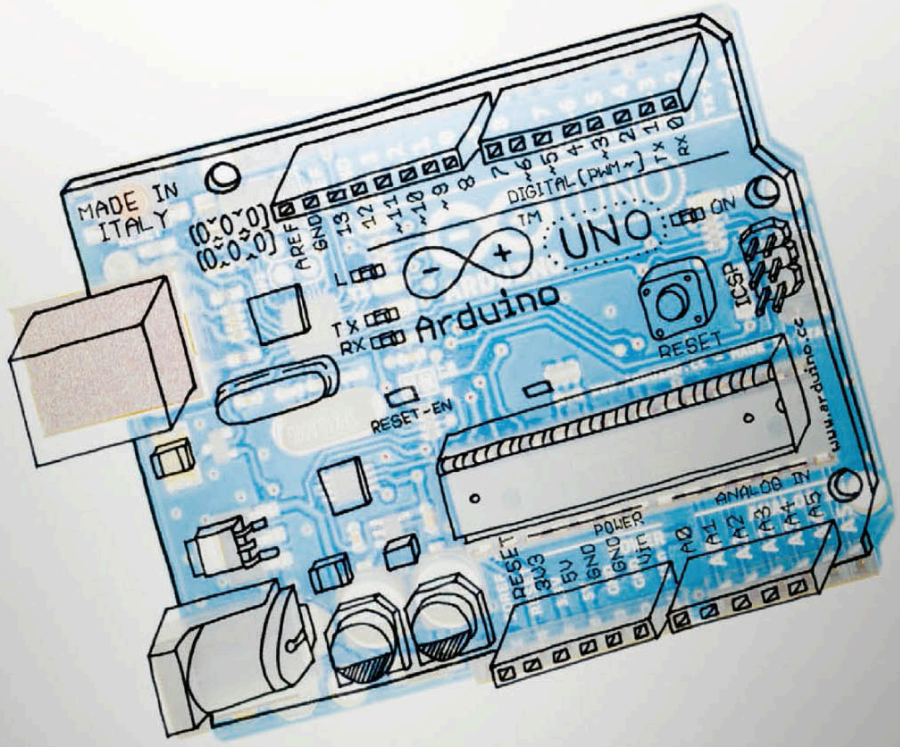
Massimo Banzi · Michael Shiloh

ARDUINO DEIN EINSTIEG

Die Open-Source-Plattform für Elektronik-Prototypen

Übersetzung der 4. US-Auflage



Make:

