Arduino – Digitale Elektronik

1 Grundlagen

1.1 Was sind Arduinos und Mikrocontroller?

„Arduino“ ist eine Marke für quelloffene (Open-Source) Software und Hardware. Es gibt nicht nur ein Arduino Hardeware-Produkt, sondern viele verschiedene Varianten. Wir verwenden im Unterricht den „Arduino UNO“. Alle Arduinos arbeiten mit einem Mikrocontrollern. Solche Mikrokontroller sind kleine Rechenmaschinen, die auch in vielen Alltagsgeräten stecken.

Unter der „Haube“ des kleinen schwarzen Mikrocontroller-Bausteines stecken unter anderem ein Speicher und ein Prozessor zum Bearbeiten von Rechenaufgaben drin. In deinem Handy, Laptop oder Playstation befinden sind die gleichen Bauteile – jedoch jeweils größer, leistungsfähiger, teurer und energiehungriger.

Mikrokontroller finden im Alltag vor allem eine so große Verbreitung, weil sie sehr günstig sind (wenige Cents) und nicht viel Strom verbrauchen. Da diese kleinen Bauteile jedoch nicht so leistungsstark sind, werden sie immer nur für spezielle, kleinere Aufgabe eingesetzt.

Bei einem Arduino ist alles um den Mikrocontroller herum aufgebaut, sodass man diesen bequem programmieren und verwenden kann. So lassen sich mit Arduinos leicht kreative Ideen verwirklichen. Das macht ihn ideal für Tüftler, Designer, Künstler, Bastler und nun auch für dich…

Wofür werden z.B. Mikrocontroller verwendet?

Fernbedienung
Ampelanlagen
Wetterstationen
DVD-Player
Heizungsanlagen
Motorsteuerungen
Waschmaschinen
Geschirrspüler
Roboter
Radios
Autos
Fahrstühle
Alarmanlagen
Fernseher
Ladegeräte
Bügeleisen
Lötkolben
Ferngesteuerte Autos
Überwachungsanlagen
Smart-Home Geräte
…

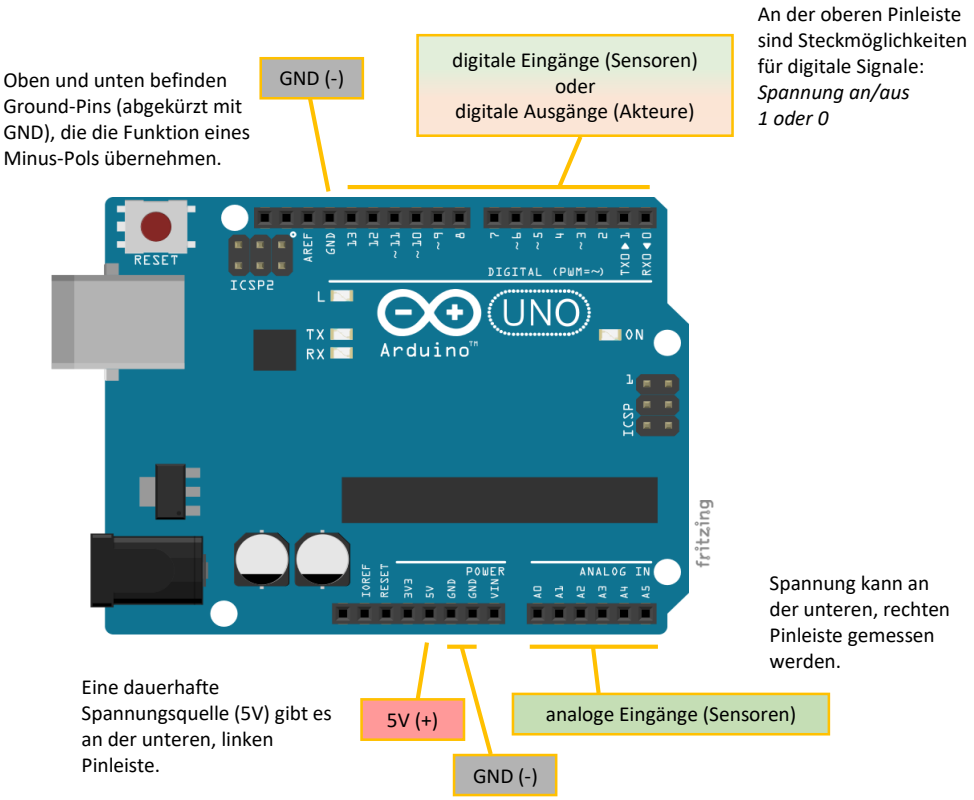
Weil Arduino-Boards quelloffen sind, dürfen diese ganz legal kopiert und nachgebaut werden. Aus diesem Grund gibt es viele weitere Boards von unterschiedlichen Herstellern, die mit dem Arduino kompatibel sind.

Abbildung 2:
Ein original Arduino UNO (ca. 30€)

Abbildung 1:
Ein Arduino UNO kompatibles Board (ca. 10€).

1.2 Steckplätze

Schauen wir uns den Arduino genauer an: Am Rand des Boards befinden sich viele Steckplätze (Pins genannt) an denen man die unterschiedlichsten elektronischen Bauteile (Sensoren oder Akteure) anschließen kann.

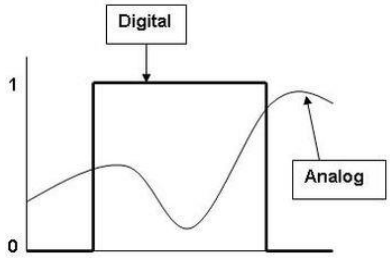


1.3 Digital und Analog

Digitale Signale kennen nur zwei Zustände:

- **An** oder **Aus**
- **1** oder **0**
- **HIGH** oder **LOW**

Beispiel: Das An- oder Ausschalten von LEDs.
Oder: Ein Taster ist entweder gedrückt oder nicht.



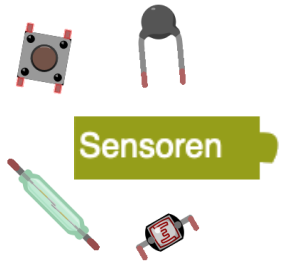
Bildquelle: https://www.edv-lehrgang.de/digitaltechnik/

Analoge Signale können unterschiedliche Werte haben:
z.B. 0V / 3 mV / 4,5 V / 5V / oder andere Werte.
Beispiel: Es gibt unterschiedliche Temperaturen, auf die ein Temperaturwiderstand reagieren kann. Auch ein Lichtwiderstand verändert den elektrischen Strom je nach Umgebungshelligkeit.

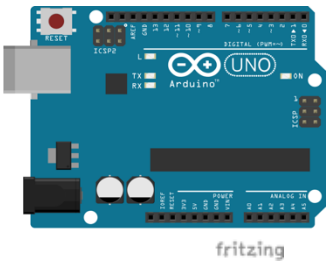
Jede Rechenmaschine (Handy, Computer, Kamera, Arduino, usw.) kann nur mit digitalen Werten umgehen. Das bedeutet, dass der Arduino analoge Signale in digitale Signale umwandeln muss. Dafür gibt es im Arduino einen integrierten Analog/Digital-Wandler (AD-Wandler). Alle analogen Spannungswerte zwischen 0V und 5V werden als Zahlen zwischen 0 und 1023 übersetzt. 2,51V wird demnach die Zahl 512.

1.4 Sensoren und Akteure (EVA-Prinzip)

Grundlage für jede Verarbeitung ist das EVA-Prinzip: Eingabe, Verarbeitung, Ausgabe.
An ein Arduino können verschiedene Sensoren angeschlossen werden. Die Werte der Sensoren werden als Eingabe erfasst. Der Microkontroller übernimmt die Verarbeitung der Eingangssignale und steuert über die Ausgabe angeschlossene Akteure.



Taster, Temperatur-Widerstand,
Licht-Widerstand, Reed-Kontakt, …

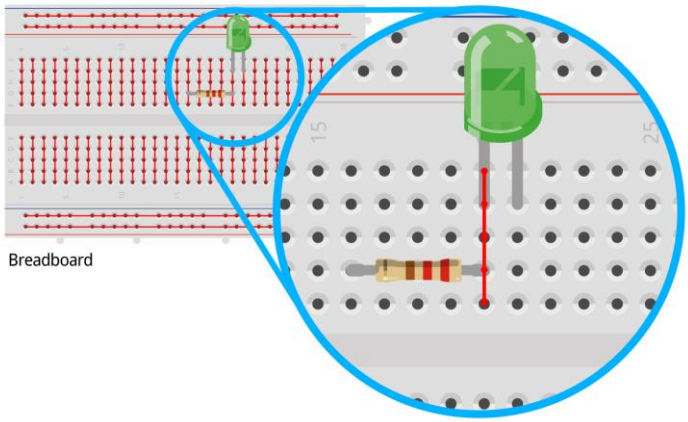


LED, RGB-LED, Summer, Servo-Motor, …

1.5 Breadboard

Für ein schnelles Testen von Schaltungen eignen sich Breadboards (Steckbretter). Dort können elektronische Bauteile schnell miteinander verbunden werden ohne löten zu müssen.

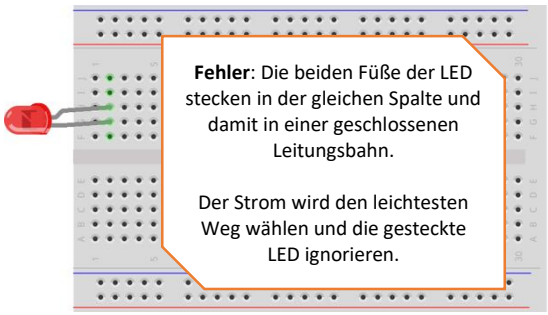
In einem Breadboard sind immer mehrere Kontakte miteinander verbunden. Auf dem Bild zeigen die roten Linien die Verbindungen zwischen den Löchern an.

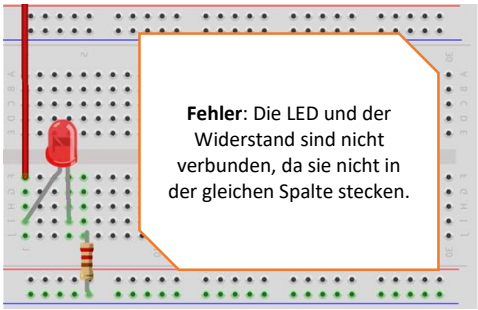


Siehst du, dass der Widerstand mit einem Beinchen in der gleichen Spalte steckt, wie die grüne LED? An dieser Stelle sind sie miteinander verbunden.

