Arduino Handbuch für Einsteiger

Michael Bonacina

2. Auflage: April 2018

ISBN 978-3-96645-011-9

Herausgegeben durch:
BMU Media GmbH
Hornissenweg 4
84034 Landshut

Arduino Handbuch für Einsteiger

Inhaltsverzeichnis

Alle Programmcodes aus diesem Buch sind als PDF zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:

https://bmu-verlag.de/arduino_handbuch/

1. Hinführung

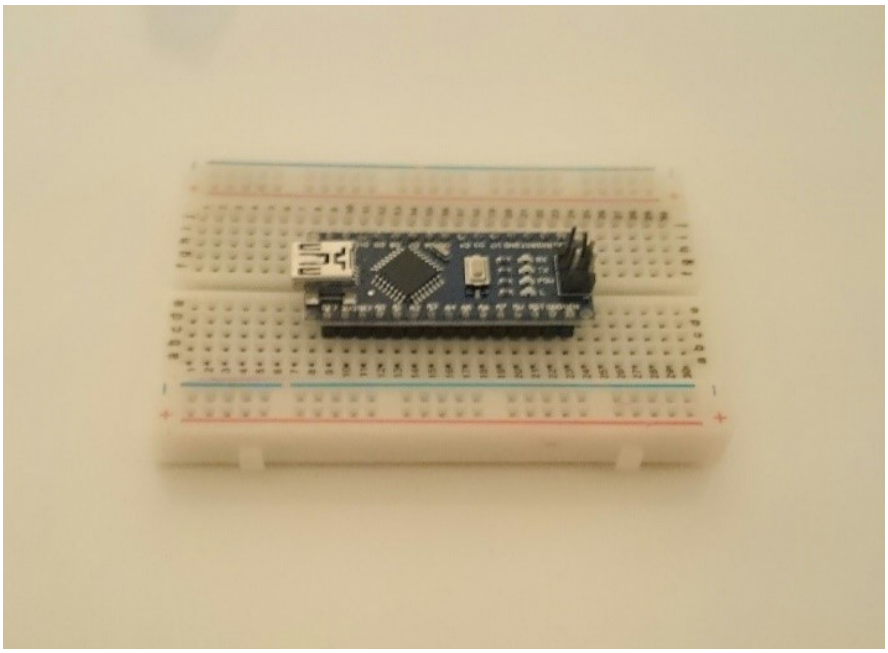
1.1 Einleitung

Lange Zeit war das Basteln mit Mikroelektronik ein teures und vor allem sehr kompliziertes Hobby, weshalb es von vielen Tüftlern links liegen gelassen wurde. Durch die Einführung der Arduino Plattform im Jahr 2005 ändert sich das jedoch gewaltig, und es hat sich bereits eine große, stark wachsende, Community rund um Arduino entwickelt. Die Arduino Plattform besteht einerseits aus einer großen Auswahl an Arduino Boards, und andererseits aus einer Programmierentwicklungsumgebung (IDE), wodurch die Einstiegshürde extrem gesenkt wurde und das Tüfteln mit Robotern und Mikroelektronik jedem, auch ohne Studium oder umfangreiche Vorkenntnisse, ermöglicht wird. Durch dieses Buch wirst du die nötigen Grundkenntnisse lernen, um einen frustfreien Start mit deinem Arduino zu haben, und ich werde dir am Ende einige Projekte zum Nachbauen vorstellen. Idealerweise solltest du das Buch von Anfang an lesen, da es aufeinander aufbaut, jedoch kannst du bei entsprechendem Vorwissen auch direkt zu den für dich interessanten Kapiteln springen. Alle Programmcodes die in diesem Buch vorgestellt werden, sind auch in digitaler Form auf Github verfügbar: https://gist.github.com/michaels123 .

1.2 Die verschiedenen Arduino Boards

Es gibt mittlerweile eine zweistellige Anzahl an Arduino Boards zur Auswahl, wodurch sich für jeden Einsatzzweck der richtige Arduino finden lässt. Am unteren Ende der Preis-, Leistungs- und Größenskala befinden sich der Arduino Micro und der Arduino

Nano. Wie bereits der Name erahnen lässt, haben diese beiden Boards einen sehr kleinen Formfaktor, und lassen sich sogar direkt in ein Breadboard stecken, wozu das gut ist aber an späterer Stelle. Wie alle Arduinos setzen auch diese auf einen integrierten Mikroprozessor von Atmel, wobei im Micro ein ATmega32U4 steckt und im Nano ein ATmega328P. Diese beiden Prozessoren sind jedoch bereits relativ leistungsfähig und besitzen beide einen 16Mhz Takt sowie 32KB (üblicher Desktop-PC: 1TB = 1000000000KB) Programmspeicherplatz.

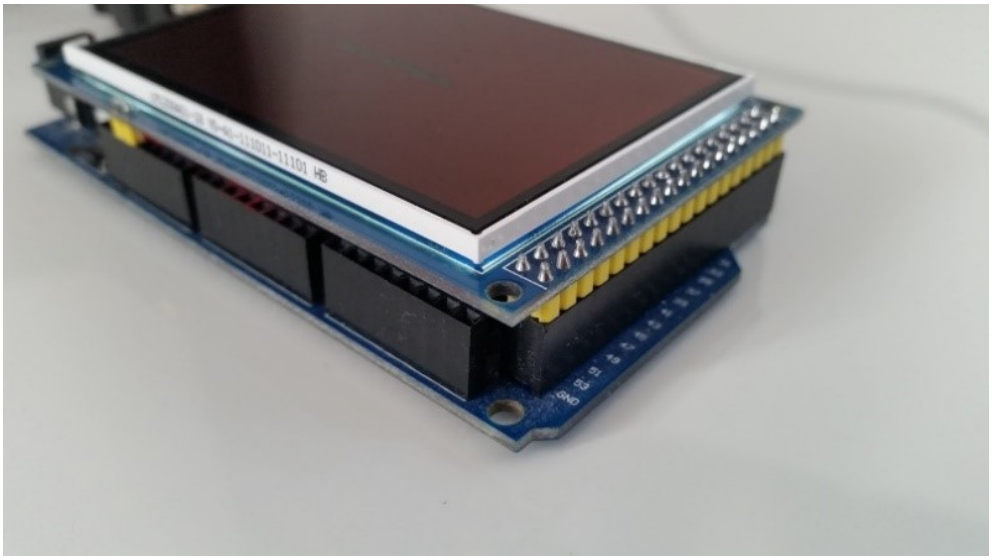


Nano V3 mit Breadboard kompatibler Bauform

Eine Stufe größer sind der Arduino Uno und der Arduino Leonardo, welche prinzipiell dieselben technischen Daten aufweisen, wie der Arduino Nano und der Arduino Micro, jedoch in einem anderen Formfaktor. Während Nano und Micro direkt in ein Breadboard gesteckt werden können, haben der Uno und der Leonardo den Vorteil, dass sie Shield kompatibel sind, also eine spezielle Anordnung ihrer Pins haben.

Arduino Uno

Shields sind eine besondere Erfindung der Arduino Welt: Es gibt verschiedene Shields, wie beispielsweise Display Shields oder WiFi Shields, auf denen die gesamte Hardware bereits fertig verlötet und Arduino kompatibel zusammengebaut ist. Dadurch muss man das Shield nur noch auf den Arduino in Shield Bauform „aufsetzen“ und kann sich auf das reine Programmieren, ohne irgendwelche Lötarbeiten, konzentrieren.



TFT Displayshield am Arduino Mega2560

Für größere Projekte gibt es den Arduino Mega 2560, welcher durch den Atmel Mega2560 Prozessor sehr gute technische Daten hat: 256KB Speicherplatz für Programme, sowie 8KB SRAM und insgesamt 70 I/O Pins, welche man programmieren kann zum Steuern von LEDs, Motoren oder Sonstigem. Nochmal eine Stufe schneller ist dann nur noch der Arduino Due, der ungefähr die selbe Pinanzahl hat, jedoch dank eines besseren Prozessors einen 84Mhz Takt (gegenüber 16Mhz bei UNO oder MEGA2560) und ganze 512KB Programmspeicherplatz (gegenüber 32KB beim UNO).