dpunkt.verlag

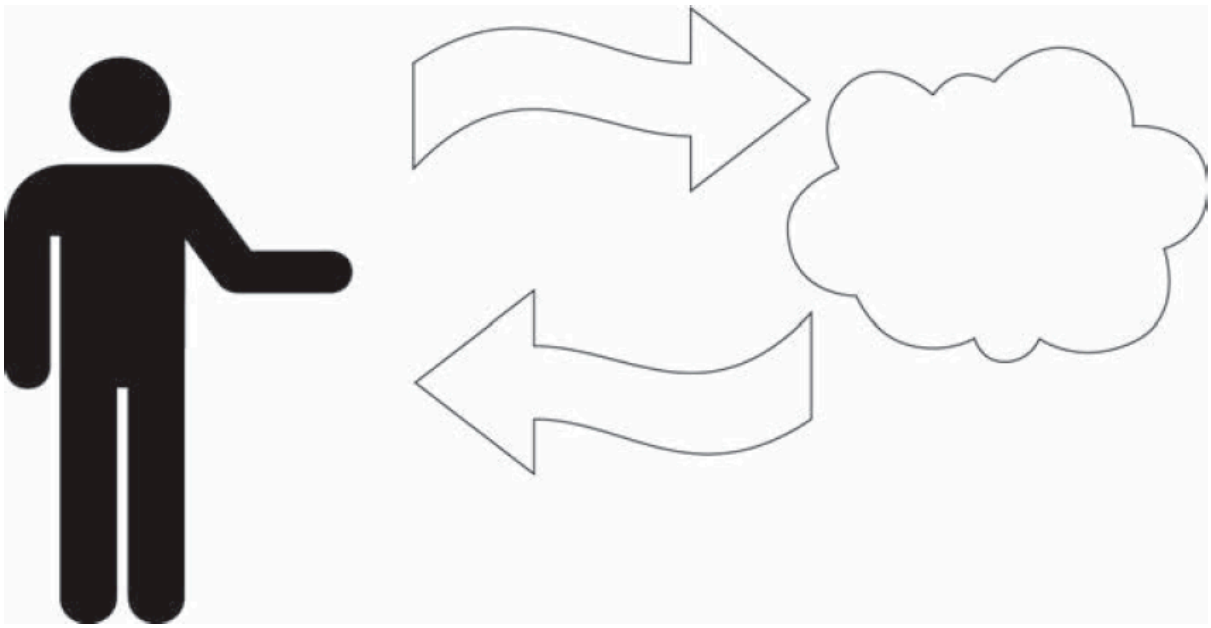
Inhalt

4 Jetzt geht es wirklich los mit Arduino

Jetzt wirst du lernen, wie man ein interaktives Gerät baut und programmiert.

Anatomie eines interaktiven Gerätes

Alle Objekte, die wir unter Verwendung von Arduino bauen werden, folgen einem ganz einfachen Muster, das wir *interaktives Gerät* nennen. Das interaktive Gerät ist eine elektronische Schaltung, die die Umwelt mithilfe von *Sensoren* (elektronische Komponenten, die Messwerte aus der realen Welt in elektrische Signale umwandeln) erkennen kann. Das Gerät verarbeitet diese Informationen, die es von den Sensoren erhält, mit dem Verhalten, das in der Software beschrieben ist. Das Gerät ist dann in der Lage, durch den Einsatz von *Aktoren* (elektronische Komponenten, die ein elektrisches Signal in eine physische Handlung umwandeln) mit der Welt zu interagieren.



Sensoren und Aktoren

Sensoren und Aktoren sind elektronische Komponenten, die einem Elektronikteil die Interaktion mit der Welt ermöglichen.

Da der Mikrocontroller ein kleiner, einfacher Computer ist, kann er nur elektrische Signale verarbeiten (ein bisschen wie elektrische Impulse, die zwischen Neuronen in unserem Gehirn versendet werden). Damit er Licht, Temperatur oder andere physikalische Mengen erkennen kann, benötigt er etwas, das sie in Elektrizität umwandelt. In unserem Körper wandelt beispielsweise das Auge Licht in Signale um, die mittels Nerven ans Gehirn geschickt werden. In der Elektronik können wir ein einfaches Bauelement namens *Fotowiderstand* (englisch Light Dependent Resistor, *LDR*) nutzen, das die auftreffende Lichtmenge messen und sie als ein Signal melden kann, das vom Mikrocontroller verstanden wird.

Sobald die Sensoren abgelesen worden sind, hat der Widerstand die Information, die er für eine Reaktion benötigt. Für den Entscheidungsfindungsprozess ist der Mikrocontroller zuständig, die Reaktion wird von Aktoren umgesetzt. In unserem Körper empfangen zum Beispiel Muskeln elektrische Signale vom Gehirn und wandeln sie in eine Bewegung um. In der elektronischen Welt könnten diese Funktionen durch eine Lampe oder einen Elektromotor ausgeführt werden.

In den folgenden Abschnitten lernst du, wie du verschiedene Arten von Sensoren liest und unterschiedliche Aktoren steuerst.

Blinkende LED

Der Sketch für die blinkende LED ist das erste Programm, das du laufen lassen solltest, um zu testen, ob dein Arduino-Board korrekt konfiguriert ist. In der Regel ist es auch die allererste Programmierübung, die jemand beim Erlernen des Programmierens eines Mikrocontrollers durchführt. Eine *Leuchtdiode* (LED) ist eine kleine Elektronikkomponente, ähnlich einer Glühbirne, aber effizienter und benötigt weniger Spannung für den Betrieb.

Auf deinem Arduino-Board ist eine LED bereits installiert. Sie ist auf dem Board mit L gekennzeichnet. Diese vorinstallierte LED ist am Pin Nummer 13 angeschlossen. Merk dir diese Nummer, denn wir werden sie später brauchen. Du kannst auch deine eigene LED[1] hinzufügen – schließe sie wie in Abb. 4–1 gezeigt an. Beachte, dass sie an der Anschlussbuchse mit der Nummer 13 eingesteckt wird.