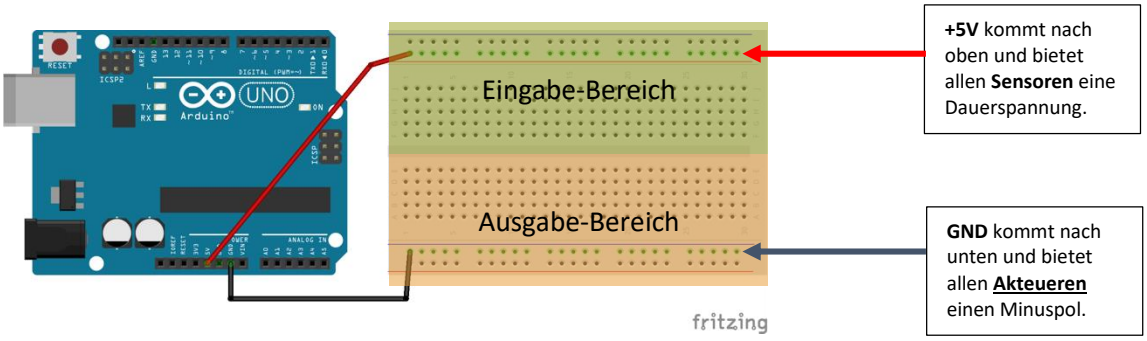
Die äußeren Steckmöglichkeiten sind im Inneren des Breadboards waagerecht miteinander verbunden. Die inneren Pins sind senkrecht miteinander verbunden. In der Mitte des Boards gibt es keine innere Verbindung.

Typische Fehler:



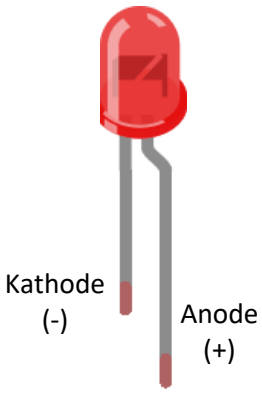


Empfohlener Aufbau:



2 Elektrische Bauteile

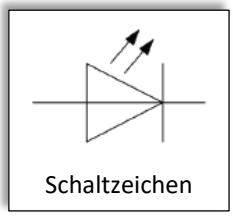
2.1 Leuchtdiode (LED)



Bei einer LED kann der Strom nur in eine Richtung fließen. Wenn die LED falsch angeschlossen wird, bleibt sie dunkel.
Die beiden Füße der LED heißen Anode und Kathode.

- Die Anode (+) hat ein längeres Bein.
- Der Fuß der Kathode (-) ist kürzer

und ist am Lampenrand abgeflacht.


Schaltzeichen

Liegt an einer LED eine zu geringe Spannung an (unter 0,7 V), leuchtet die LED überhaupt nicht. Fließt durch die LED ein zu großer Strom, brennt diese schnell durch. Aus diesem Grund wird beim Einsatz einer LED immer ein Vorwiderstand benötigt.

Der Vorwiderstand kann vor oder auch nach der LED eingebaut werden. Wenn mehrere LEDs eingesetzt werden, muss jede Leuchtdiode einen eigenen Vorwiderstand haben.

Vorwiderstand berechnen
LEDs gibt es in unterschiedlichen Farben, Stärken, Bauarten, von unterschiedlichen Herstellern und können an verschiedene Spannungsquellen angeschlossen werden. Deswegen muss der Vorwiederstand immer individuell berechnet werden. Für die LEDs, die wir im Unterricht einsetzen, gilt die nachfolgende Tabelle. Dort sind die Herstellerangaben bereits eingefügt und der Vorwiderstand ausgerechnet.

<u>Wieviel Strom muss vernichtet werden?</u>
(Vorwiderstand selbst berechnen)

$$R_{LED} = \frac{U_{Gesamt} - U_{LED}}{I_{LED}}$$

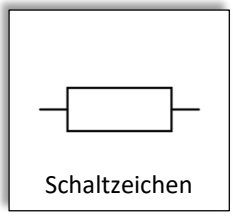
$R_{LED} = Vorwiderstand$
$U_{Gesamt} = Betriebsspannung$
$U_{LED} = Geeignete\ Spannung\ an\ der\ LED$
$I_{LED} = Maximaler\ Stromfluss\ durch\ die\ LED$

Als Vorwiderstand wird der nächst höhere Widerstand eingesetzt den man bekommen kann, da der errechnete Wert in der Regel nicht erhältlich ist.

Technologiedaten unserer LEDs	**Rot**	**Grün**	**Gelb**	**Weiß**
Betriebsspannung (Arduino)	$5V$	$5V$	$5V$	$5V$
Geeignete Spannung (LED)	$2{,}25V$	$2{,}2V$	$2{,}1V$	$3{,}2V$
Maximaler Stromfluss (LED)	$20mA$	$20mA$	$20mA$	$20mA$
Vorwiderstand, Rechnungsergebnis (für LED)	$137{,}5\Omega$	140Ω	145Ω	90Ω
Vorwiderstand, Bauteil (für LED)	$\mathbf{150\Omega}$	$\mathbf{150\Omega}$	$\mathbf{150\Omega}$	$\mathbf{91\Omega}$

2.2 Festwiderstand

Ein Widerstand begrenzt oder schwächt den Stromfluss. Ist ein Stromfluss zu stark, kann es elektrische Bauteile zerstören oder bei Leitungen zu Bränden kommen.



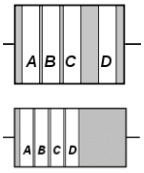
(a) Wieviel LEDs dürftest du maximal mit deinem Arduino steuern?

Aus diesem Grund benötigst du z.B. immer einen Vorwiderstand bei einer LED, damit der Stromfluss durch das Bauteil begrenzt wird. Auch der Mikrokontroller deines Arduinos hält nur einen Stromfluss von unter 200mA aus. Ein einzelner Pin sollte dabei nicht über 40 mA belastet werden. Wird dein Arduino über die Grenze hinaus belastet, zerstörst du deinen Mikrokontroller.

Die gängigsten Festwiderstände sind die Kohleschichtwiderstände (Grundfarbe: beige; Farbringe: 4) und Metallschichtwiderstände (Grundfarbe: blau; Farbringe: 5). Wie stark der Widerstand ist, kannst du an den Farbringen ablesen.

Farbringe

Widerstände kann man in beide Richtungen einbauen. Deshalb ist es auch schwierig herauszufinden, von wo man die Farbringe lesen muss. Welcher ist der erste und welcher der letzte Ring?

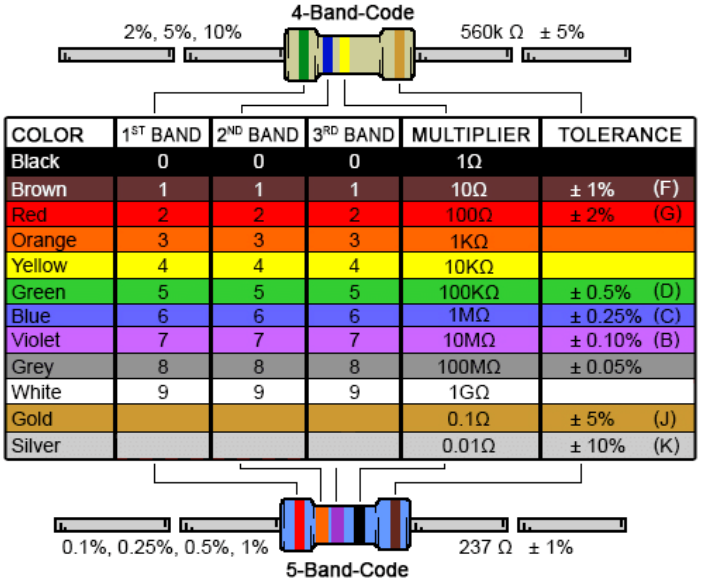


Bildquelle: https://de.wikipedia.org/wiki/Widerstand_(Bauelement)

Der letzte Ring liegt meist immer etwas abseits oder ist etwas dicker aufgemalt.

In seltenen Fällen findet man aber keinen Unterschied zwischen den Ringen. Dann sind alle Farbringe linksbündig, sodass dadurch die Leserichtung vorgegeben wird.

Wenn die Farbringe nicht gut lesbar sind, kann der Widerstandswert mit dem Messgerät herausgefunden werden



COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

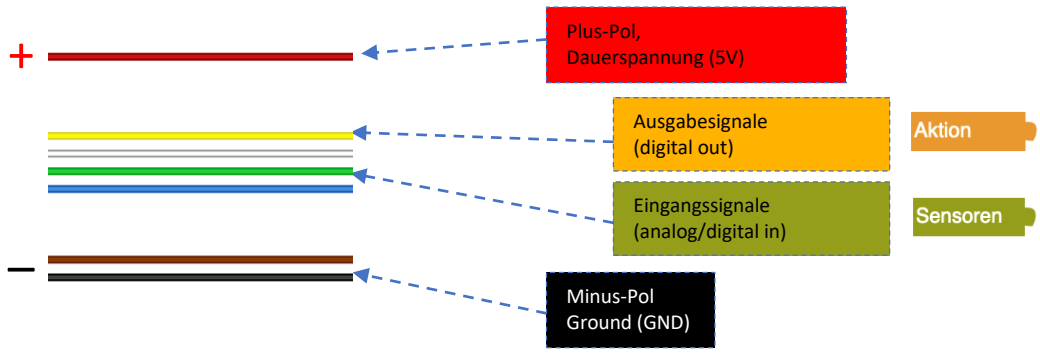
Bildquelle: https://www.digikey.de/de/resources/conversion-calculators/conversion-calculator-resistor-color-code-5-band

Wir verwenden nur Metallwiderstände und haben damit 5 Farbringe:
Die Zahlen der ersten 3 Farbringen müssen aneinandergereiht werden.
Der 4. Farbring ist der Multiplikator und gibt an, wie viele Nullen hinzugefügt werden.
Der 5. Farbring zeigt den Toleranzwert an, den die Widerstände in der Realität abweichen dürfen.

2.3 Kabel / Verbindungen

Bei Elektronikarbeiten haben bestimmte Kabelfarben immer eine Bedeutung. So lassen sich Fehler vermeiden, die einen Kurzschluss verursachen könnten. Durch die Kabelfarben erhältst du zudem einen besseren Überblick über deine Schaltung auf deinem Breadboard.