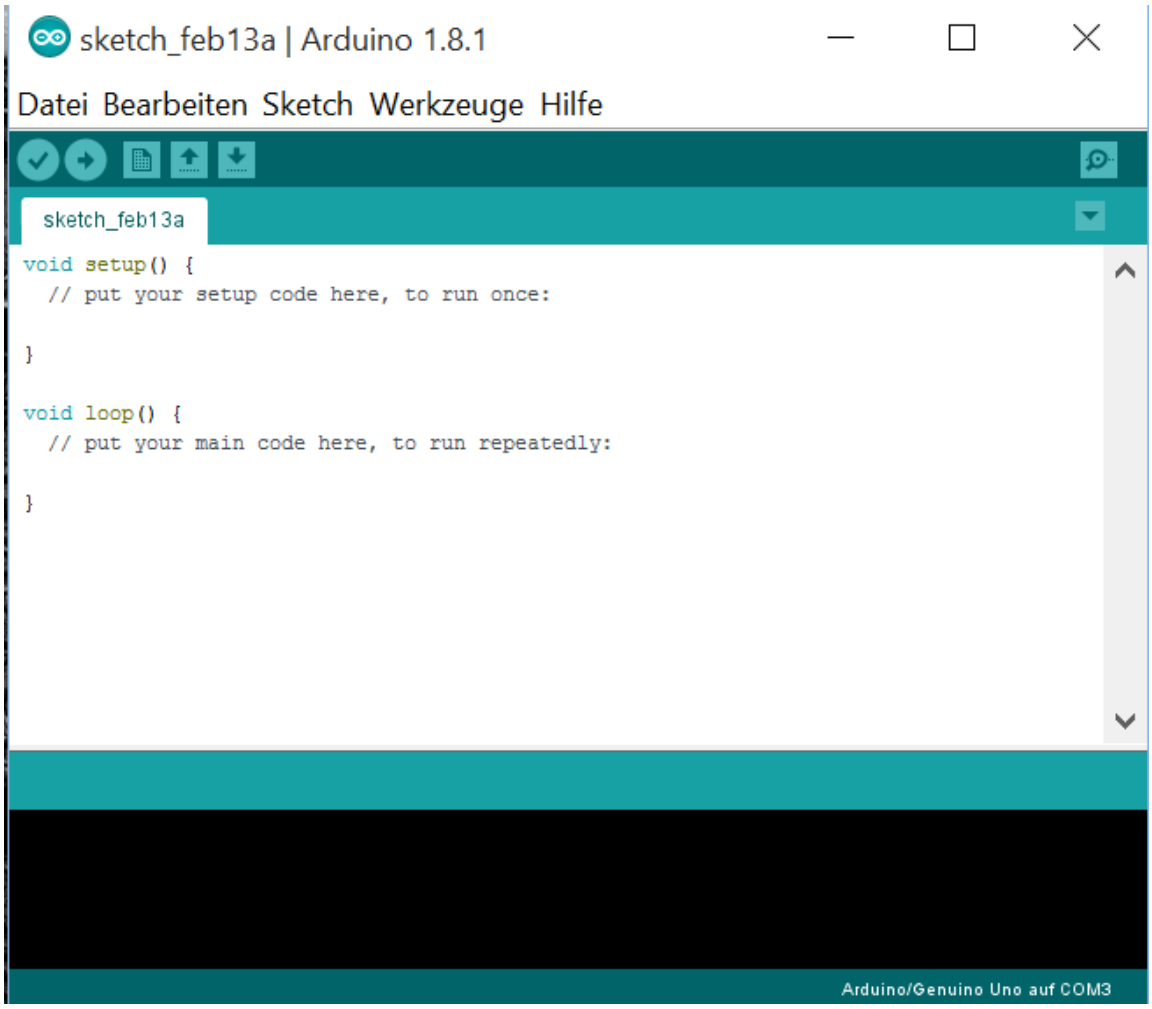
Zusätzlich gibt es mittlerweile eine Reihe Arduinos, welche bereits internetfähig sind, wie beispielsweise der Arduino Yun. Dieser besteht einerseits aus einem „normalen“ Arduino Leonardo, hat jedoch zusätzlich noch einen zweiten Prozessor mit WiFi Chip und Linux, für die Verwendung des Internets.

Erwähnenswert ist noch, dass es neben diesen Original Arduinos auch noch viele Arduino Klone gibt. Diese sind meist 1 zu 1 gleich aufgebaut (möglich dank der Open Source Architektur), sind jedoch nicht von Arduino selbst, sondern von namenlosen Fabriken in China, weshalb sie auch nicht den Namen Arduino tragen. Meistens funktionieren diese Klone sehr gut, und das zu einem Drittel des Preises, jedoch sind Defekte hier nicht ausgeschlossen, weshalb ich zum Start einen original Arduino (UNO) empfehlen würde, und du dir dann bei entsprechenden Kenntnissen auch Arduino Klone nachbestellen kannst. Auch in den folgenden Beispielen wird, soweit möglich, ein Arduino UNO Version 3 verwendet.

UNO R3 China Klon links – Original Arduino Uno Version 3 rechts

1.3 Installation und Setup der Arduino DIE

Sobald du dir deinen Arduino bestellt hast, geht es an die Programmierung, wobei für den Start die original Arduino IDE am besten geeignet ist. Diese kannst du dir von der offiziellen Arduino Website unter https://www.arduino.cc/en/Main/Software einfach herunterladen. Nach erfolgreichem Download wird man durch den Installations-Wizard durch die Installation geführt. Nun kann man die Arduino IDE öffnen und sein erstes Programm schreiben.



Arduino IDE nach dem ersten Öffnen

In der Mitte der Arduino IDE befindet sich ein Texteditorfeld, in welches man das eigentliche Programm, Fachsprache Sketch, schreiben kann. Links oben kann man unter Datei die üblichen Operationen durchführen, wie speichern, öffnen oder eine neue Datei erstellen. Außerdem finden sich unter Datei > Beispiele eine Menge fertiger Beispielprogramme, wodurch man bereits erste Ergebnisse ohne Programmieraufwand bekommen kann. Bevor man jedoch einen Sketch auf den Arduino hochladen kann, muss man zuvor unter Werkzeuge > Board den passenden Arduino und unter Werkzeuge > Port den aktiven Port auswählen. Um nun einen Sketch auf den Arduino hochzuladen kann man sich der Symbole in der zweiten Zeile bedienen: Der Haken speichert und

kompiliert den aktuellen Sketch, übersetzt also deinen C++ Code in „Maschinensprache“ und überprüft ihn gleichzeitig auf Syntaxfehler, also beispielsweise falsche Schreibweisen oder fehlende Klammern. Der Knopf eins weiter rechts tut dasselbe, lädt den Sketch jedoch zusätzlich auch noch auf den Arduino hoch.

1.4 Aufbau der Arduino Sketche

Wie man im Editorbereich nach dem Öffnen der Arduino IDE bereits erkennen kann, besteht jeder Arduino Sketch prinzipiell aus zwei Teilen: dem setup Teil und dem loop Teil. Dabei wird der Setup Teil nur einmal am Anfang ausgeführt, der loop Teil ständig von oben bis unten durchlaufen. Im setup Teil werden daher einmalige Festlegungen getroffen, wie die Definition von Variablen (Speicherplatzhaltern in die du Werte speichern kannst) oder welche Pins als Output verwendet werden, und welche als Input. Im loop Teil befindet sich das eigentliche Programm, welches beispielsweise die LED anmacht, eine Sekunde wartet, die LED ausmacht, und wieder eine Sekunde wartet. Durch die ständige Wiederholung dieses Abschnittes entsteht hieraus eine blinkende LED. Zusätzlich zu diesen zwei Bereichen gibt es zwei Ausnahmen, die wir später genauer betrachten werden: Externe Bibliotheken werden noch vor dem setup Teil im Programm nachgeladen und Funktionsdefinitionen werden nach dem loop Teil aufgeschrieben.

2. Grundlagen

2.1 Hello World – Blinken einer LED

In den meisten Programmiersprachen lernt man zunächst als Einführung das Ausgeben von „Hello World“ am Bildschirm, das Pendant hierfür am Arduino ist das Blinken einer LED. Praktischerweise haben alle Arduinos bereits eine LED fertig auf dem Board verbaut, meist an Pin 13, weshalb wir uns zunächst einmal auf den Sketch fokussieren können. Im Sketch selbst müssen wir dem Arduino zunächst einmal mitteilen, dass wir Pin 13 des Arduino als Output verwenden möchten, wir hier also Signale erzeugen und nicht empfangen möchten. Die Pindefinition erfolgt im setup Teil über den Befehl:

```
pinMode(Pinnummer, Pinart);
```

Dabei gibt man als erstes Argument (=Werte die man einer Methode, in diesem Fall pinMode, übergibt) die Pinnummer, welche auch immer direkt am Arduino steht, an und als zweites Argument OUTPUT oder INPUT, je nachdem welche Funktion der Pin haben soll. In unserem Fall muss die Zeile im setup Teil also:

pinMode(13, OUTPUT);

lauten.