Wenn du die LED längere Zeit leuchten lassen möchtest, solltest du, wie unter »Lichtsteuerung mit PWM« auf Seite 60 beschrieben, einen Widerstand verwenden.

K kennzeichnet die Kathode (negativ) bzw. den kürzeren Draht, A kennzeichnet die Anode (positiv) bzw. den längeren Draht.

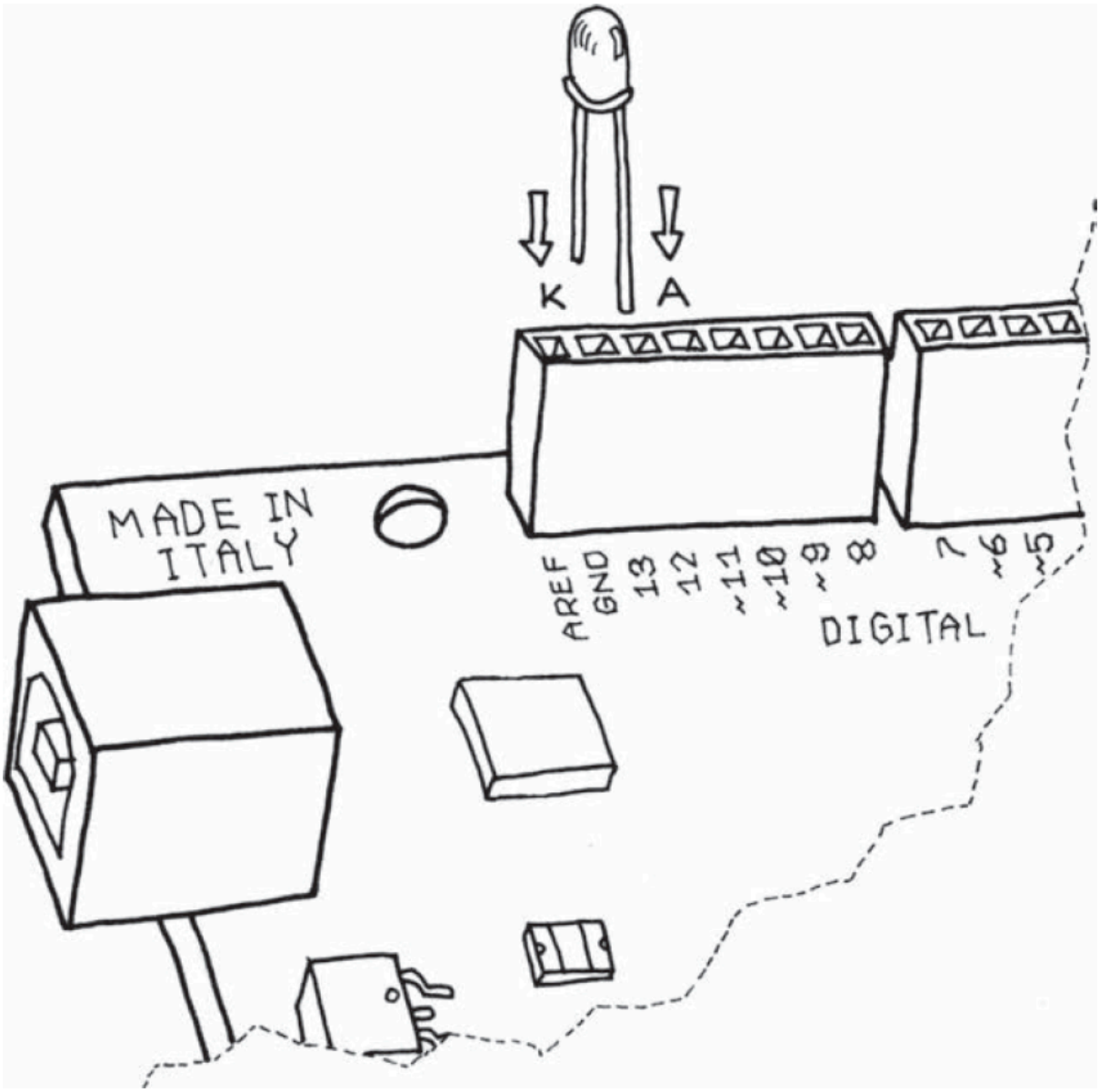


Abb. 4–1 Anschluss einer LED am Arduino

Sobald die LED angeschlossen ist, musst du Arduino sagen, was es tun soll. Das erfolgt durch Code: eine Liste mit Anweisungen, die du dem Mikrocontroller gibst, damit er tut, was du möchtest. (Die Wörter *Code, Programm* und *Sketch* sind alles Begriffe, die sich auf dieselbe Liste mit Anweisungen beziehen.)