Rot ist immer der Plus-Pol und führt Dauerspannung. Schwarz oder Braun führt immer zum Minus-Pol. Die anderen Farben haben je nach Einsatzort und -art unterschiedliche Bedeutungen. Beim Arduino kannst du diese für andere Signal-Verbindungen nutzen. Im Unterricht einigen wir uns darauf, dass Gelb für Ausgangssignale und Grün für Eingangssignale reserviert ist.



2.4 Software – Blockprogrammierung

Um ein Arduino zu programmieren gibt es eine Hilfestellung durch die Blockprogrammierung.
Bei dieser Programmierung wird mit vorgefertigten Bausteinen programmiert. Die Bausteine werden wie Lego-Bausteine zusammengesetzt und ergeben zusammen ein Programm.

Programmiert wird im Webbrowser auf der Seite „lab.open-roberta.org“.
Die Verbindung zur Internetseite wird über dem Programm „Open Roberta Connector“ hergestellt.

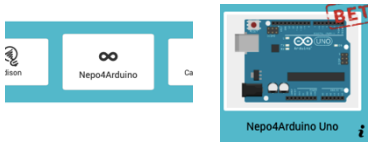
Schritt-für-Schritt Anleitung

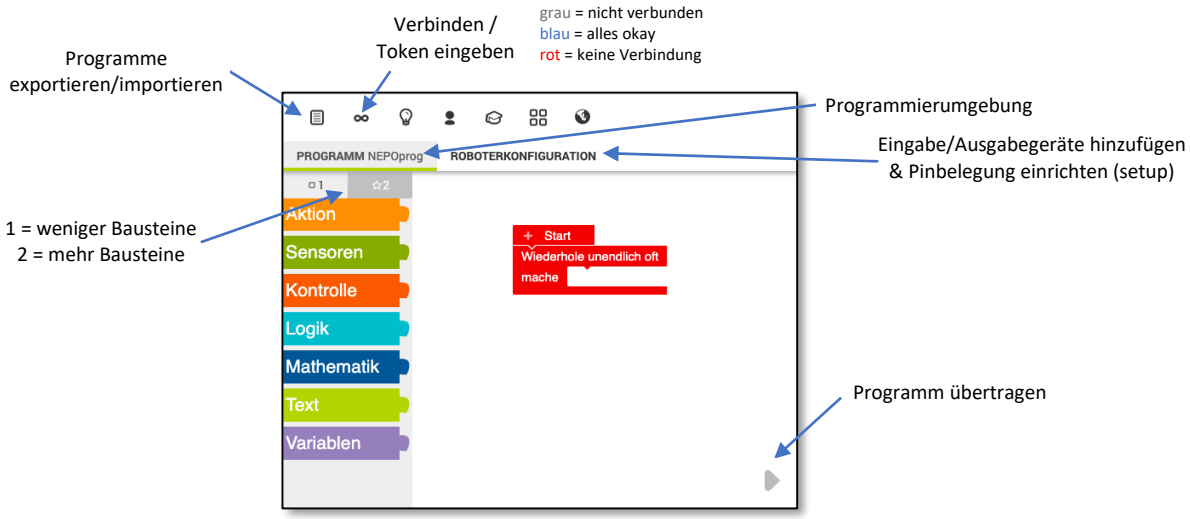
Videoanleitung
https://kurzelinks.de/connector

Einrichten

1. Schließe dein Arduino mit dem USB-Kabel an dein Laptop.
2. Öffne das Programm „*Open Roberta Connector*“
3. Wenn dein Arduino erkannt wurde, kannst du auf „*Verbinden*“ klicken.
4. Kopiere den Token.

5. Öffne die Webseite: lab.open-roberta.org
6. Wähle „*Nepo4Arduino*“ und danach „*Nepo4Arduino UNO*“ aus.
7. Lass die Webseite mit dem Programm verbinden, indem du deinen kopierten Token einträgst.





Konfiguration

8. Zunächst muss unter dem Reiter „*Roboterkonfiguration*“ ausgewählt werden, welche Ein- & Ausgabebauteile verwendet, und an welche Pins diese angeschlossen werden.

Programmieren

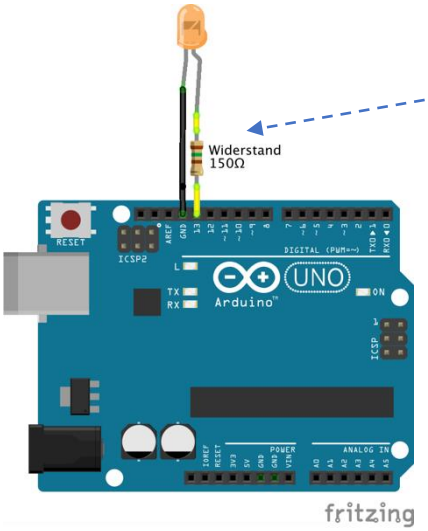
9. Anschließend kann unter dem Reiter „*Programm NEPOprog*“ dein Programm mit den Bausteinen zusammengebaut werden.

Programm starten

10. Klicke auf den „Play“-Button, um dein Programm auf dein Arduino zu übertragen.
    Die Übertragung dauert einige Sekunden und wird durch schnell blinkende Lämpchen angezeigt auf dem Arduino angezeigt.

3 Erste Programme

3.1 Blinken



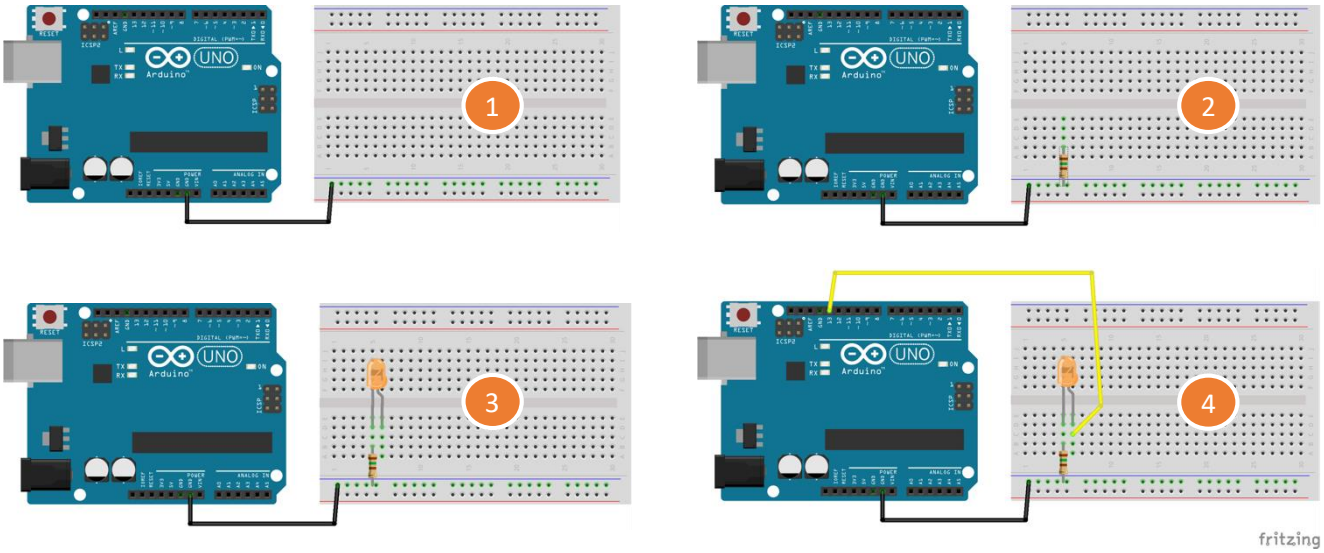
Die LED und der Widerstand sollen nicht mit der Hand zusammengehalten werden. In einem Breadboard können diese einfach verbunden werden.

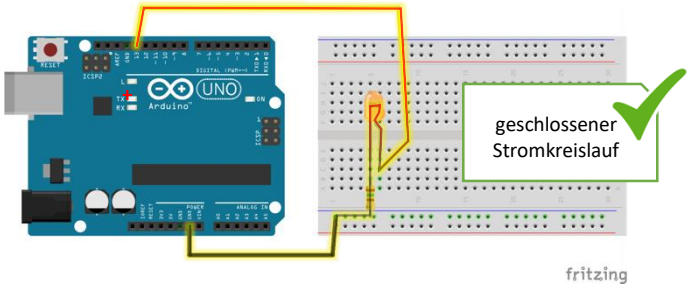
Grundsätzliches zum Aufbau:

- Baue nach Plan: Fang bei der Minus-Seite (GND) an und arbeite dich Schritt-für-Schritt zur Plus-Seite ab.
- Wer ohne Plan die Bauteile steckt, riskiert einen Kurzschluss und damit den Arduino-Tod.
- Ob der GND-Anschluss an der oberen oder unteren Seite verwendet wird, ist egal.
- Ob der Vorwiderstand vor oder nach der LED kommt, ist bei einer Reihenschaltung egal.
- Es ist egal, ob die LED in der 4 oder 5 Spalte im Breadboard steckt. Du musst nicht die einzelnen Löcher im Breadboard abzählen, sondern überlegen, was womit verbunden werden muss. Es geht darum, dass ein geschlossener Stromkreislauf entsteht.