Arduino Uno Version 3

Die obere Leiste sind die 14 digitalen I/O (Input/Output) Pins des Arduino mit den Nummern 0 bis 13, und unten rechts sind die sechs analogen Eingänge A0 bis A5. Links unten ist mit dem Ground (Minuspol) und +5V und +3.3V die Spannungsversorgung für Sensoren zu finden.

Wie in den meisten Programmiersprachen muss man auch hier jeden Befehl mit einem Semikolon (;) beenden, damit der „Übersetzer“ in Maschinencode weiß, dass dieser Befehl vorbei ist, und jetzt der Nächste kommt. Nun muss man die LED noch blinken lassen. Dazu muss man sie im loop Teil zunächst einmal anschalten, also auf den Output Pin Spannung anlegen. In der Praxis funktioniert das über den Befehl:

```
digitalWrite(Pinnummer, Status);
```

wobei Pinnummer wieder die Angabe darüber ist, welcher Pin geschaltet werden muss und Status LOW oder HIGH, je nachdem ob Spannung anliegen soll oder nicht. Dabei bedeutet HIGH laut Datenblatt +5V und LOW natürlich 0V. In der Praxis sind es aber aufgrund mehrerer Faktoren nicht ganz +5V, sondern ehr +4.8V. In unserem Beispiel zum Anschalten der LED an Pin 13 müsste der Befehl also so aussehen:

```
digitalWrite(13, HIGH);
```

Bevor man nun die LED wieder ausschalten kann, muss man eine kurze Verzögerung einbauen, da sonst kein menschlich wahrnehmbares Blinken entstehen würde. Dazu kann man sich des Befehls

```
delay(Anzahl Millisekunden);
```

bedienen. Für eine Sekunde Wartezeit muss der Befehl also

```
delay(1000);
```

lauten. Jetzt kann man im loop Teil die LED wieder ausschalten und wieder eine Sekunde Wartezeit einbauen.

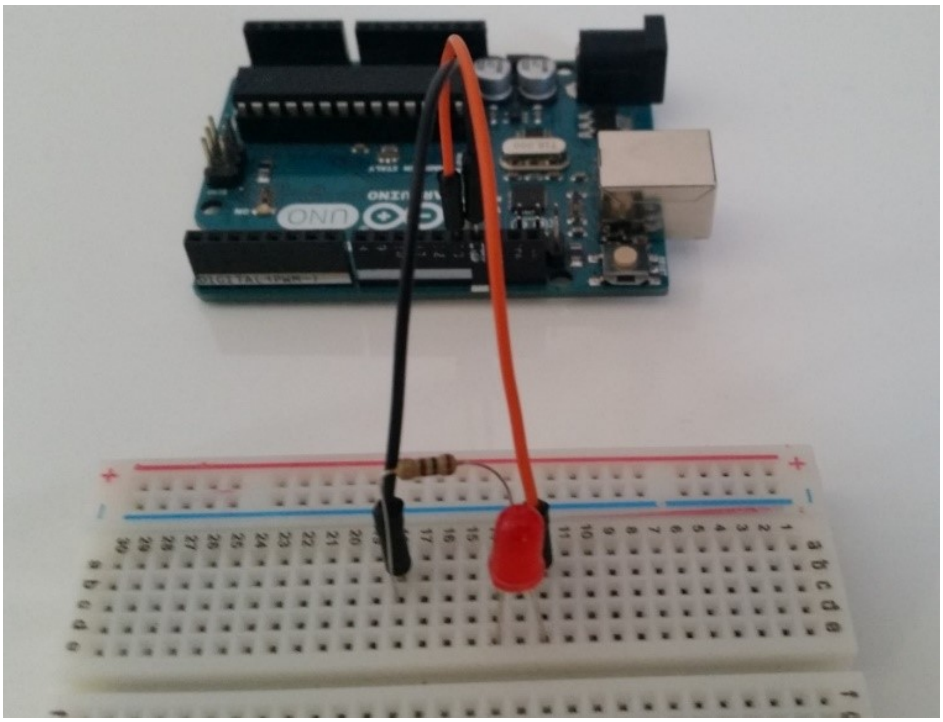
```
void setup() {
  pinMode(13, OUTPUT);            //Pin 13 als Output festlegen

}

void loop() {
  digitalWrite(13, HIGH);         //Spannung an Pin 13 anlegen
```

```
  delay(1000);                    //Eine Sekunde warten
  digitalWrite(13, LOW);          //Keine Spannung an Pin 13
  delay(1000);                    //Eine Sekunde warten
}
```

Durch die ständige Wiederholung des loop Teils ergibt sich nun eine blinkende LED am Arduino. Dazu musst du den Sketch einfach über den zweiten Button von links in der zweiten Zeile (Hochladen) auf den Arduino hochladen.

2.2 Widerstände

Durch einen Widerstand im Stromkreis lässt sich der Stromfluss begrenzen. Das ist oft notwendig, da viele Geräte einen sehr geringen Innenwiderstand haben (= der Widerstand des Geräts selbst) und dadurch der Stromfluss durch den Stromkreis zu groß werden könnte, und das Gerät durchbrennen würde. Um den Stromfluss also durch einen Widerstand zu begrenzen, muss eine Unterbrechung im Stromkreis gemacht werden, die man über den Widerstand schließt (= in Reihe schalten). Dabei spielt es prinzipiell keine Rolle ob der Widerstand „vor“ dem Gerät oder „nach“ dem Gerät in den Stromkreis eingebaut wird.



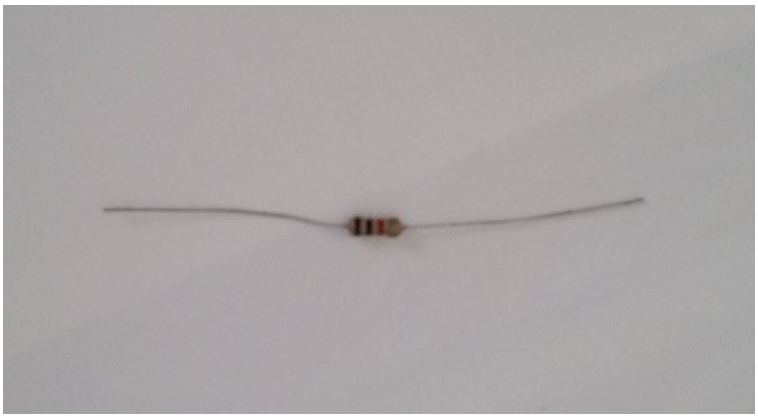
Widerstand im Stromkreis

Es gibt sehr viele verschiedene Arten von Widerständen, jedoch verwendet man im Arduino Bereich hauptsächlich Kohleschichtwiderstände oder Metallschichtwiderstände. Kohleschichtwiderstände sind meist gelb, Metallschichtwiderstände meist blau. Metallschichtwiderstände sind dabei tendenziell etwas genauer, dafür aber auch etwas teurer. In fast allen Arduino Projekten sind jedoch einfache Kohleschichtwiderstände genug.

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		± 0.05%	
White	9	9	9			
Gold				0.1	± 5%	(J)
Silver				0.01	± 10%	(K)

Erklärung der Farben bei Widerständen

Um die Stärke eines Widerstandes abzulesen muss man sich die vier oder fünf farbigen Striche genauer ansehen. Ein Strich ist immer etwas dicker, oder etwas weiter entfernt. Dieser gibt die Toleranz des Widerstands an. Von der gegenüberliegenden Seite des Toleranzstrichs beginnend muss man nun zwei (insgesamt vier Striche) oder drei (insgesamt fünf Striche) ablesen und gemäß der Tabelle in Zahlen übersetzen. Der Strich neben dem Toleranzstrich gibt den Faktor an mit dem die zusammengesetzte Zahl multipliziert werden muss. Der braun-schwarz-rot-goldene Widerstand hat also folgende Stärke: Braun bedeutet 1, schwarz bedeutet 0, also 10. Bei rot muss man die Zahl mit 100 Ohm multiplizieren, also 10 mal 100 Ohm bedeutet 1000 Ohm oder 1 kOhm. Dabei ist jedoch zu beachten, dass als Toleranz golden, also 5% angegeben wurde.