Ruf auf deinem Computer die Arduino-IDE auf (auf dem Mac sollte sie im *Anwendungsordner* sein; in Windows findet sich entweder der Shortcut auf deinem Desktop oder im Startmenü). Wähle *File* ⟶ *New*, und du wirst gebeten, einen Namen für den *Sketch*-Ordner zu wählen: Hier wird dein Arduino-Sketch gespeichert. Nenne ihn *Blinkende LED* und klicke auf OK. Gib dann den

folgenden Sketch (Beispiel 4–1) in den Arduino Sketch Editor ein (das Hauptfenster der Arduino-IDE). Du kannst ihn auch über den Link für die Beispielcodes von der Katalogseite (*https://makezine.com/go/arduino-4e-github/*) der englischen Originalausgabe des Buches herunterladen.

Du kannst diesen Sketch auch einfach durch Anklicken von *File* ⟶ *Example* ⟶ *01.Basics* ⟶ *Blink* laden, aber du lernst besser, wenn du ihn selbst eingibst. Das integrierte Beispiel mag ein wenig anders sein, aber im Grunde genommen tut es genau dasselbe.

Es sollte wie in Abb. 4–2 dargestellt erscheinen.

Beispiel 4–1 Blinkende LED

```
// Blinking LED

const int LED = 13; // LED connected to

                    // digital pin 13

void setup()

{

  pinMode(LED, OUTPUT);      // sets the digital

                          // pin as output

}

void loop()

{

  digitalWrite(LED, HIGH);   // turns the LED on

  delay(1000);               // waits for a second
```

```
  digitalWrite(LED, LOW);    // turns the LED off

  delay(1000);               // waits for a second

}
```