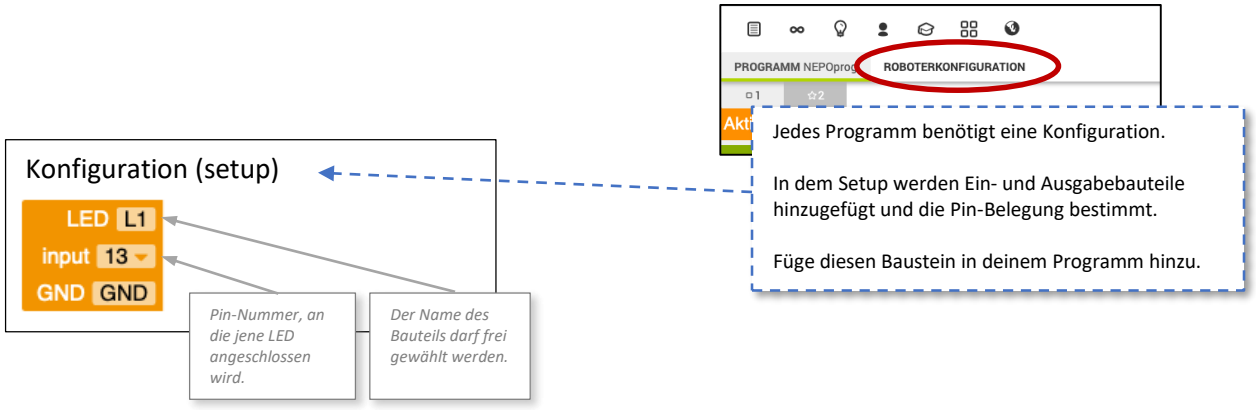
Aufgabe: Die LED blinkt. Sie wird an- und ausgeschaltet.
Bauteile: 1x LED (an Pin 13)
1x Widerstand, 150 Ω

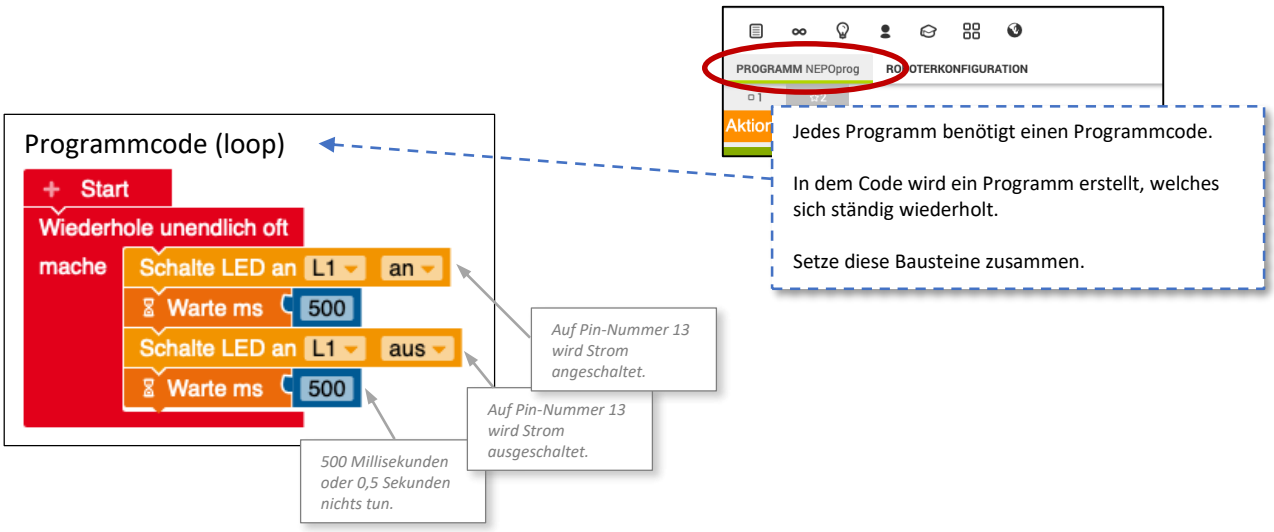
➢ AUFBAU AUF DEM BREADBOARD





➢ PROGRAMMIERUNG





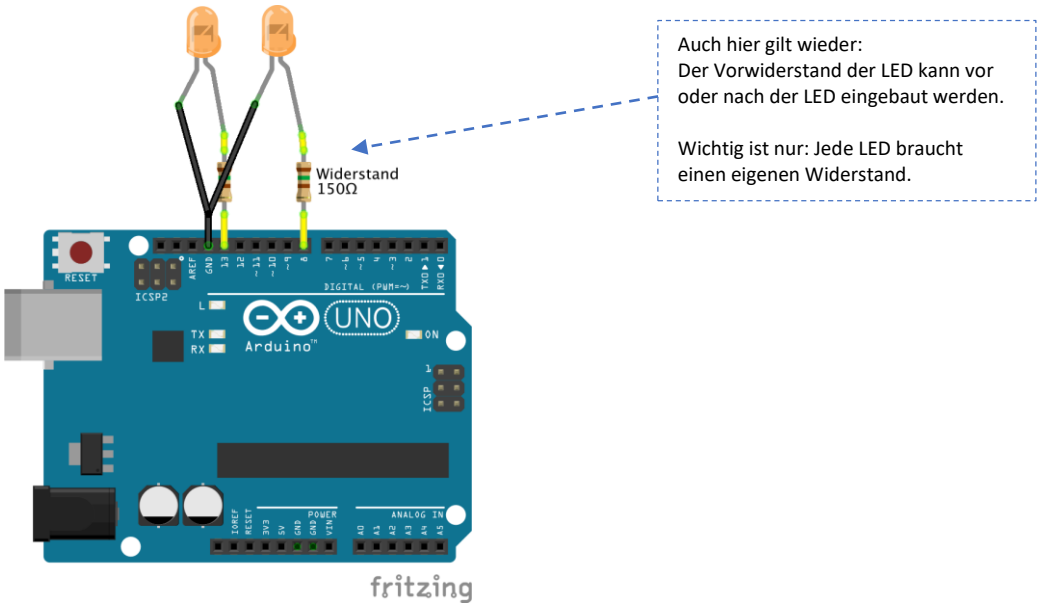
Textprogrammierung
(für Fortgeschrittene)

```
int led_L1 = 13;

void setup()
{
   pinMode(led_L1, OUTPUT);
   pinMode(led_L2, OUTPUT);
}

void loop()
{
   digitalWrite(led_L1, HIGH);
   delay(500);
   digitalWrite(led_L1, LOW);
   delay(500);
}
```

Die Programmierblöcke werden immer für den Arduino in Textprogrammierung übersetzt.
Hier kannst du sehen, wie die Code-Blöcke in der Arduino-Programmiersprache C/C++ aussehen.

Wer in Programmierungen schon fortgeschritten ist oder sich mit Arduino-Code gut auskennt, kann auch direkt in Textform programmieren. Verwende dazu die installierte Arduino-IDE.

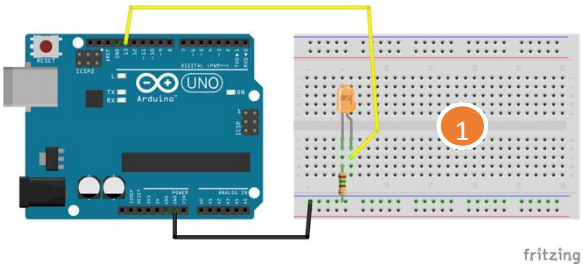
3.2 Wechselblinker



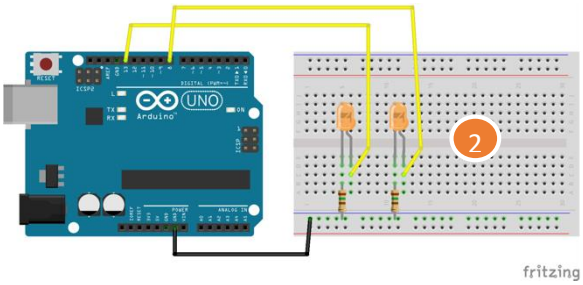
Aufgabe: Die beiden LEDs sollen immer im Wechsel an- und ausgeschaltet werden.
Bauteile: 2x LEDs (an Pin 13 und 8)
2x Widerstände, 150 Ω

➢ AUFBAU AUF DEM BREADBOARD

Deine vorherige Schaltung hatte nur eine LED und sollte so aussehen:



Erweitere deine vorherige Schaltung mit der zweiten LED und Widerstand.
Nutze dabei den gleichen Ground-Anschluss.

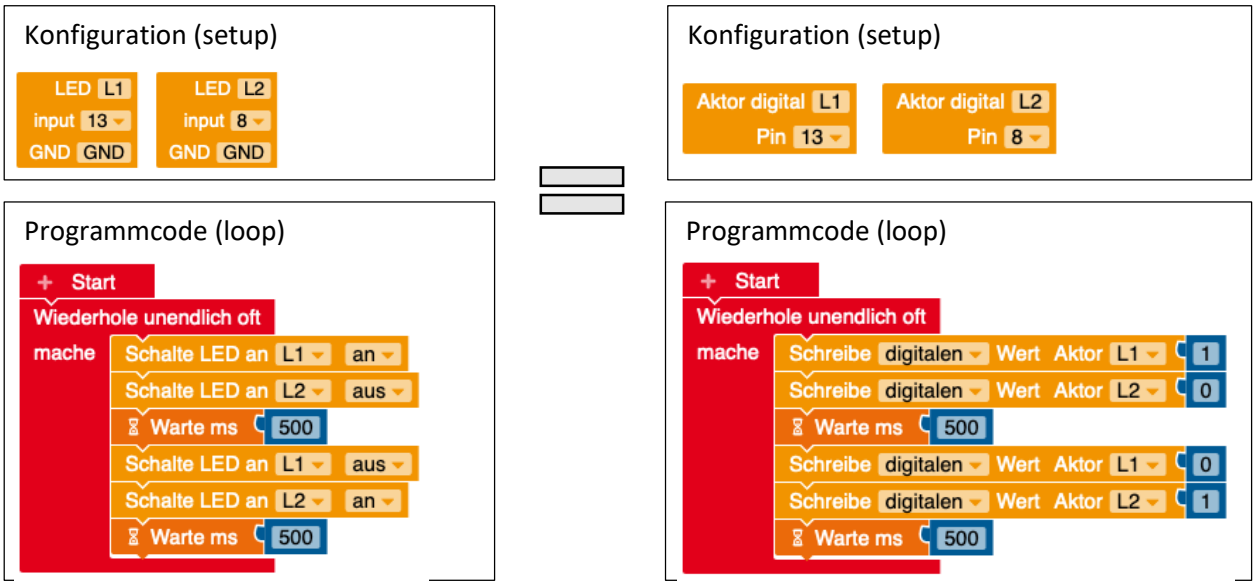


➢ PROGRAMMIERUNG

Open-Roberta hat sich entschieden, dass die einen extra LED-Baustein anbieten, da es für Anfänger und Grundschüler einfacher ist dies zu verstehen.

Eine LED ist jedoch immer ein digitaler Aktor (Ausgabe).
Deshalb machen beide Bausteine und beide Programmierungen genau das gleiche. Probiere einmal beides aus.