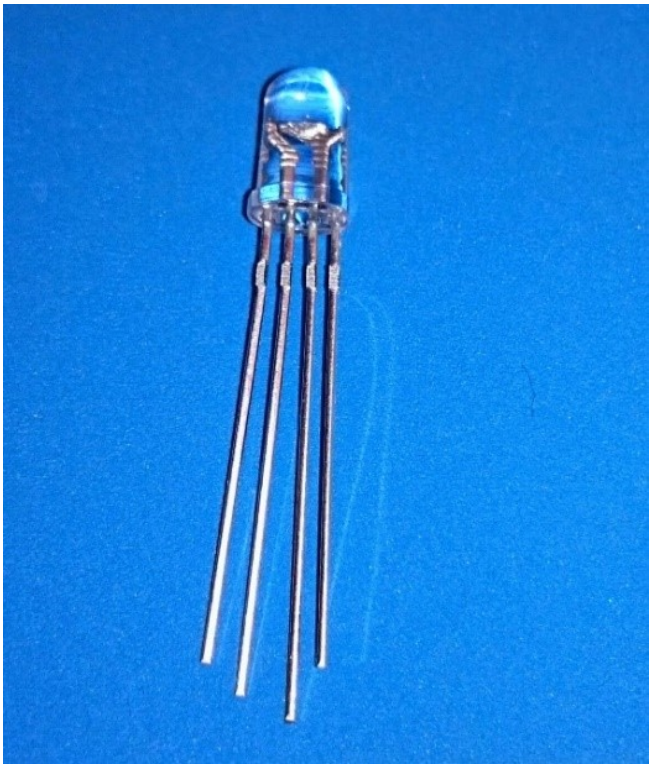
Widerstand: braun – schwarz- rot – golden

Eine andere Art der Verwendung von Widerständen sind Pull-up oder Pull-down Widerstände. Hier möchte man einen Kontakt HIGH (Pull-up) oder LOW (Pull-down) „ziehen“, ohne jedoch einen großen Stromfluss zu haben. Aus diesem Grund wird der Kontakt zwar mit dem Ground oder dem Pluspol verbunden, jedoch über einen Widerstand. Dadurch ist der Kontakt nur leicht HIGH oder LOW und kann durch ein aktiv gesetztes Signal trotzdem noch gesteuert werden. Ziel ist es, lediglich ein zufälliges hin und her Wechseln am Eingang zu verhindern, wenn kein Signal gesendet wird.

2.3 RGB LED

Eine RGB LED besteht eigentlich aus drei verschiedenen LEDs, nämlich aus einer Roten, einer Grünen und eine blauen LED, welche in nur einem „Gehäuse“ zusammengefasst wurden. Durch das unterschiedlich starke Leuchten der drei Farben, kann man dann neue Farben erzeugen. Eine typische RGB LED (common cathode) hat vier Pins: einen Ground, also einen Anschluss für den Minuspol und drei Pins für die Pluspole der einzelnen Farben.

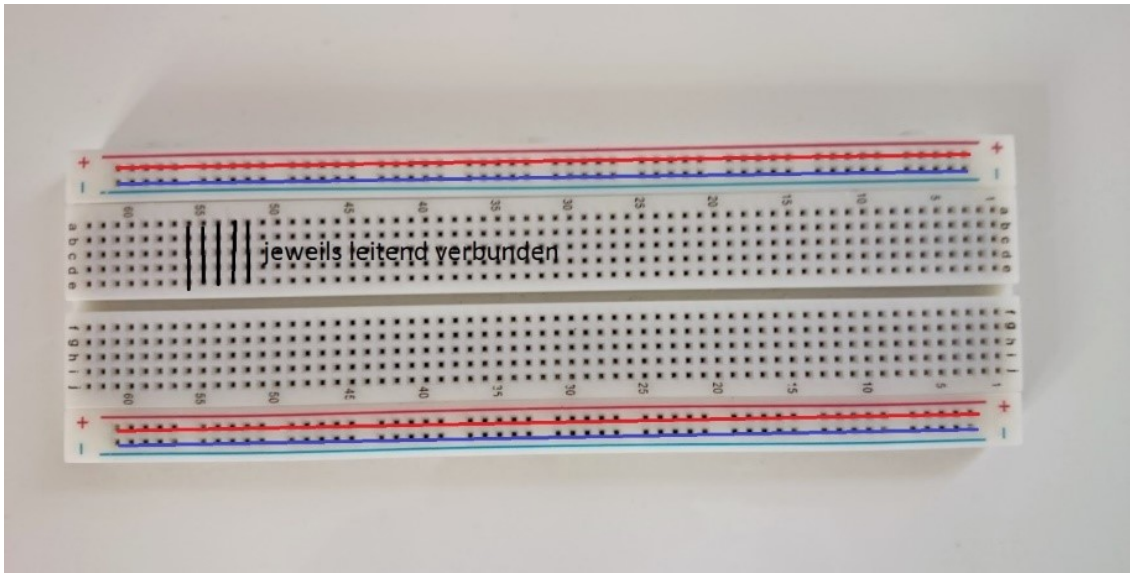
Durch diese Konstruktion spart man sich zwei Pins, da alle drei Einzelleds denselben Minuspol haben, jedoch erst leuchten können sobald zusätzlich auch noch Spannung an ihrem Pluspol anliegt. Dabei ist der längste Pin bei einer solchen LED der Minuspol (=Ground), welche mit dem Ground des Arduino verbunden werden muss. Welcher der anderen Pins zu welcher Farbe gehört kann von Modell zu Modell variieren, weshalb man hier jeden Pin mithilfe unseres Blink Sketches durchgehen kann, um die Farben herauszufinden. Für Kabel die zum Ground führen verwendet man meist dunkle Farben (schwarz oder blau), für Kabel die zu digitalen I/O Pins oder der Versorgungsspannung führen meist helle (orange, gelb, weiß).



Common Cathode RGB LED

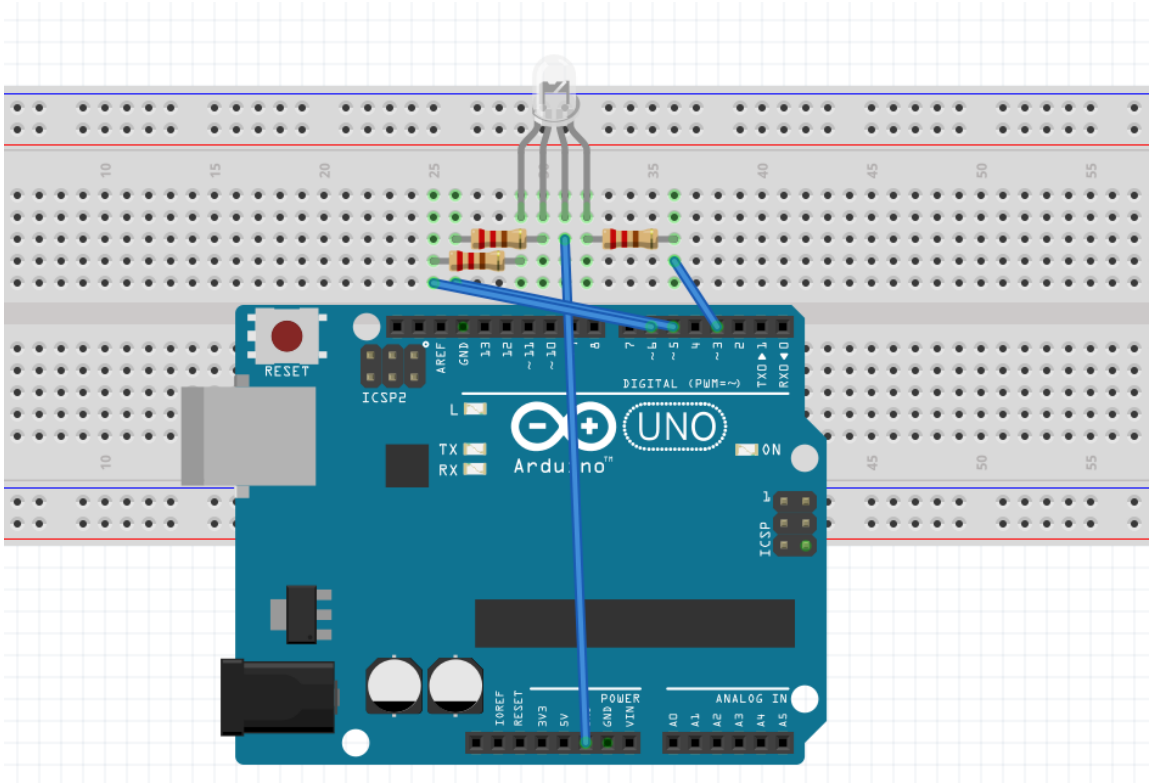
Damit man nicht bei jedem Projekt zum leitenden Verbinden aller Teile direkt alles miteinander verlöten muss, gibt es sogenannte

Breadboards. Diese bestehen aus einzelnen Steckplätzen die entlang der kurzen Seite leitend verbunden sind, jedoch mit einer Unterbrechung in der Mitte.