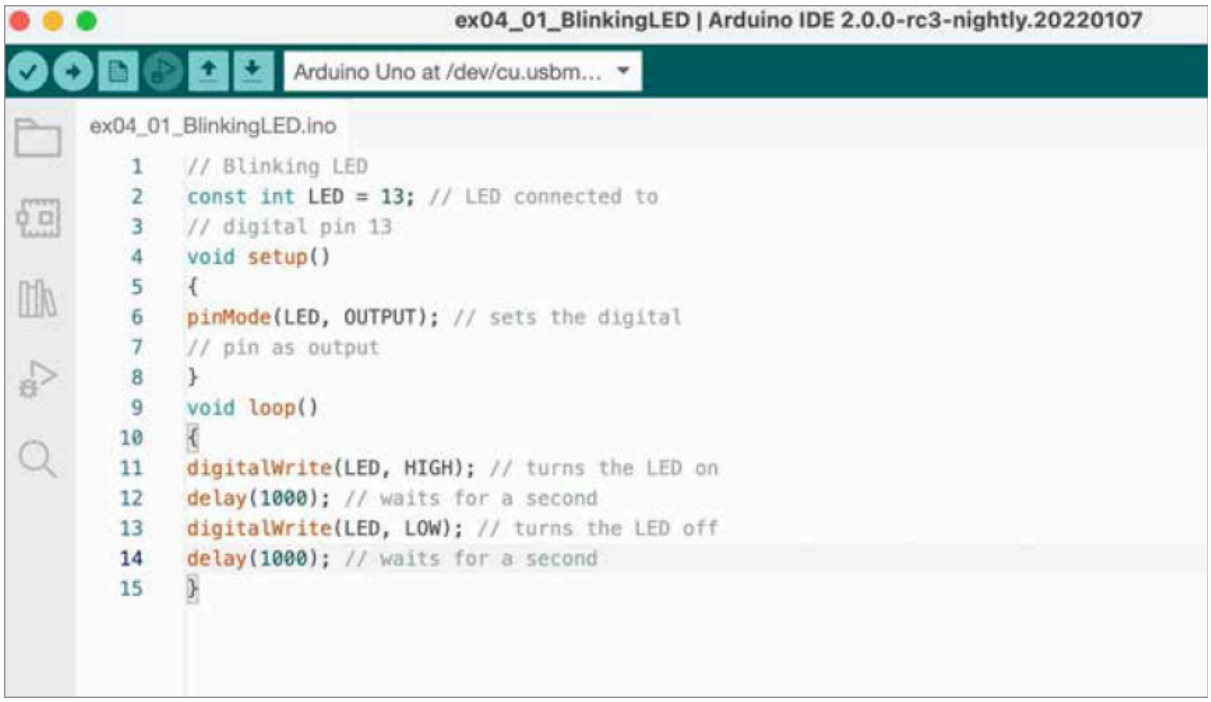


Abb. 4–2 Die Arduino-IDE mit deinem ersten geladenen Sketch

Nun, da der Code in deiner IDE ist, musst du überprüfen, ob er korrekt ist. Klicke auf die Verify-Schaltfläche (Abb. 4–2 zeigt ihre Platzierung – der Haken oben links); wenn alles korrekt ist, siehst du am unteren Rand der Arduino-IDE die Mitteilung »Done compiling«. Diese Mitteilung bedeutet, dass die Arduino-IDE deinen Sketch in ein ausführbares Programm übersetzt hat, das vom Board ausgeführt werden kann – ein bisschen wie eine *.exe*-Datei in Windows oder eine *.app*-Datei auf einem Mac.

Wenn du eine Fehlermeldung erhältst, hast du höchstwahrscheinlich bei der Eingabe des Codes einen Fehler gemacht. Schau jede Zeile sehr sorgfältig durch und prüfe in Bezug auf jedes einzelne Zeichen, insbesondere Symbole wie Klammern, Semikola und Kommata. Achte darauf, dass du Groß- und Kleinschreibung genau kopiert und den Buchstaben O bzw. die Ziffer 0 korrekt verwendet hast.

Sobald dein Code als korrekt bestätigt worden ist, kannst du ihn durch Anklicken der Upload-Schaltfläche neben Verify auf das Board laden (siehe Abb. 4–2). Dies wird die IDE auffordern, den Upload-Vorgang zu beginnen, der zuerst das Arduino-Board zurücksetzt, es zwingt, die aktuelle Tätigkeit einzustellen und auf Anweisungen zu achten, die vom USB-Port kommen. Die Arduino-IDE schickt dann den Sketch an das Arduino-Board, das den Sketch in seinem Dauerspeicher ablegt. Sobald die IDE den gesamten Sketch geschickt hat, beginnt das Arduino-Board, deinen Sketch auszuführen.

Das passiert recht schnell. Wenn du deine Augen auf den unteren Bereich der Arduino-IDE gerichtet hältst, wirst du sehen, wie einige Mitteilungen im schwarzen Bereich unten im Fenster erscheinen, und genau darüber siehst du möglicherweise die Mitteilung »Compiling«, dann »Uploading« und schließlich »Done uploading«, um dich zu informieren, dass der Vorgang korrekt abgeschlossen worden ist.

Auf dem Arduino-Board befinden sich zwei LEDs, gekennzeichnet mit RX und TX; diese leuchten jedes Mal auf, wenn ein Byte vom Board versendet oder empfangen wird. Während des Upload-Vorgangs flackern sie. Das passiert auch sehr schnell; sofern du also nicht zur rechten Zeit dein Arduino-Board anschaust, könntest du es verpassen.

Wenn du die LEDs nicht flackern siehst oder wenn statt »Done uploading« eine Fehlermeldung erscheint, dann liegt ein Kommunikationsproblem zwischen deinem Computer und Arduino vor. Vergewissere dich, dass du im Menü *Tools* ⟶ *Serial Port* den richtigen seriellen Port gewählt hast (siehe Kapitel 3). Prüfe auch das Menü *Tools* ⟶ *Board*, um zu bestätigen, dass dort das korrekte Arduino-Modell gewählt ist.

Wenn du weiterhin Probleme hast, sieh in Kapitel 11 nach.

Sobald der Code in deinem Arduino-Board ist, bleibt er dort, bis du einen anderen Sketch lädst. Der Sketch überlebt, wenn das Board zurückgesetzt oder ausgeschaltet wird, etwa so wie Daten auf der Festplatte deines Computers.

