenschaltung fungiert als Spannungsteiler. Der veränderbare Widerstand im Sensor ist ein Heizdraht, der je nach Luftverunreinigung seinen Widerstandswert ändert. Die Änderung des Widerstands sorgt in der Reihenschaltung für eine Änderung des Spannungsverhältnisses. Das kann der Arduino messen. Der Aufbau einer CO_2-Ampel mit MQ-135-Sensor und RGB-LED sieht in Tinkercad kompliziert aus. Kaufen wir einen MQ-135, so kommt er auf einem „Breakout-Board“. In diesen Fall (Bild rechts) muss der Sensor einfach an 5V (VCC), GND und mit AO an A0 angeschlossen werden:

Tinkercad:



mit MQ-135 „Breakout-Board“:



Aufgabe: Baue in Tinkercad den Schaltplan (Bild links) auf.

Programmierung des MQ-135

Der MQ-135 ist sehr leicht zu programmieren, da er ein einfacher analoger Sensor ist. Im ersten Schritt müssen wir mit der seriellen Schnittstelle uns die Werte anzeigen lassen. Dann können wir uns einen Wert für saubere Luft (gemessen direkt nach dem Lüften) und einen für verunreinigte Luft (nach ca. 15-20 Minuten mit Personen im Raum) notieren. Dieser Wert ist unser Grenzwert. Anschließend können wir Werte festlegen, zu denen unsere RGB-LED eine bestimmte Farbe annimmt. Beispiel: verunreinigte Luft (Grenzwert) = LED rot, saubere Luft = LED grün.

Der Sketch mit RGB-LED kann beispielsweise so aussehen:

Text

```
1  int sensorWert = 0;
2
3  void setup()
4  {
5    pinMode(A0, INPUT);
6    Serial.begin(9600);
7
8    pinMode(5, OUTPUT);
9    pinMode(9, OUTPUT);
10   pinMode(6, OUTPUT);
11 }
12
13 void loop()
14 {
15   sensorWert = analogRead(A0);
16   Serial.println(sensorWert);
17   delay(100); // Wait for 100 millisecond(s)
18   if (sensorWert >= 150) {
19     analogWrite(5, 255);
20     analogWrite(9, 0);
21     analogWrite(6, 0);
22   } else {
23     analogWrite(5, 51);
24     analogWrite(9, 204);
25     analogWrite(6, 0);
26   }
27 }
```

Dieser Sketch kann nun simuliert werden. Wenn er funktioniert, übertrage den Text in die Arduino-IDE und lade ihn auf den Arduino Uno. Mit dem MQ-135 als „Breakout-Board“ muss der Schaltplan wie auf der vorherigen Seite rechts aussehen.

Die CO_2-Ampel ist nun fertig, kann aber noch optimiert werden. Siehe dazu die folgenden Aufgaben.

Aufgaben:

1.) Füge eine orangene Farbe für die LED ein, für die Hälfte der Steigerung des Messwertes (zum Beispiel sensorWert ≥ 120).
2.) Füge bei Erreichen des Grenzwertes zusätzlich ein akustisches Signal mit einem passiven Buzzer (Seite 22) ein.

Zusatz: ein Relais ansteuern

Was ist ein Relais?

Ein Relais ist wie ein kleiner Schalter, der von einem elektrischen Signal gesteuert wird. Es besteht aus einer Spule und einem Schalter. Wenn Strom durch die Spule fließt, erzeugt sie ein Magnetfeld, das den Schalter im Relais umlegt. Dadurch kann das Relais einen größeren Stromkreis ein- oder ausschalten, als es normalerweise möglich wäre.

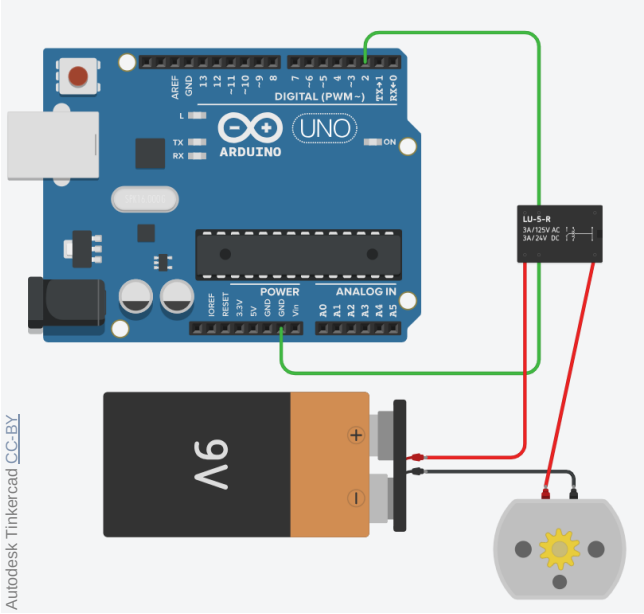
Mit einem Arduino kann ein Relais angesteuert werden, indem ein digitaler Ausgangspin verwendet wird, um Strom an die Spule des Relais zu senden. Wenn der Strom fließt, schaltet das Relais den größeren Stromkreis ein oder aus. Das ist sehr nützlich, wenn größere Verbraucher gesteuert werden sollen, für die der Arduino nicht genügend Leistung liefert, um sie zu betreiben. Als Spannungsquelle für die Verbraucher können Netzteile, 12V-Autobatterien, Modellbau-Akkus oder 9V-Batterien genutzt werden. Verbraucher können zum Beispiel Aquarium-Pumpen, Lampen, Kfz-Hupen oder Autoradios sein.