830 Kontakte Breadboard: Die Kontakte auf den einzelnen roten, blauen und schwarzen Linien sind jeweils leitend verbunden.

Um eine LED leitend mit dem Arduino zu verbinden, muss man in derselben Reihe auf dem Breadboard, in der der LED Kontakt ist, dass andere Ende des Kabels vom Arduino anschließen. Außerdem empfiehlt es sich vor die drei Pluspole jeweils einen Widerstand einzubauen (220 Ohm) um ein Durchbrennen eurer LED zu verhindern. Dazu muss man die Kabel versetzt zu den Kontakten im Breadboard anbringen, und jeweils über einen Widerstand verbinden.



RGB-LED mit dem Arduino Uno verbinden

Um mit einer RGB LED Farben zu erzeugen ist es nicht genug, wie im vorherigen Beispiel einfach nur die LEDs an oder aus zu schalten, da man dadurch nur sehr wenige Mischfarben erzeugen kann. Stattdessen muss man auch die Stärke des Leuchtens bestimmen können. Für diese Fälle gibt es PWM, Pulsweitenmodulation, welche analoge Signale an den PWM Pins des Arduino erzeugt. Diese Signale kann man von der Stärke sehr genau variieren, und dadurch auch einzelne Farben unserer RGB LED dimmen. Da PWM besondere Hardware voraussetzt, kann man nur manche Pins des Arduinos zur Erzeugung analoger Signale einsetzten: Pin 3, 5, 6, 9, 10, 11. Im Kapitel zu wichtigen Bibliotheken wird auch die SoftwarePWM Bibliothek vorgestellt dank der es auch möglich ist an beliebigen I/O Pins des Arduino ein PWM Signal zu erzeugen. Dabei kann die Stärke im Sketch über den Befehl

```
analogWrite(Pinnummer, Stärke);
```

festgelegt werden. Dabei bedeutet für die Stärke 0, dass kein Strom fließt und 255, dass die vollen 5V Spannung anliegen. Da die Erzeugung von PWM Signalen unabhängig von normalem LOW oder HIGH ist, könnte man hier sogar die Pindefinition mit pinMode() im setup Teil weglassen, jedoch würde ich sie trotzdem empfehlen, da man sonst übersehen könnte, dass die Pins bereits als OUTPUT verwendet werden. Ein kompletter Sketch, welcher eine RGB LED schaltet sähe dann ungefähr so aus:

```
void setup() {
  pinMode(3, OUTPUT);            //Pin 3 als Output festlegen
  pinMode(5, OUTPUT);            //Pin 5 als Output festlegen
  pinMode(6, OUTPUT);            //Pin 6 als Output festlegen

  analogWrite(3, 100);
  analogWrite(5, 200);
  analogWrite(6, 50);
}

void loop() {
    //keine zu wiederholende Aufgabe
}
```

Nun wollen wir einen Schritt weitergehen, und die Farbe im Sketch fließend verändern, also beispielsweise von Rot zu Blau langsam wechseln, und wieder zurück. Dazu bedient man sich eines Programmiertricks, nämlich der for-Schleife. Diese for Schleife besteht aus drei Teilen: Zunächst wird ein Variable erzeugt, welche üblicherweise bei Null beginnt (1.). Diese Variable wird nach jedem Durchlauf des Programmcodes in der for-Schleife um

eins hochgezählt (3.). Dabei wird jedoch vor jedem neuen Durchlauf überprüft, ob diese Variable noch unterhalb eines vorgegebenen Wertes liegt (2.), wenn ja, wird die for-Schleife ein weiteres Mal durchlaufen, sollte die Bedingung nicht mehr erfüllt sein, ist die for-Schleife beendet, und das Programm macht nach ihr weiter. Als Arduino Code sähe das dann so aus:

```
       1.        2.    3.
for(int i = 0;  i<10; i++)
{
       CODE
}
```

Bei dieser for-Schleife wird zunächst eine Variable namens i erzeugt, welche den Wert Null trägt. Durch Variablenname++ wird die Variable um eins hochgezählt, und durch i<10 wird überprüft, ob der Code schon zehnmal ausgeführt wurde. Das int vor dem i wird benötigt, um festzulegen, welcher Typ von Variable i sein soll. Dabei gibt es viele verschiedene Variablentypen. Hier die vier wichtigsten:

Name	Erklärung
int	Ganze Zahlen von -32768 bis +32767
float	Kommazahlen
char	Genau ein Zeichen
String	Zeichenkette

Die wichtigsten Variablentypen

Das i = 0, weist der neuen Variable vom Typ int lediglich den Wert 0 als Startwert am Anfang zu. Der Vorteil der for-Schleife ist, dass

man einen speziellen Code sehr oft wiederholen kann, und jedes Mal durch den veränderten Wert von i der Code im Ergebnis leicht abgeändert ist. Um in unserem Beispiel die LED von Rot nach Blau zu schalten wäre folgender Code, natürlich abhängig von den verwendeten Pins, notwendig:

```
for(int i = 0; i<255; i++)
{
        analogWrite(3, i);
        analogWrite(5, (255-i));
}
```

Hierdurch wird die Spannung an Pin 3 immer größer, während sie an Pin 5 immer kleiner wird. Der gesamte Code um eine RGB LED fließend von Rot nach Blau durchgehend wechseln zu lassen sieht dann so aus:

```
void setup() {
  pinMode(3, OUTPUT);             //Pin 3 als Output festlegen
  pinMode(5, OUTPUT);             //Pin 5 als Output festlegen
  pinMode(6, OUTPUT);             //Pin 6 als Output festlegen

}

void loop() {
  for(int i = 0; i < 255; i++)
  {
    analogWrite(3, i);
    analogWrite(5, (255-i));
    delay(10);
  }
  delay(500);
  analogWrite(3, 0);
  analogWrite(5, 255);
  delay(500);
```

```
}
```

2.4 Felder (Arrays)

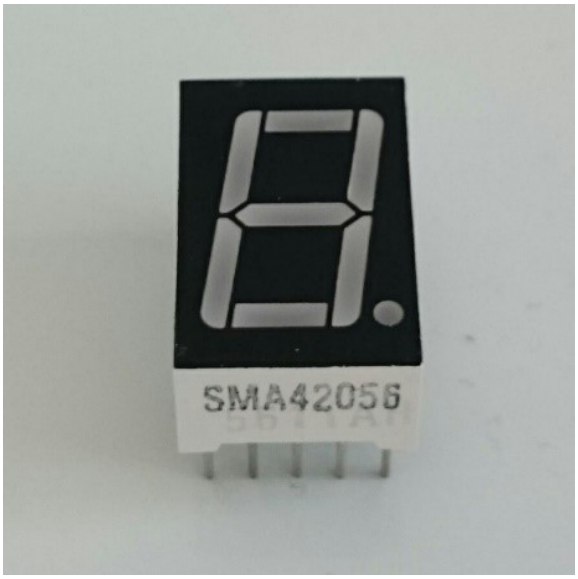
Ein Feld (englischer Fachbegriff Array) ist eine besondere Möglichkeit, um Daten abzuspeichern. Dabei ist ein Array vergleichbar mit einer Liste von Variablen. So kann man beispielsweise ein Array vom Datentyp int erstellen mit zehn Elementen. Dadurch hat man zehn Variablen vom Datentyp int, die sehr leicht anzusprechen sind: NameDesArrays[Index]. Dabei haben die einzelnen Speicher bei insgesamt zehn Stück die Indizes null bis neun. Der Vorteil dabei ist, dass man den Index im Programm auch als Variable angeben kann. Dadurch kann man abhängig von einer Berechnung auch erst zur Laufzeit des Programms entscheiden welcher Wert benötigt wird.

Um ein Array zu erstellen muss man noch vor dem setup Teil den Datentyp, den Namen des Arrays und die Anzahl der Plätze angeben: int testArray[10]; Der testArray hätte Platz für zehn Variablen vom Datentyp int. Alternativ kann man auch schon zu Beginn des Programms beim Erstellen des Arrays die Inhalte festlegen: int testArray[10] = {2, 4, 5, 8, 35, 2, 17, 6, 8, 8}; Das Abspeichern beziehungsweise Auslesen von Werten eines Arrays ist dabei ähnlich wie bei Variablen: testArray[4] = 11 zum Abspeichern oder zum Auslesen testArray[8].