Hinweis:
*Digital heißt: **1** oder **0** / **an** oder **aus** / **high** oder **low***

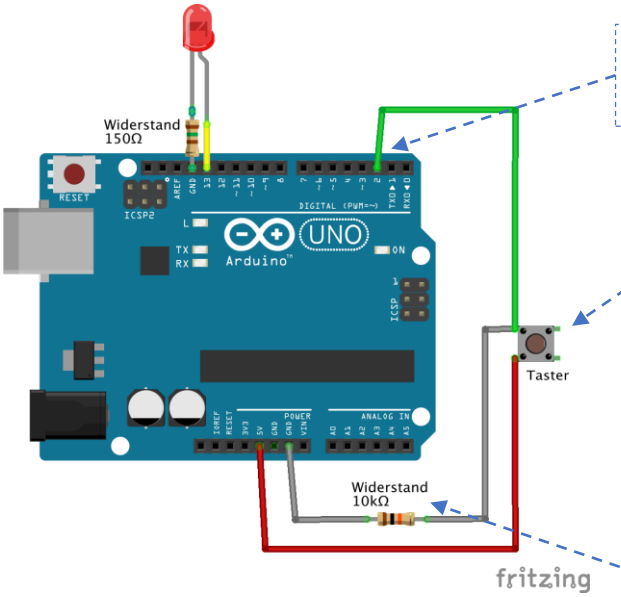


Textprogrammierung
(für Fortgeschrittene)

```
int led_L1 = 13;
int led_L2 = 8;

void setup()
{
   pinMode(led_L1, OUTPUT);
   pinMode(led_L2, OUTPUT);
}

void loop()
{
   digitalWrite(led_L1, HIGH);
   digitalWrite(led_L2, LOW);
   delay(500);
   digitalWrite(led_L1, LOW);
   digitalWrite(led_L2, HIGH);
   delay(500);
}
```

Beide Varianten der Block-Programmierung sehen als Text genau gleich aus.

3.3 Taster (einfach)

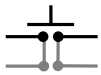


An Pin 2 wird gemessen, ob Strom ankommt (Input / Eingang).

Wird der Taster nicht gedrückt, ist die Leitung durchtrennt.

Bei gedrücktem Taster ist die Leitung geschlossen.

Hinweis:
Die Füße, die weit voneinander liegen, sind immer miteinander verbunden.
Geschaltet werden diese Füße, die näher zusammen liegen.



Ein sehr hoher Widerstand wird parallel zu dem Eingangssignal (Input) geschaltet.

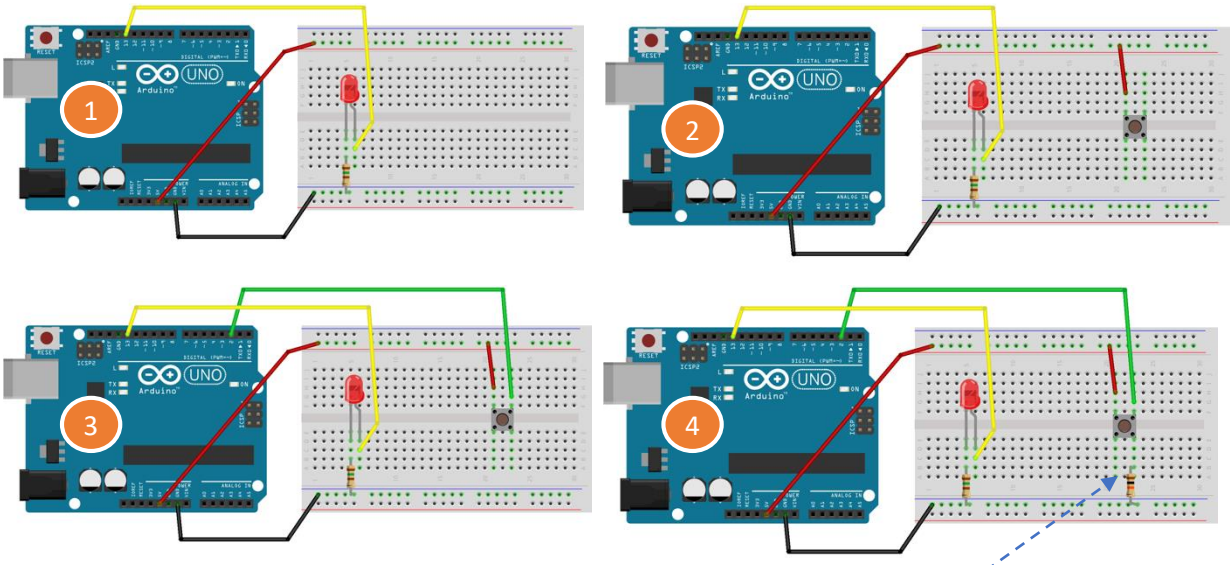
Begründung:
Die Eingangsleitung muss sich immer entladen können, wenn der Taster losgelassen wird.
Der Arduino ist so sensibel, dass ansonsten der Zustand des Tasters nicht immer zuverlässig erkannt werden kann.

Man nennt den Widerstand deshalb „Pull-Down Widerstand“, weil er die restlichen Spannungsteilchen auf Masse zieht.

Aufgabe: Die LED leuchtet, wenn der Taster gedrückt wird.
Wird die Taste losgelassen, geht die Lampe aus.

Bauteile:
1x LED (an Pin 13)
1x Widerstand, 150 Ω
1x Taster (an Pin 2)
1x Widerstand, 10 kΩ

➢ AUFBAU AUF DEM BREADBOARD



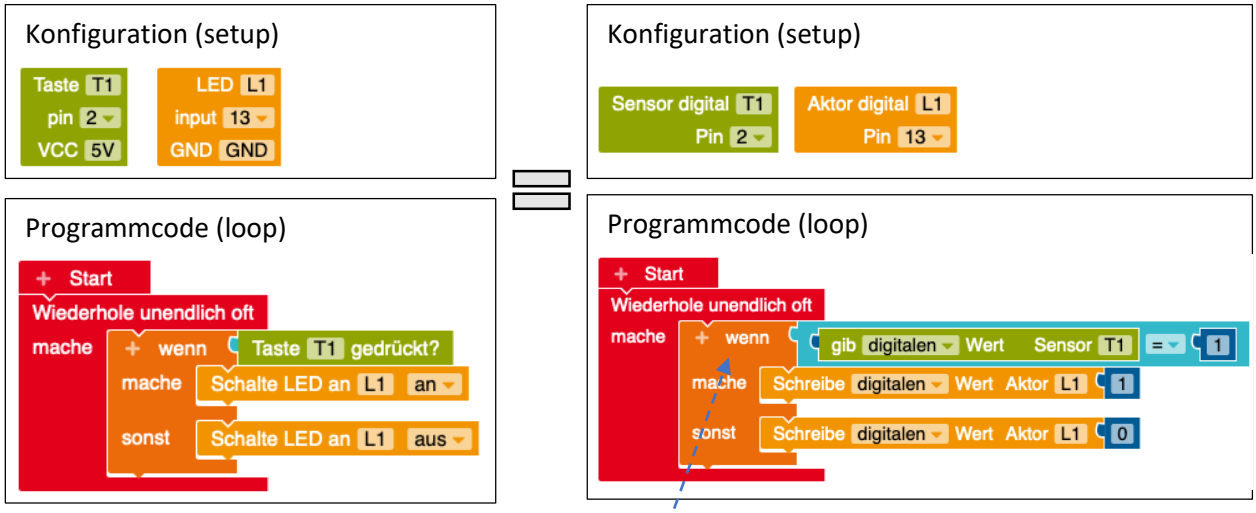
Achte darauf, wo der Widerstand hinkommt. Der Widerstand liegt parallel zum Eingangssignal (grüne Leitung). Über den Taster sind diese dauerhaft miteinander verbunden.

➢ PROGRAMMIERUNG

Genauso wie bei den LEDs hat sich Open-Roberta dazu entschieden, dass die einen extra Taster-Baustein anbieten, da es für Anfänger und Grundschüler einfacher ist dies zu verstehen.

Eine Taster ist jedoch immer ein digitaler Sensor (Eingabe).
Deshalb machen beide Bausteine und beide Programmierungen genau das gleiche. Probiere einmal beides aus.

Hinweis:
Digital heißt: ***1*** *oder* ***0*** */* ***wahr*** *oder* ***falsch*** */* ***Strom*** *oder* ***kein Strom***



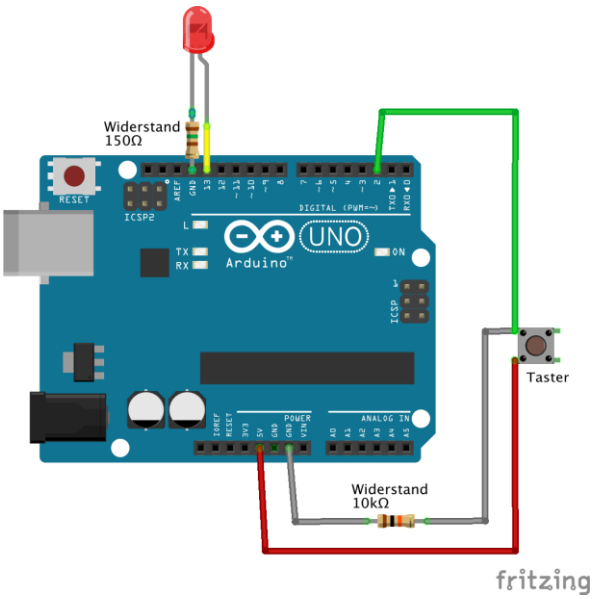
Wenn-Abfragen (if-Abfrage) sind immer Ja/Nein-Fragen!

Stimmt es? Ist es wahr?
Wenn ja, dann mache…
Wenn nein, dann mache ansonsten…

Textprogrammierung
(für Fortgeschrittene)

```
int led_L = 13;
int taster_T1 = 2;

void setup()
{
   pinMode(led_L, OUTPUT);
   pinMode(taster_T1, INPUT);
}

void loop()
{
   if ( digitalRead(taster_T1) == 1) {
      digitalWrite(led_L, HIGH);
   } else {
      digitalWrite(led_L, LOW);
   }
}
```

3.4 Taster (erweitert)



Aufgabe: Wenn der Taster gedrückt wird, geht die LED entweder an oder aus (je nachdem, welchen Zustand die LED zuvor hatte).

Bauteile:
1x LED (an Pin 13)
1x Widerstand, 150 Ω
1x Taster (an Pin 2)
1x Widerstand, 10 kΩ

- AUFBAU AUF DEM BREADBOARD

  Der Breadboard-Aufbau bleibt identisch mit der vorherigen Aufgabe.