Theoretisch wäre es auch möglich mit dem Arduino und einem Relais die Lampen in der eigenen Wohnung zu steuern. Es ist jedoch verboten, am 230V-Netz selbst Arbeiten durchzuführen! Der Grund dafür sind die Gefahren, einen elektrischen Schlag zu bekommen oder einen Brand zu verursachen!

Anschließen des Relais:

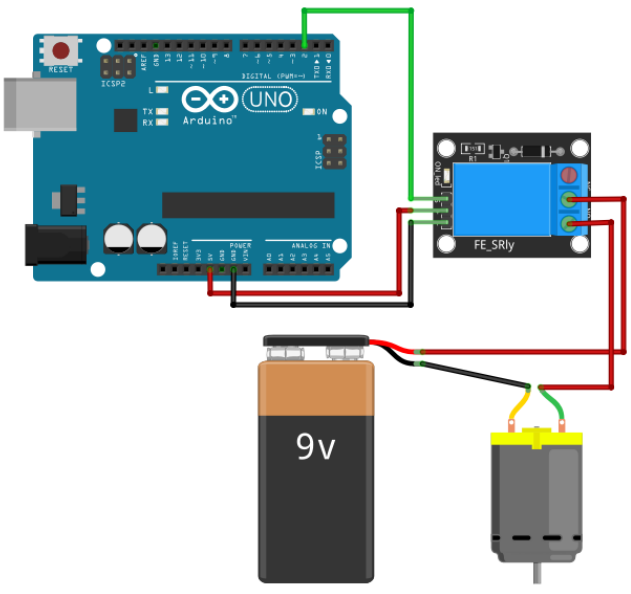
Der Schaltplan in Tinkercad ist unterschiedlich zu dem Aufbau mit den echten Komponenten. Das liegt daran, dass es die Relais auf fertigen „Breakout-Boards“ zu kaufen gibt. In beiden Schaltplänen ist die Steuerleitung grün. Als Spannungsquelle für den zu steuernden Verbraucher wurde beispielhaft eine 9V-Batterie gewählt. Ein Motor stellt beispielhaft den Verbraucher da. Die Plusleitung des Verbrauchers wird durch das Relais „unterbrochen“, denn das Relais ist ja quasi der Schalter.

Tinkercad:



Autodesk Tinkercad CC-BY

mit Relais auf einem „Breakout-Board“



Fritzing.org CC-BY-SA

Programmieren des Relais:

```
// C++ code
//
void setup()
{
  pinMode(2, OUTPUT);
}

void loop()
{
  digitalWrite(2, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(2, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Um das Relais einzuschalten, muss nur Pin auf „HOCH“ eingestellt werden, an dem das Relais angeschlossen ist. In diesem Beispiel ist es der Pin 2.

Der Sketch schaltet den Motor abwechselnd für 2 Sekunden an und aus.

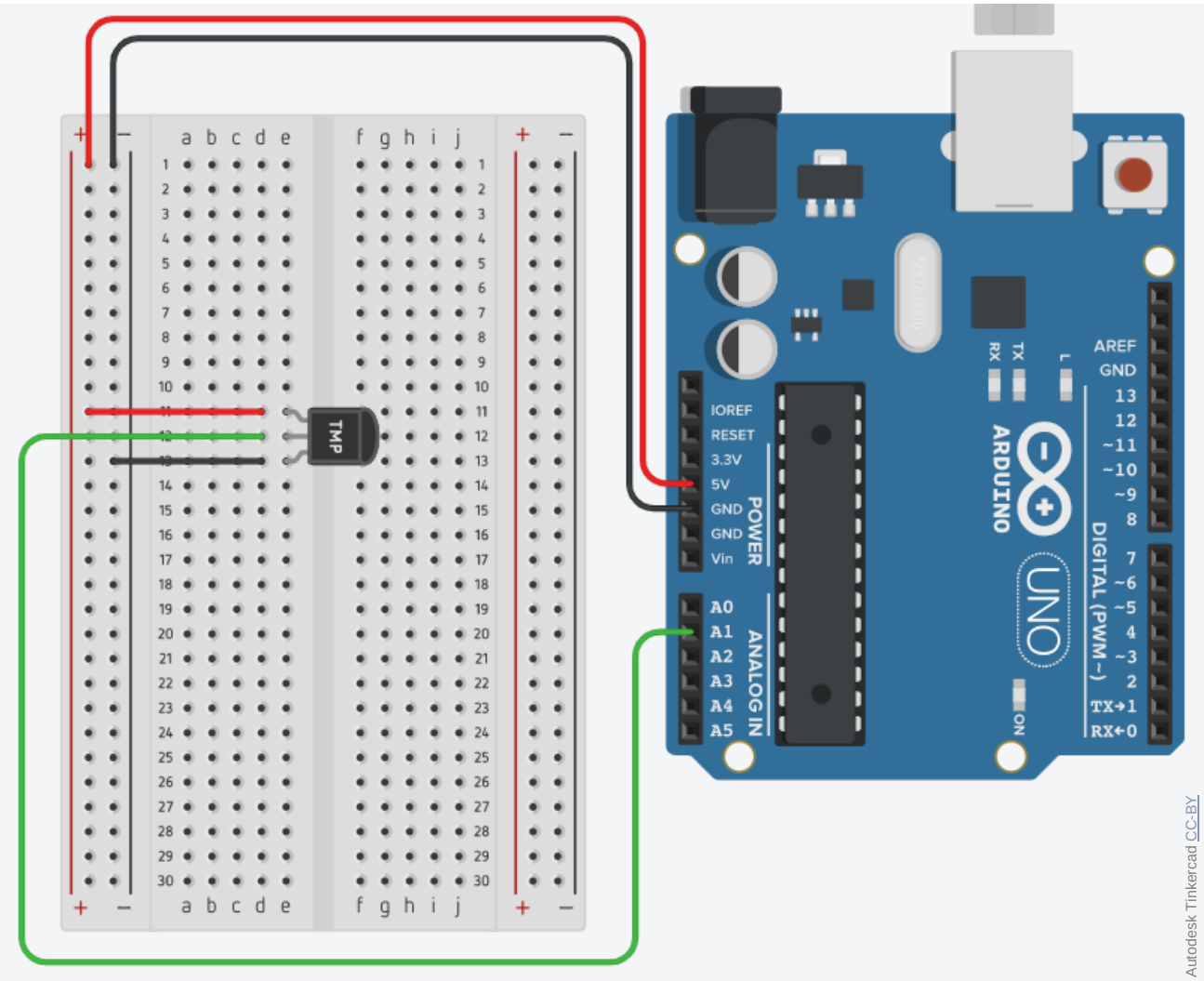
Zusatz: Temperatursensor TMP36

Der Temperatursensor TMP36 ist ein analoger Sensor. Er ist sehr einfach anzuschließen und zu programmieren. Er gibt einen analogen Sensorwert aus. Dieser kann mit „analogRead“ ausgelesen werden und eignet sich gut, wenn bei einem Schwellwert eine Aktion ausgeführt werden soll. Soll die Temperatur in °C ausgegeben werden, muss der Sensor kalibriert werden. Das ist kompliziert. Zum Glück gibt es in Tinkercad dafür einen Befehl.

Beispiel Hitzewarner:
Benötigt werden ein Gerät, um die Temperatur zu erhöhen (zum Beispiel einen Föhn), und ein Raumthermometer. Die Temperatur des Raumthermometers und des TMP36 werden gleichmäßig bis zur gewünschten Höhe des Schwellwerts erhöht. Dieser könnte zum Beispiel 30°C sein. Bei diesem Wert auf dem Raumthermometer wird nun mit der seriellen Schnittstelle der Wert des TMP36 ausgelesen. In einer bedingten Anweisung („if“) kann dieser Wert als Auslöser für eine Aktion, zum Beispiel einen Alarm, verwendet werden.

Temperatur in °C auslesen:
Mit dem Befehl „Temperatursensor an Anschluss … in Einheiten °C lesen“ statt „Analogen Anschluss … lesen“ wird der Wert in °C angegeben. Dieser Wert kann jedoch ungenau sein. Eventuell muss noch eine Differenz der Temperatur hinzugefügt oder abgezogen werden. Dafür kann das Raumthermometer genutzt werden.

Schaltplan:



Programmieren als Schwellwert (Beispiel Hitzewarner:

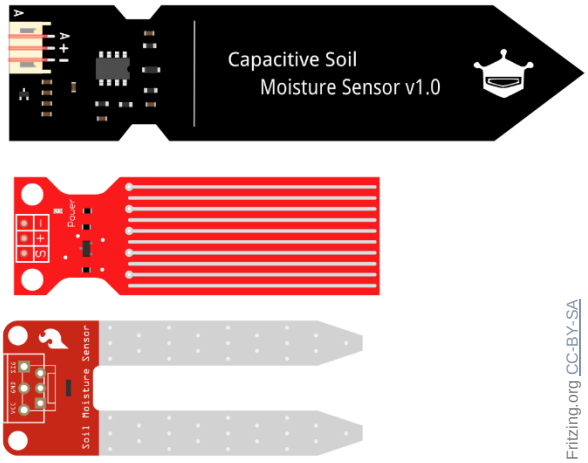
```
// C++ code
//
int sensorwert = 0;