2.5 7-Segment-Display

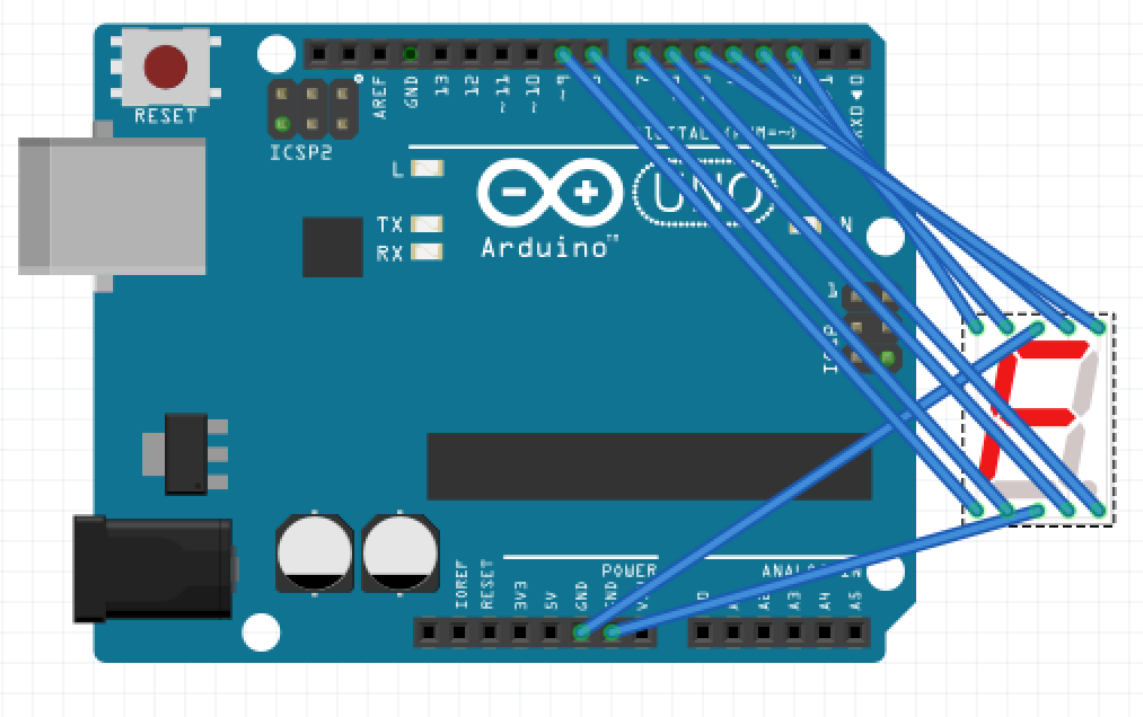
Ein 7-Segment-Display ist ein Display bestehend aus sieben LED-Strichen, die wie eine acht angeordnet sind. Dadurch können alle Ziffern eindeutig dargestellt werden, je nachdem welche LED-

Striche gerade leuchten. Viele 7-Segment-Displays haben zusätzlich auch noch einen Punkt, um beispielsweise ein Komma darzustellen.



7-Segment Display

Um die Verbindung des Displays mit dem Arduino zu vereinfachen haben 7-Segment-Displays meist einen gemeinsamen Ground/Minuspol (Common Anode Display). Dieser ist meist der mittlere Pin, sowohl in der oberen als auch in der unteren Reihe und muss daher mit dem Ground Pin des Arduinos (GND) verbunden werden. Die acht anderen Pins müssen mit den digitalen I/O Pins des Arduinos verbunden werden.



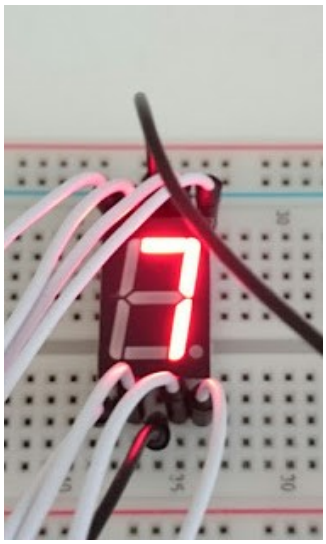
7-Segment Display mit dem Arduino Uno verbinden

Über die acht anderen Pins steuert man die acht LEDs des Displays wobei hier die Anordnung welcher Strich über welchen Pin angesteuert wird leider von Hersteller zu Hersteller variieren kann. Daher müssen wir zunächst einmal feststellen welcher digital Ausgang nun welchen LED-Strich steuert. Dafür kann man folgenden Sketch benutzen:

```
void setup() {
        int i;
        Serial.begin(9600);
        for(int i=2; i<10; i++)
        {
                Serial.println(i);

                pinMode(i, OUTPUT);

                digitalWrite(i, HIGH);
                delay(5000);
```

```
                digitalWrite(i, LOW);
        }
}

void loop() {
        //keine zu wiederholende Aktion
}
```

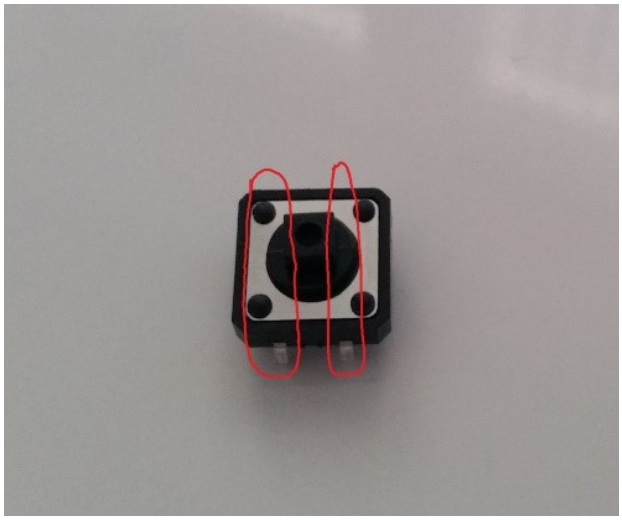
Hierbei werden nacheinander Pin 2 bis 9 für fünf Sekunden auf HIGH gesetzt, wodurch der jeweilige LED-Strich leuchtet. Mit diesem Wissen kann man nun eine Ziffer auf einem 7-Segment-Display anzeigen lassen, indem man nur die Pins der gewünschten Striche auf HIGH setzt und den Rest am Arduino auf LOW belässt. Hierbei bietet es sich auch an für jede Ziffer eine eigene Methode zu schreiben in der die Ausgänge entsprechend geschaltet werden. Dadurch muss man dann beispielsweise um die Ziffer 7 anzuzeigen nur noch die Methode sieben_anzeigen() aufrufen und sich nicht immer merken, wie die Pins geschaltet sein müssen, um eine sieben anzuzeigen. Wie man eine eigene Methode definiert wird in Kapitel 3.1 erklärt. Zusätzlich wird der jeweilige Pin auch in der seriellen Konsole am PC angezeigt (vgl. Kapitel 7).