Angenommen, der Sketch ist korrekt hochgeladen worden, dann siehst du, wie sich die LED *L* für eine Sekunde ein- und dann für eine Sekunde ausschaltet. Wenn du, wie zuvor in Abb. 4–1 gezeigt, eine separate LED installiert hast, dann blinkt diese ebenfalls. Was du hier geschrieben und ausgeführt hast, ist ein *Computerprogramm* oder *Sketch*, wie Arduino-Programme genannt werden. Der Arduino ist, wie wir bereits erwähnt haben, ein kleiner Computer, und er kann so programmiert werden, dass er das tut, was du von ihm verlangst. Das erfolgt durch Eingabe einer Reihe von Anweisungen in der Arduino-IDE, die sie dann in eine Anwendung für dein Arduino-Board umwandelt.

Als Nächstes zeigen wir dir, wie man einen Sketch versteht. Zunächst einmal führt Arduino den Code schrittweise von oben nach unten aus, das heißt, die erste Zeile ganz oben wird als erste gelesen; dann bewegt es sich abwärts, so wie du dieses Buch liest, vom Anfang jeder Seite bis zum Ende.

Reich mir den Parmesan

Beachte die geschweiften Klammern, die verwendet werden, um Codezeilen zu gruppieren. Diese sind besonders hilfreich, wenn du einer Gruppe von Anweisungen einen Namen zuweisen möchtest. Wenn du beim Essen bist und zu jemandem sagst: »Reich mir bitte den Parmesankäse«, dann leitet das eine Reihe von Handlungen ein, die durch den kurzen Satz, den du gerade sagtest, zusammengefasst werden. Für uns als Menschen geht das völlig unbewusst, aber all die einzelnen winzigen Aktionen, die hierfür erforderlich sind, müssen dem Arduino mitgeteilt werden, da er nicht so leistungsfähig ist wie unser Gehirn. Daher stellen wir zum Gruppieren von Anweisungen ein *{* vor den Codeblock und ein *}* dahinter.

Du siehst, dass hier auf diese Weise zwei Codeblocks definiert sind. Vor jedem befinden sich ein paar merkwürdige Wörter:

```
void setup()
```

Diese Zeile gibt einem Codeblock einen Namen. Wenn du eine Liste mit Anweisungen schreiben wolltest, die Arduino beibringt, wie der Parmesan weiterzureichen ist, dann würdest du am Anfang eines Blocks `void passTheParmesan()` schreiben, und dieser Block wäre eine Anweisung, die du von jeder beliebigen Stelle im Arduino-Code aufrufen könntest. Diese Blöcke nennen sich *Funktionen*. Jetzt, da du aus diesem Codeblock eine Funktion gemacht hast, kannst du an jeder beliebigen Stelle in deinem Sketch passTheParmesan() schreiben, und Arduino springt zur `passTheParmesan()`-Funktion, führt jene Anweisungen aus, springt dann zurück zur ursprünglichen Stelle und macht dort weiter.

Dies weist auf etwas Wichtiges in jedem Arduino-Programm hin. Arduino kann nur eine Sache auf einmal machen, eine Anweisung nach der anderen ausführen. Während Arduino dein Programm Zeile für Zeile ablaufen lässt, führt es nur *diese eine* Zeile *aus*. Wenn es zu einer Funktion springt, führt es diese Funktion aus, Zeile für Zeile, bevor es wieder dorthin zurückkehrt, wo es war. Arduino kann keine zwei Sätze von Anweisungen gleichzeitig ausführen.

Arduino ist nicht für Feiglinge

Arduino erwartet stets, dass du zwei Funktionen erstellt hast: eine namens `setup()` und eine namens `loop()`.

`setup()` ist, wo du den ganzen Code ablegst, der am Anfang deines Programms ausgeführt werden soll, und `loop()` enthält den Kern deines Programms, der immer wieder ausgeführt wird. Das ist so, weil Arduino nicht wie dein normaler Computer ist – er kann nicht mehrere Programme gleichzeitig ausführen, und Programme können nicht abschalten. Wenn du dem Board Strom zuführst, läuft der Code; wenn du ihn stoppen willst, schalte ihn einfach ab.