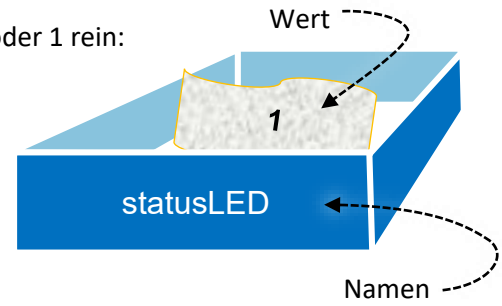
- PROGRAMMIERUNG

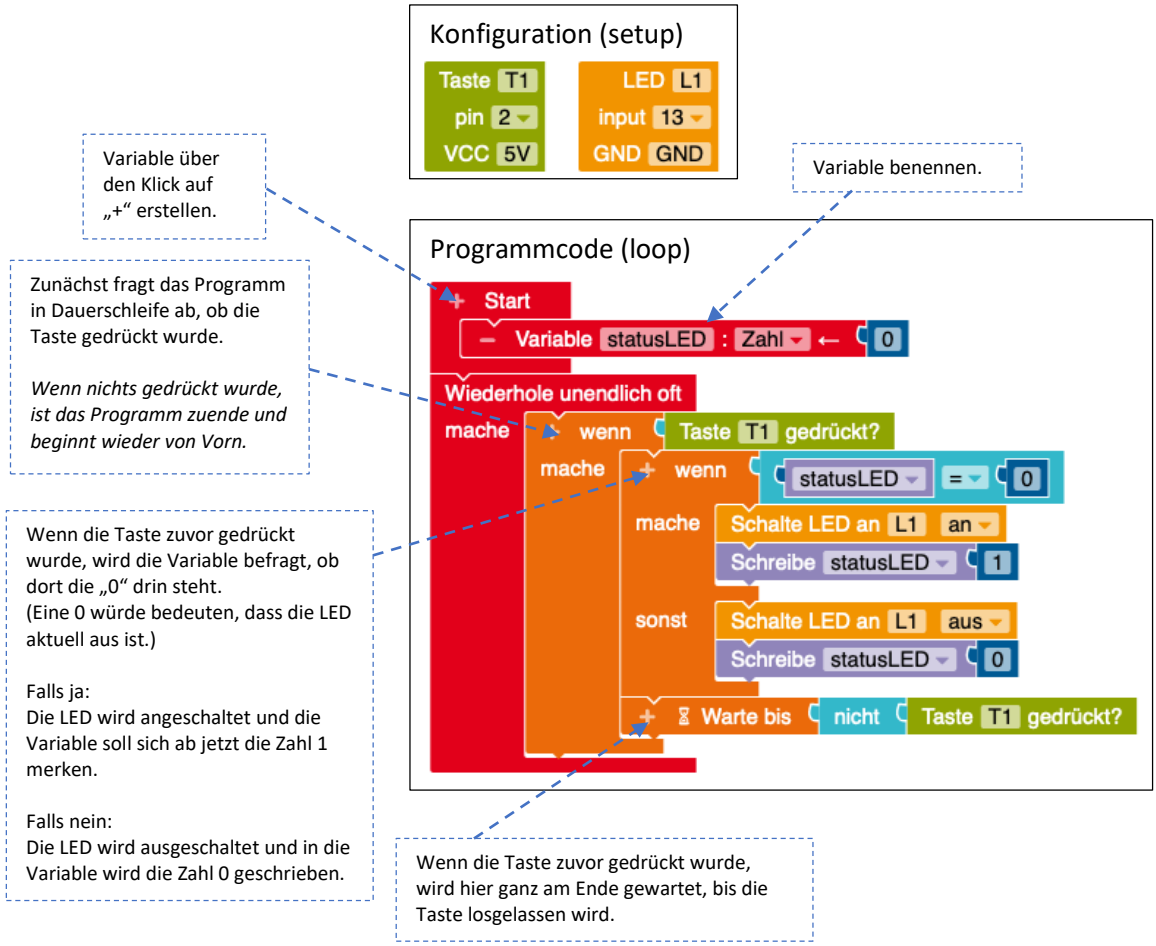
  Für die Programmierung muss sich der Arduino den Zustand der LED merken:
  „Habe ich gerade die LED angeschaltet oder ausgeschaltet?“

  Damit ein Programm sich etwas merken kann, werden **Variablen** benötigt. Eine Variable kann man sich wie ein Behälter (Speicher) vorstellen, in dem immer nur ein Zettel (Wert) reinpasst.

  Jeder Behälter (Varibale) braucht einen Namen, damit man diesen aufrufen kann.
  In jedem Behälter (Variable) liegt ein Zettel mit einem Wert.

  Wir schreiben in unsere Variable immer nur entweder 0 oder 1 rein:
  Die LED wurde angemacht (1) oder ausgemacht (0).



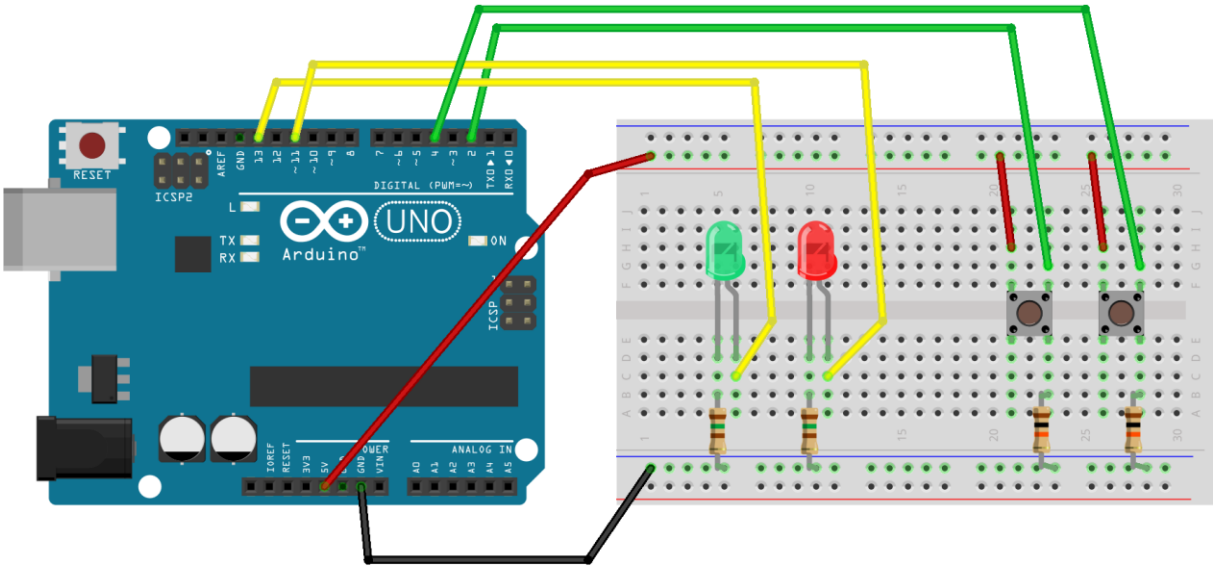


Textprogrammierung

(für Fortgeschrittene)

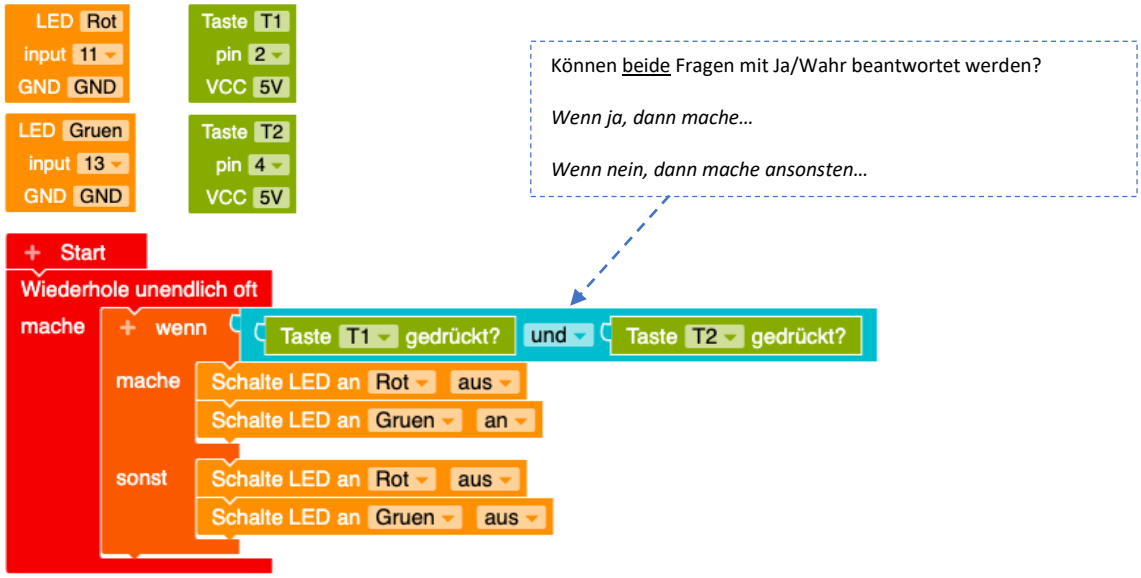
```
int statusLED;
int led_L1 = 13;
int taster_T1 = 2;

void setup()
{
   pinMode(led_L1, OUTPUT);
   pinMode(taster_T1, INPUT);
   statusLED = 0;
}

void loop()
{
   if ( digitalRead(taster_T1) == 1 ) {
      if ( statusLED == 0 ) {
         digitalWrite(led_L1, HIGH);
         statusLED = 1;
      } else {
         digitalWrite(led_L1, LOW);
         statusLED = 0;
      }
            while ( digitalRead(_taster_T1) ) {
         delay(1);
      }
   }
}
```

3.5 Und-Schaltung



Aufgabe: Die rote LED leuchtet, wenn kein Taster gedrückt wurde.
Wenn beide Taster gedrückt werden, leuchtet die grüne LED.