7-Segment Display im Einsatz

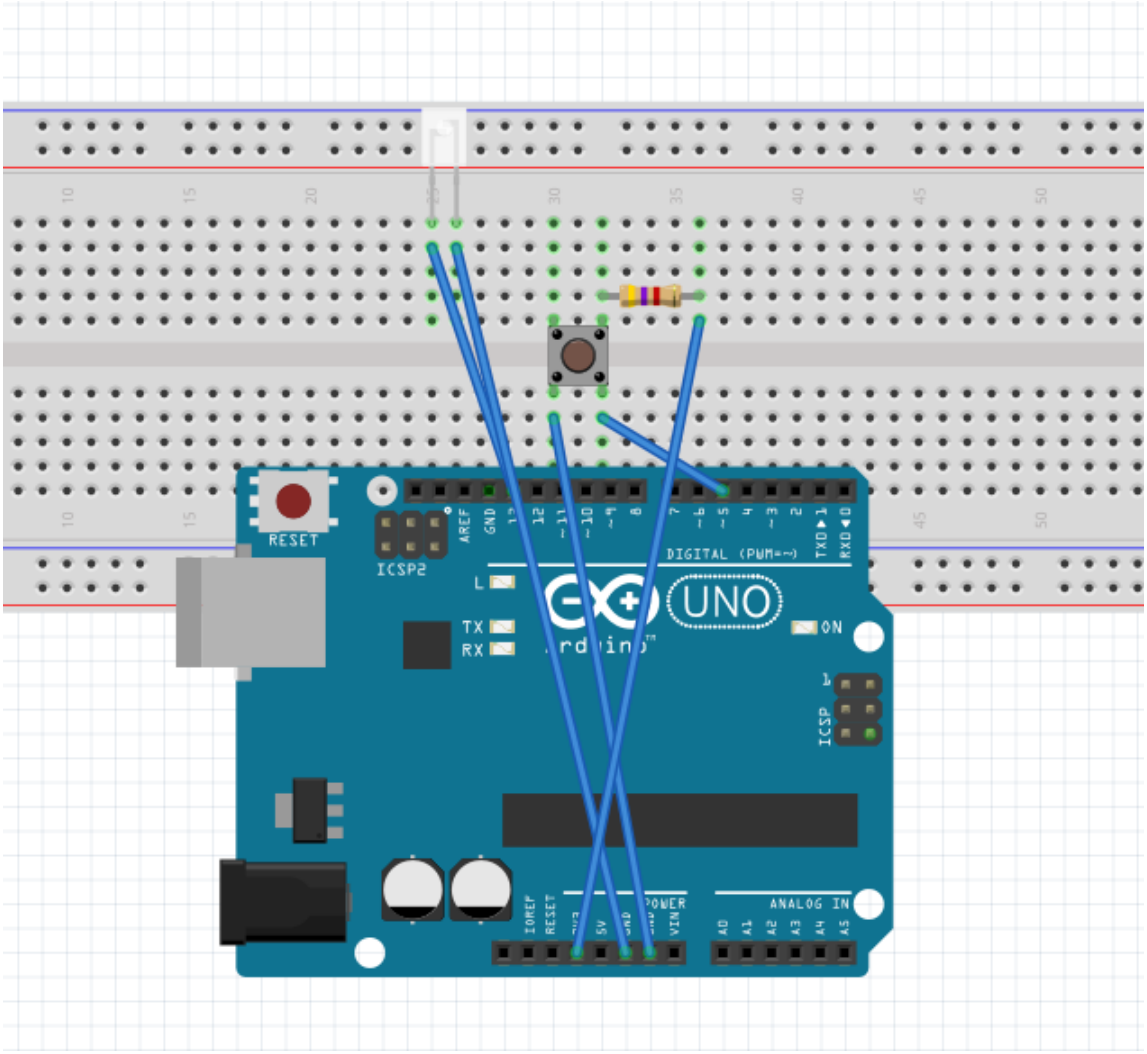
2.6 Benutzung eines Buttons – if Abfrage

Ein Pushbutton ist eine sehr häufig anzutreffende Art, über die der Benutzer mit einer Maschine interagieren kann. Dabei haben Pushbuttons üblicherweise vier Beine, wobei die sich gegenüberliegenden Pins leitend miteinander verbunden sind.



„Beine“ links vorne und links hinten, sowie rechts vorne und rechts hinten leitend verbunden

Normalerweise gibt es keine Verbindung zwischen dem linken und dem rechten Stromkreis, wird der Pushbutton jedoch gedrückt so werden sie verbunden. Dieses Verhalten kann man sich zunutze machen, indem man an eine Seite eine Verbindung zu einem digitalen Eingang des Arduinos herstellt, und gleichzeitig über einen Widerstand diese Seite mit +3.3V verbindet. Die andere Seite wird mit dem Ground verbunden. Dadurch liegt durchgehend eine schwach positive Spannung am Eingang des Arduino an. Sobald nun jedoch der KnopF gedrückt wird, wird der Eingang durch den Ground auf LOW gezogen.



Button an Pin 5 und LED an Pin 13

Zur Veranschaulichung des Projekts soll die LED des Arduinos an Pin 13 angeschaltet werden, solange der Button gedrückt ist. Dazu muss im Sketch überprüft werden, ob der Button gerade gedrückt ist, also der Eingangspin auf LOW ist, und dementsprechend Pin 13 auf LOW oder HIGH geschaltet werden. Zur Überprüfung einer Bedingung verwendet man sogenannte if-Abfragen. Diese bestehen aus einer Bedingung in runden Klammern und einem Code innerhalb von geschweiften Klammern. Ist die Bedingung wahr, so wird der Code ausgeführt, wenn sie falsch ist, wird der Code übersprungen. Zusätzlich kann man auch noch ein else inklusive Code in geschweiften Klammern angeben. Dieser Programmteil wird ausgeführt, falls die Bedingung falsch ist. Um den aktuellen Status eines als INPUT definierten Pins auszulesen, verwendet man die Methode (= den Befehl):

```
digitalRead(Pinnummer);
```

Dieser gibt einen Wahrheitswert (boolean) zurück, also wahr oder falsch zurück je nachdem ob LOW oder HIGH am Eingang anliegt. Da wir durch diesen Befehl direkt einen fertigen Wahrheitswert erhalten, können wir das Auslesen des Pins direkt als Bedingung für die if Schleife in die runden Klammern schreiben, müssen jedoch darauf achten, dass wahr (HIGH) bedeutet, der Knopf also gerade nicht gedrückt ist (Pin 13 auf LOW), und stattdessen falsch (LOW) bedeutet, dass der Button gedrückt ist. Der gesamte Sketch für die, über einen Pushbutton gesteuerte, LED sieht dann so aus:

```
void setup() {
  pinMode(5, INPUT);
  pinMode(13, OUTPUT);
}
```

```
void loop() {
  if(digitalRead(5))        //Wenn 5 HIGH -> Knopf
  {                         // nicht gedrueckt
    digitalWrite(13, LOW);
  }
  else                     //Sonst
  {
    digitalWrite(13, HIGH);
  }
}
```

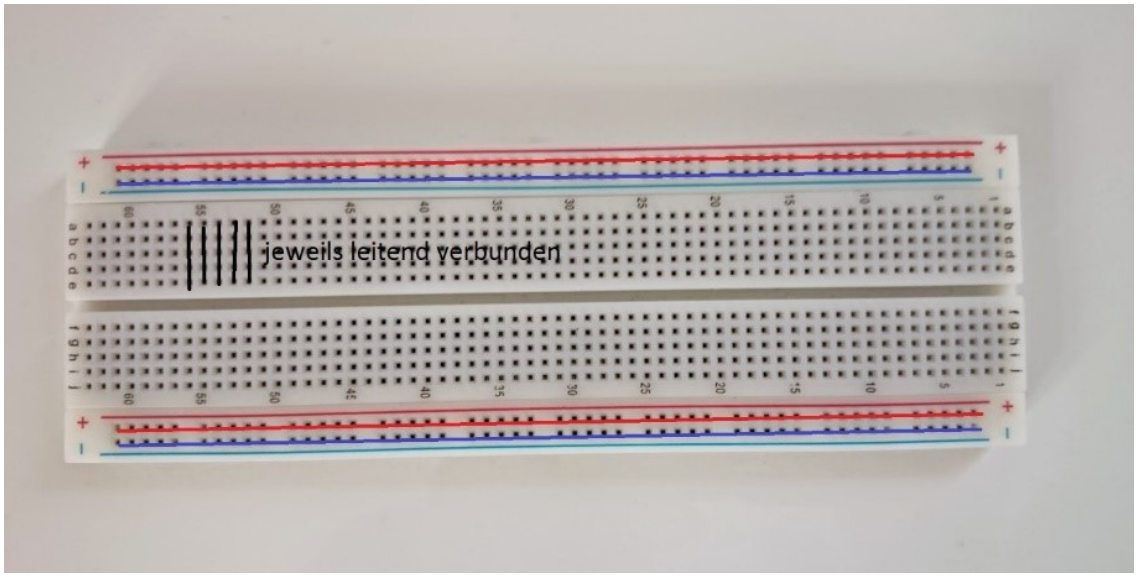
2.7 LCD Textdisplay für den Arduino

Mittels eines LCD Textdisplays kann man über den Arduino einen Text ausgeben, um mit der Außenwelt zu kommunizieren. Dabei ist die Verwendung eines solchen Displays dank eines Tricks relativ einfach: Bibliotheken. Eine Bibliothek ist im Programmiersprachgebrauch nicht ein Ort, mit vielen Büchern die man sich ausleihen kann, sondern eine Art fertiges Unterprogramm, welches man einfach nur in sein Programm einbinden muss. Dadurch muss man nicht die gesamte hardwarenahe Programmierung des Displays am Arduino noch einmal programmieren, sondern kann über Befehle der Bibliothek die gewünschten Aktionen durchführen, und lässt die Bibliothek die eigentliche Arbeit machen. Dadurch gelingt es über den Befehl

lcd.print(Text);

einen Text auf das LCD Textdisplay zu schreiben, wobei man natürlich darauf achten sollte, die maximale Anzahl an Zeichen des jeweiligen Displays einzuhalten. Die häufigsten Displays sind LCDs mit 16 Zeichen pro Zeile und 2 Zeilen, oder mit 20 Zeichen pro Zeile und 4 Zeilen. Die Verbindung ist leider relativ kompliziert da über ein Dutzend Verbindungen getätigt werden müssen.