void setup()
{
  pinMode(A1, INPUT);
  Serial.begin(9600);
}

void loop()
{
  sensorwert = analogRead(A1);
  Serial.println(sensorwert);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Programmieren als Ausgabe in °C:

```
// C++ code
//

int temperatur = 0;

void setup()
{
  pinMode(A1, INPUT);
  Serial.begin(9600);
}

void loop()
{
  temperatur = (-40 + 0.488155 * (analogRead(A1) - 20));
  Serial.println(temperatur);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

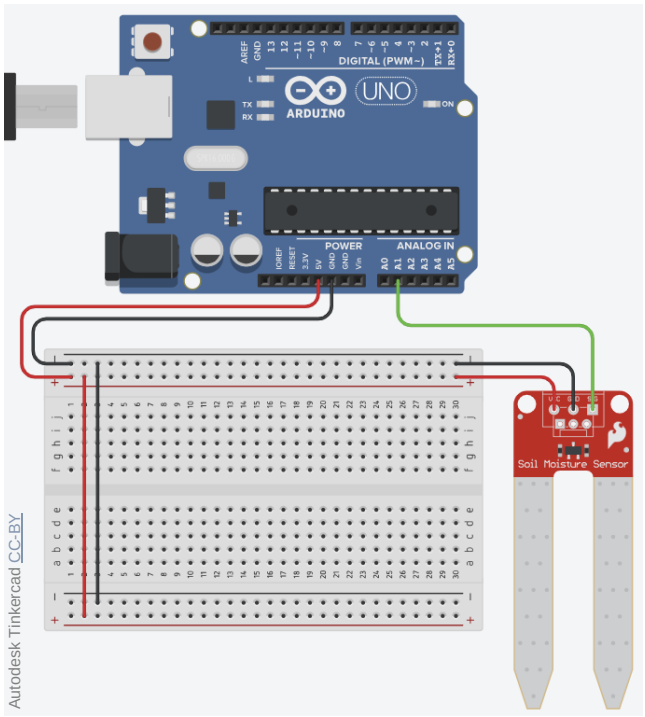
Zusatz: Feuchtigkeitssensor

Mit dem Bodenfeuchtigkeitssensor (engl.: „moisture sensor) kann gemessen werden, wie feucht die Erde (zum Beispiel in einem Blumentopf) ist. Es handelt sich hierbei um einen analogen Sensor. Sein Aussehen kann variieren. Rechts sind 3 Beispiele eines Bodenfeuchtigkeitssensor dargestellt. Angeschlossen wird er immer nach demselben Prinzip. An „VCC“ oder „+“ werden 5V angeschlossen, an „GND“ oder „-“ wird GND angeschlossen und „A“, „S“ oder „SIG“ steht für die Signalleitung und wird an einem analogen Eingang angeschlossen. Der analoge Wert kann dann über die serielle Schnittstelle ausgegeben werden.



Fritzing.org CC-BY-SA

Schaltplan in Tinkercad:



Autodesk Tinkercad CC-BY

Beispielprogramm, um Werte an der seriellen Schnittstelle auszugeben:

```
// C++ code
//
int feuchtigkeit = 0;

void setup()
{
  pinMode(A1, INPUT);
  Serial.begin(9600);
}

void loop()
{
  feuchtigkeit = analogRead(A1);
  Serial.println(feuchtigkeit);
  delay(1000); // Wait for 1000 millisecond(s)
}
```