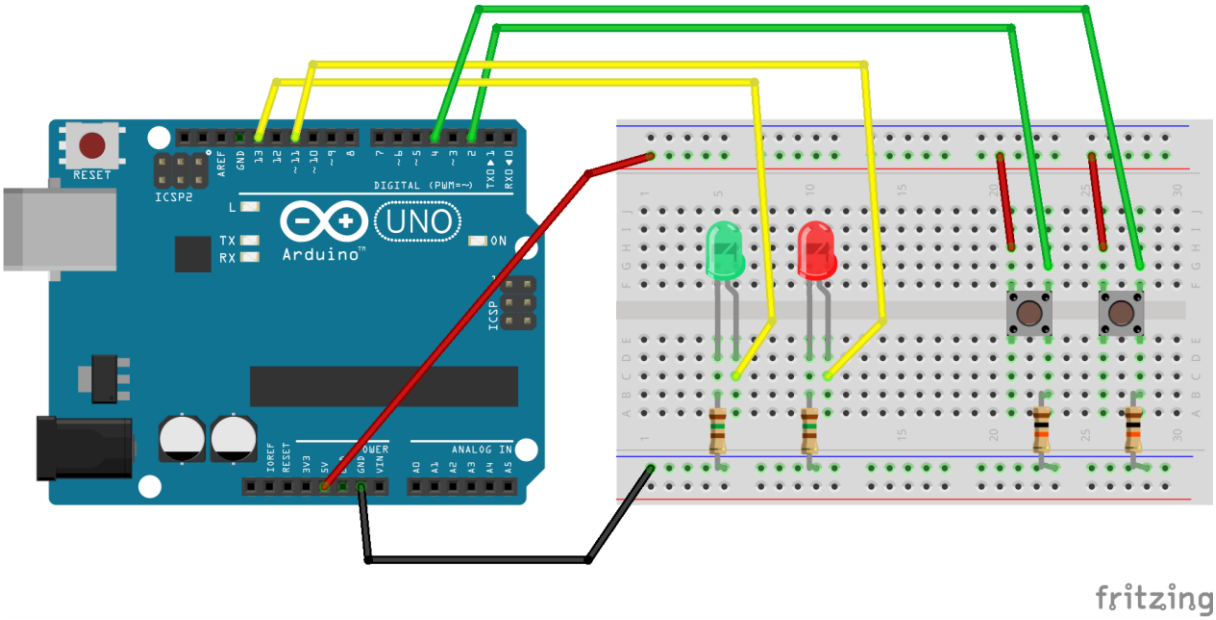
Bauteile: 2x LEDs (an Pin 10 & 12)
2x Widerstände, 150 Ω
2x Taster (an Pin 2 & 4)
2x Widerstände, 10 kΩ



Textprogrammierung
(für Fortgeschrittene)

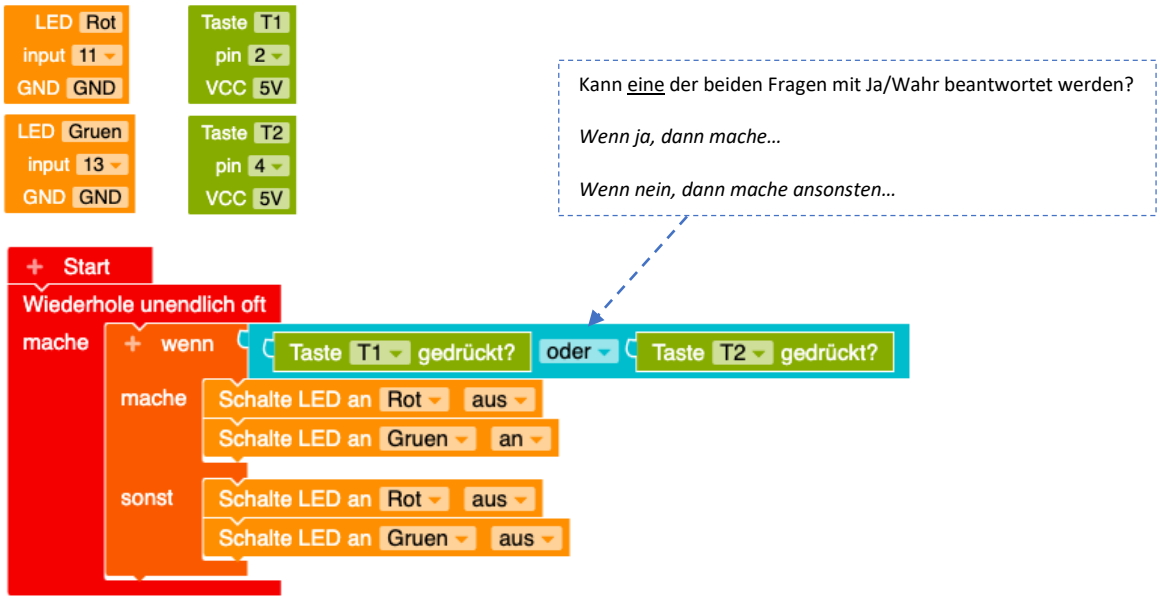
```
//Eine UND-Verknüpfung wird mit „&&“ programmiert.
  if ( digitalRead(T1) == 1 && digitalRead(T2) == 1) {
  }
```

3.6 Oder-Schaltung



Aufgabe: Die rote LED leuchtet, wenn kein Taster gedrückt wurde.
Wenn eine der beiden Tasten gedrückt wird, leuchtet die grüne LED.

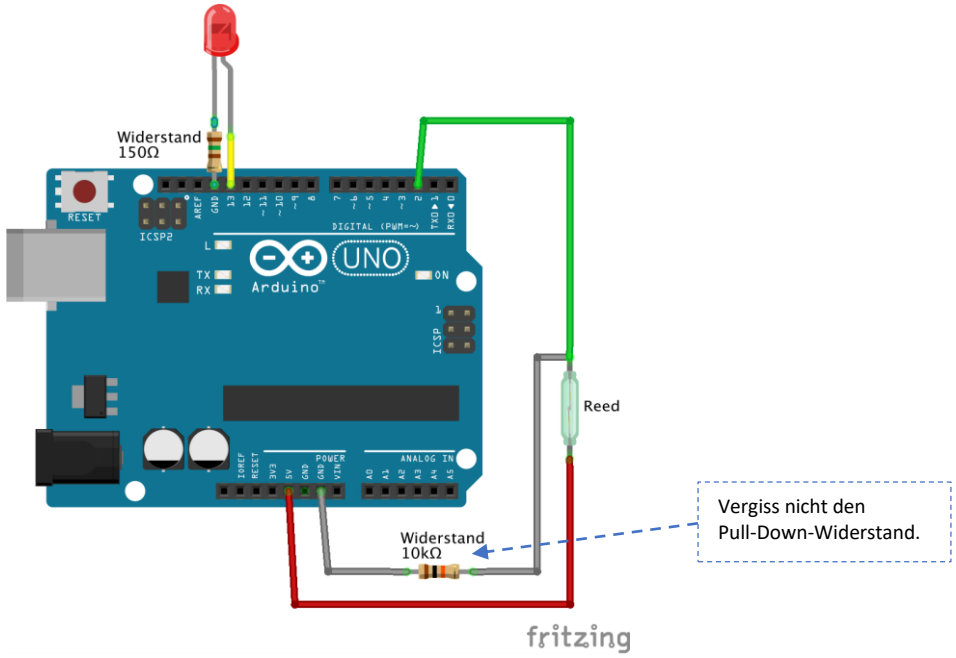
Bauteile: 2x LEDs (an Pin 10 & 12)
2x Widerstände, 150 Ω
2x Taster (an Pin 2 & 4)
2x Widerstände, 10 kΩ



Textprogrammierung
(für Fortgeschrittene)

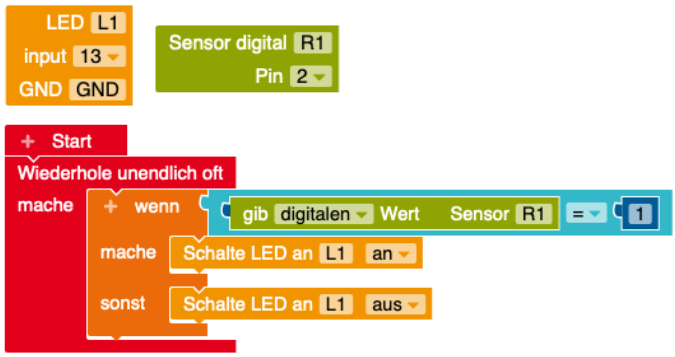
```
//Eine UND-Verknüpfung wird mit „||“ programmiert.

  if ( digitalRead(T1) == 1 || digitalRead(T2) == 1) {

  }
```

3.7 Reed-Kontakt



Aufgabe: Wird der Reed-Kontakt durch einen Magneten geschlossen, leuchtet die LED.

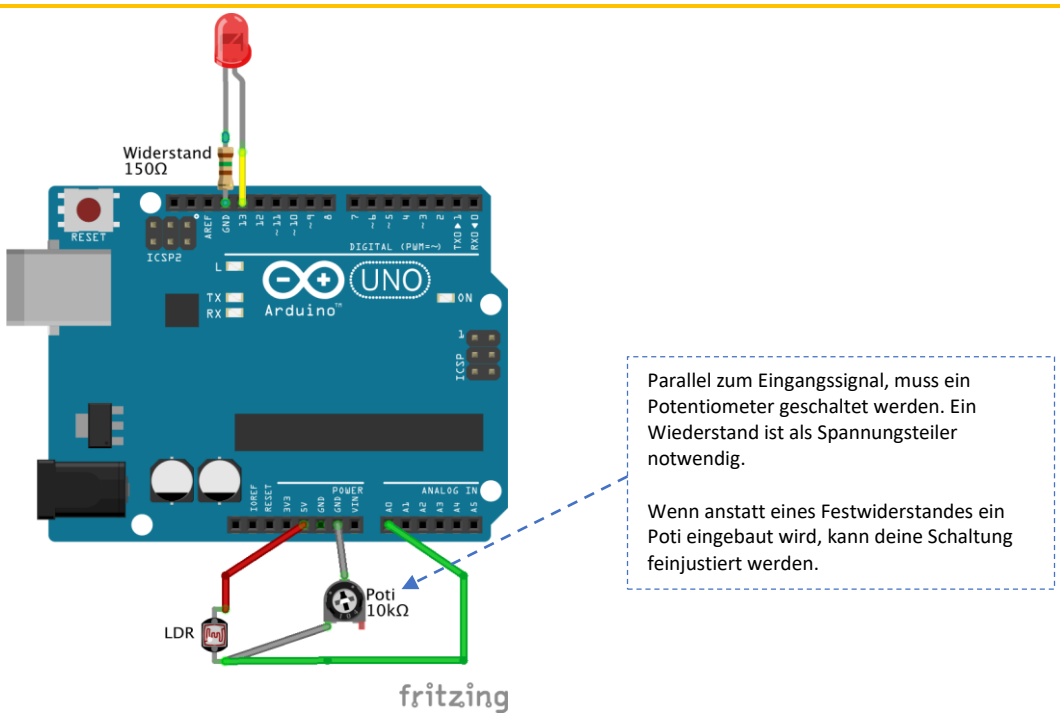
Bauteile: 1x LEDs (an Pin 13)
1x Widerstand, 150 Ω
1x Reed-Kontaktschalter (an Pin 2)
1x Widerstand, 10 kΩ