Pin 1, Pin 3, Pin 5 und Pin 16 des LCD Textdisplays müssen mit dem Ground des Arduinos verbunden werden. Hierzu kann man sich des Tricks bedienen, nur einmal die Verbindung vom Ground des Arduinos zur „Minus-Lane“ (blau) des Breadboards herzustellen, und von dort die Pins mit dem Ground zu verbinden.



Pin 2 des Displays muss mit den +5V des Arduinos verbunden werden. Pin 15 des Displays ist für die Hintergrundbeleuchtung zuständig und sollte für eine angenehme Helligkeit mit den +3.3V des Arduinos verbunden werden. Zur Datenübertragung muss Pin 4 des Displays, der RS Pin, zu digital Pin 12, Pin 6 des Displays, als E markiert, zu Pin 11, D4 also Pin 11 des Displays zum Arduino Pin 5, D5 am Display zu Pin 4 des Arduino, D6 zu Pin 3 des Arduino und D7 zu Pin 2 des Arduino. Im Sketch muss zunächst die passende Bibliothek, in unserem Fall „LiquidCrystal“ nachgeladen werden. Da die Arduino IDE diese Bibliothek standardmäßig nicht mitinstalliert hat, muss sie über den Library Manager nachgeladen werden. Diesen findest du unter Sketch > Bibliothek einbinden > Bibliotheken verwalten. Im sich öffnenden

Pop Up kannst du einfach nach LiquidCrystal suchen und auf „Installieren“ klicken. Nach diesem Schritt können wir die Bibliothek in unseren Sketch noch vor dem setup Teil über

```
#include <Name der Bibliothek.h>
```

die Bibliothek (LiquidCrystal) nachladen. Diese Bibliothek basiert wie viele Bibliotheken und Programme auf dem Prinzip der Objektorientierung, bei dem reale Dinge durch Objekte repräsentiert werden. Um also unser LCD Display zu verwenden, müssen wir zunächst ein Objekt im Programm erzeugen, welches dieses Display repräsentiert, was in unserem Fall über den Befehl

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

funktioniert. Allgemein sieht das Erzeugen eines Objekts so aus:

```
Objekttyp Objektbezeichner(eventuelle Eigenschaften des Objekts);
```

In unserem Fall sind die Eigenschaften des Displays die Pins, über die das Display mit dem Arduino Uno verbunden wurde. Nun muss man nur noch im setup Teil die Größe des Displays über den Befehl lcd.begin(16,2) oder lcd.begin(20,4), je nachdem wie groß es ist, angeben und über lcd.setCursor(0,0) den Cursor links oben setzen. Über den Befehl lcd.print(Text) kann man nun einen Text auf dem Display ausgeben, beginnend an der aktuellen Position des Cursors. Ein Beispielsketch der Hello World auf dem LCD Textdisplay ausgibt sieht so aus:

```
#include <LiquidCrystal.h>

LiquidCrystal lcddisplay(12, 11, 5, 4, 3, 2);

void setup()
{
  lcddisplay.begin(16, 2);
  lcddisplay.print("Hello World");
}

void loop()
{
  //keine zu wiederholenden Aufgaben
}
```

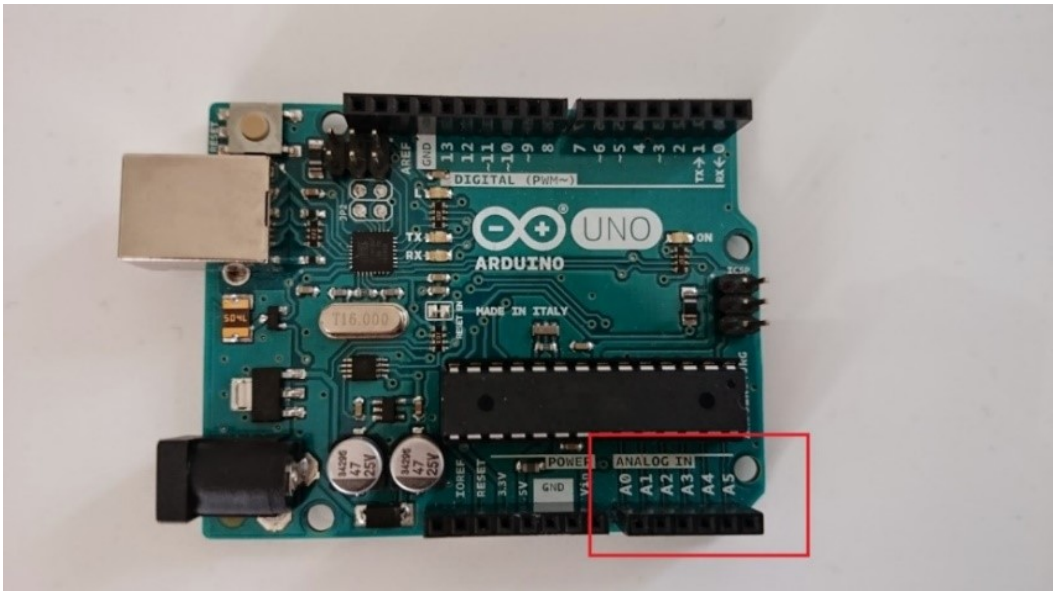
Wichtig im Zusammenhang mit dem Display ist noch zu wissen, dass es unproblematisch möglich ist, Zeichen zu überschreiben. Man kann jedoch auch einfach über den Befehl lcd.clear(), den gesamten Inhalt löschen, und den Cursor nach links oben (0, 0) setzt.

2.8 Analoge Signale vom Potentiometer

Ein Potentiometer ist ein Widerstand welchen der Benutzer manuell einstellen kann. Dabei gibt es verschiedene Bauarten, beispielsweise das Drehpotentiometer oder ein Schiebepotentiometer. Bei einem Drehpotentiometer beispielsweise kann der Benutzer durch das Drehen den Widerstand so groß machen, dass kein Strom mehr durchkommt, oder so klein drehen, dass das Potentiometer keinen wirklichen Widerstand mehr darstellt.

Drehpotentiometer

Um dieses analoge Signal am Arduino auszulesen, benötigt man die analogen Eingänge, welche sich gegenüber den digital I/O (Input/Output) Pins des Arduinos befinden.



Analoge Eingänge des Arduino Uno Version 3

Diese Pins können bauartbedingt unterschiedliche Spannungen in 1024 verschiedenen Abstufungen messen von 0 gleich 0V und

1023 gleich der Spannung des Arduinos, laut Datenblatt also 5V in Realität etwa 4,8V. Das Auslesen dieser Pins funktioniert über den Befehl

```
analogRead(Pinnummer);
```

Dabei haben die analogen Eingänge zur Unterscheidung von den 14 digitalen I/O Pins ein A vor ihrer Pinnummer, die aktuelle Spannung am analog Pin 3 würde man also über

```
analogRead(A3);
```

auslesen. Die Verbindung des Drehpotentiometers mit dem Arduino ist relativ einfach: Der mittlere Pin ist der Signalpin, welcher mit einem analogen Eingang verbunden werden muss, der linke Pin muss mit dem Ground verbunden werden, und der rechte mit den +5V des Arduinos. Um beispielsweise eine LED, angeschlossen an einen PWM Pin des Arduinos, entsprechend dem Drehpotentiometer zu steuern, sähe der Sketch so aus, wobei man darauf achten muss das Eingangssignal (0-1023) in das andere Format des Ausgangssignals umzurechnen (0 -255):

```
void setup()
{
  pinMode(A3, INPUT);
  pinMode(3, OUTPUT);
}

void loop()
{
  analogWrite(3, (analogRead(A3)*0.249));
  delay(50);
}
```

Hat Ihnen diese Leseprobe gefallen?

Bestellen Sie das Buch als Taschenbuch komfortabel auf Amazon!

eBook: https://amzn.to/2T1IzmU

Taschenbuch: https://amzn.to/2SniwBG

Möchten Sie über neue Bücher und Sonderangebote vom BMU Verlag informiert werden?

Tragen Sie sich jetzt in unseren E-Mail Newsletter ein: https://bmu-verlag.de/