Textprogrammierung
(für Fortgeschrittene)

```
int led_L1 = 13;
int input_R1 = 2;

void setup()
{
   pinMode(led_L1, OUTPUT);
   pinMode(input_R1, INPUT);
}

void loop()
{
   if ( digitalRead(input_R1) == 1 ) {
      digitalWrite(led_L1, HIGH);
   } else {
      digitalWrite(led_L1, LOW);
   }
}
```

3.8 Lichtwiderstand (LDR)

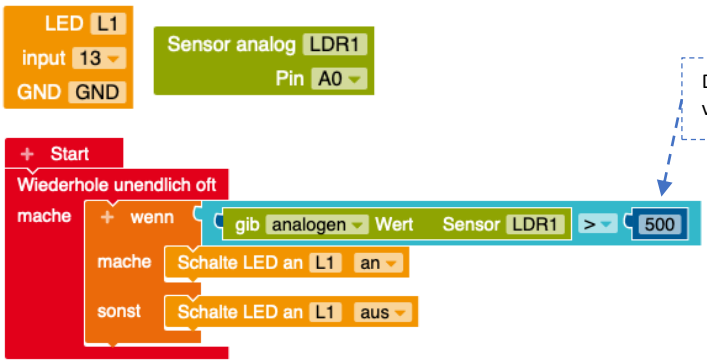


Parallel zum Eingangssignal, muss ein Potentiometer geschaltet werden. Ein Wiederstand ist als Spannungsteiler notwendig.

Wenn anstatt eines Festwiderstandes ein Poti eingebaut wird, kann deine Schaltung feinjustiert werden.

Aufgabe: Wird es dunkel, leuchtet die LED.
Wird es hell, geht die LED aus.

Bauteile: 1x LEDs (an Pin 13)
1x Widerstand, 150 Ω
1x LDR-Widerstand (an A0)
1x Potentiometer, 10 kΩ

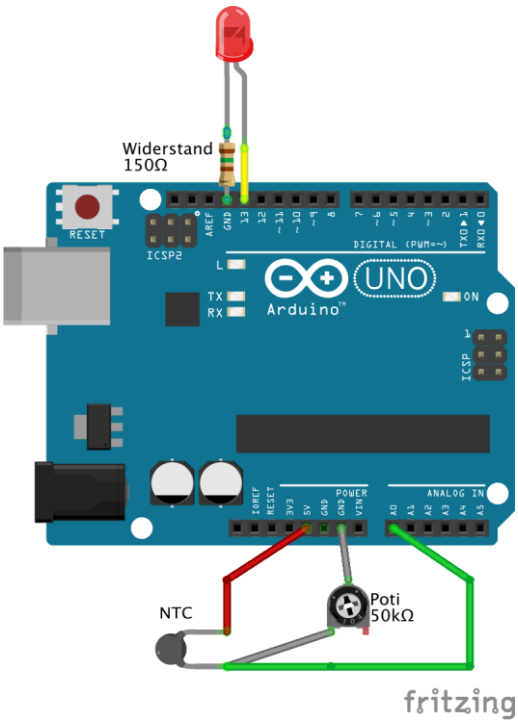


Der Wert 500 kann je nach Wunsch verändert werden.

Textprogrammierung
(für Fortgeschrittene)

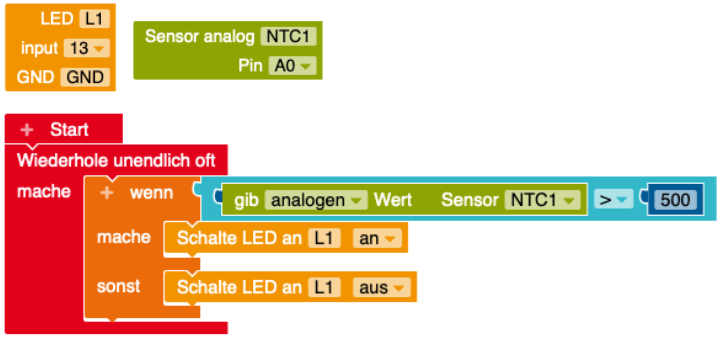
```
int input_LDR1 = A0;
int led_L1 = 13;

void setup()
{
   pinMode(input_LDR1, INPUT);
   pinMode(led_L1, OUTPUT);
}

void loop()
{
   if ( analogRead(input_LDR1) > 500 ) {
      digitalWrite(led_L1, HIGH);
   } else {
      digitalWrite(led_L1, LOW);
   }
}
```

3.9 Temperaturwiderstand (NTC)



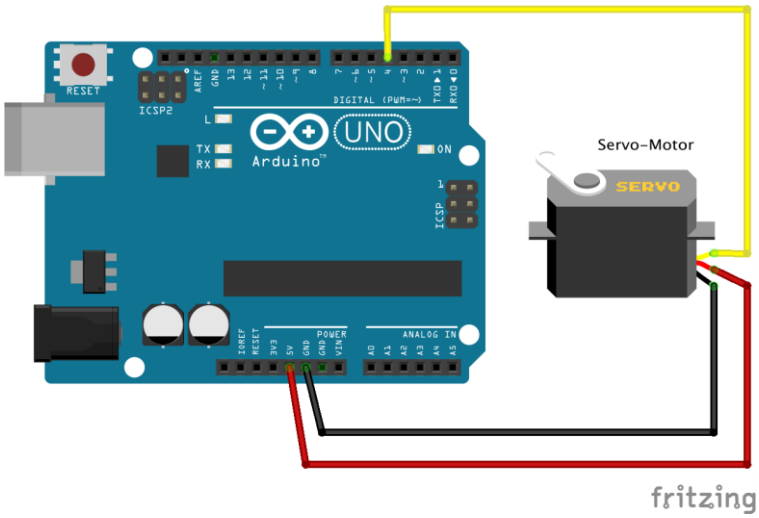
Aufgabe: Wird es kalt, leuchtet die LED.
Wird es warm, geht die LED aus.

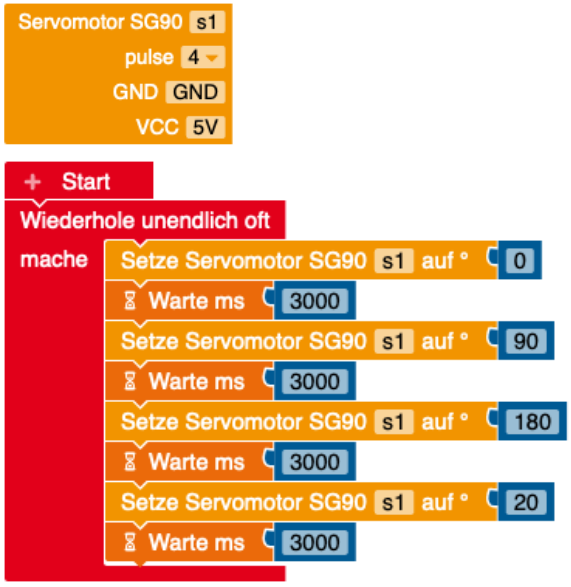
Bauteile: 1x LEDs (an Pin 13)
1x Widerstand, 150 Ω
1x NTC-Widerstand (an A0)
1x Potentiometer, 10 kΩ




```
int input_NTC1 = A0;
int led_L1 = 13;

void setup()
{
   pinMode(input_NTC1, INPUT);
   pinMode(led_L1, OUTPUT);
}

void loop()
{
   if ( analogRead(input_NTC1) > 500 ) {
      digitalWrite(led_L1, HIGH);
   } else {
      digitalWrite(led_L1, LOW);
   }
}
```

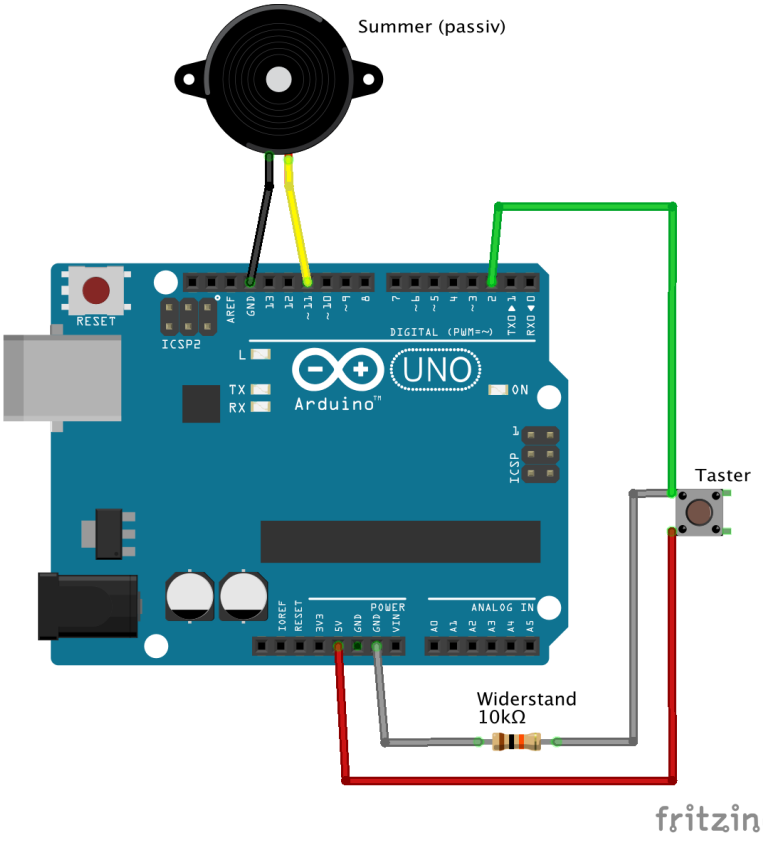
3.10 Servo-Motor






```
#include <Servo.h>

Servo servo_s1;

void setup()
{
   servo_s1.attach(4);
}

void loop()
{
   servo_s1.write(0);
   delay(3000);
   servo_s1.write(90);
   delay(3000);
   servo_s1.write(180);
   delay(3000);
   servo_s1.write(20);
   delay(3000);
}
```

3.11 Summer (passiv)




```
int taster_T = 2;
int summer_s1 = 11;

void setup()

{
   pinMode(taster_T, INPUT);
}

void loop()
{
   if ( digitalRead(taster_T) ) {
      tone(summer_s1,300);
      delay(1000);
   } else {
      noTone(summer_s1);
   }
}
```