Echte Tüftler schreiben Kommentare

Jeglicher Text, der mit // beginnt, wird von Arduino ignoriert. Diese Zeilen sind *Kommentare*, also Hinweise, die du für dich selbst im Programm hinterlässt, damit du dich erinnerst, was du getan hast, als du es schriebst, oder für jemand anders, damit er oder sie deinen Code versteht.

Es kommt sehr häufig vor (wir wissen das, weil es uns ständig passiert), dass du einen Teil eines Programms schreibst, es auf das Board lädst und dir sagst: »Okay, mit diesem Kram muss ich mich nie wieder beschäftigen!«, um sechs Monate später festzustellen, dass du den Code aktualisieren oder einen Fehler beheben musst. An diesem Punkt öffnest du das Programm, und wenn du keine Kommentare im ursprünglichen Programm eingefügt hast, denkst du: »Wow, was für ein Mist! Wo fange ich nur an?« Im weiteren Verlauf wirst du einige Tricks kennenlernen, wie du das Programm lesbarer gestalten und leichter pflegen kannst.

Der Code, Schritt für Schritt

> *Zuerst magst du diese Erläuterungen als überflüssig erachten – etwa so wie ich damals, als ich mich in der Schule mit Dantes* Divina Commedia *beschäftigen musste (jeder italienische Schüler muss da durch, wie auch durch ein weiteres Buch mit dem Titel* I promessi sposi, *auf Deutsch* Die Verlobten *– oh, welch Albträume). Für jede einzelne Zeile der Werke gab es hundert Zeilen mit Kommentaren! Allerdings wird die Erläuterung hier wesentlich nützlicher sein, wenn du dich dem Schreiben deiner eigenen Programme zuwendest.*
>
> *– Massimo*

```
// Blinking LED
```

Ein Kommentar ist für uns eine nützliche Methode, kleine Hinweise zu schreiben. Der vorangestellte Titel-Kommentar erinnert uns einfach, dass dieses Programm, Beispiel 4–1, eine LED blinken lässt.

```
const int LED = 13; // LED connected to
                    // digital pin 13
```

const int bedeutet, dass *LED* der Name einer Ganzzahl ist, die sich nicht ändern lässt (d. h. eine Konstante), deren Wert auf 13 eingestellt ist. Das ist wie ein automatisches Suchen und Ersetzen für deinen Code; in diesem Fall sagt er Arduino, bei jedem Auftauchen des Wortes *LED* die Zahl 13 zu schreiben.

Der Grund, warum wir die Zahl 13 brauchen, ist, dass die vorab installierte LED, die wir bereits erwähnten, am Arduino-Pin 13 angeschlossen ist. Eine geläufige Konvention ist, für Konstanten Großbuchstaben zu verwenden.

```
void setup()
```

Diese Zeile sagt Arduino, dass der nächste Codeblock eine Funktion namens `setup()` sein wird.

```
{
```

Mit dieser öffnenden geschweiften Klammer beginnt ein Codeblock.

```
pinMode(LED, OUTPUT);    // sets the digital
                         // pin as output
```

Abschließend eine wirklich interessante Anweisung! `pinMode()` sagt Arduino, wie es einen bestimmten Pin konfigurieren soll. Alle Pins des Arduino lassen sich entweder als Eingang oder als Ausgang nutzen, aber wir müssen Arduino sagen, wie wir den Pin zu nutzen gedenken.

In diesem Fall benötigen wir einen Ausgangs-Pin zum Steuern unserer LED.

`pinMode()` ist eine Funktion, und die innerhalb der Klammern angegebenen Wörter (oder Zahlen) werden als ihre *Argumente* bezeichnet. Argumente sind jede Art von Information, die eine Funktion zur Ausführung ihrer Aufgabe benötigt.

Die `pinMode()`-Funktion benötigt zwei Argumente. Das erste Argument sagt `pinMode()`, von welchem Pin wir reden, und das zweite Argument sagt `pinMode()`, ob wir diesen Pin als Eingang oder Ausgang nutzen wollen. `INPUT` (Eingang) and `OUTPUT` (Ausgang) sind vordefinierte Konstanten in der Arduino-Sprache.

Du erinnerst dich, dass das Wort *LED* der Name der Konstante ist, die auf die Nummer 13 eingestellt wurde, nämlich die Pin-Nummer, an der die LED angeschlossen ist. Damit ist das erste Argument `LED`, der Name der Konstante.

Das zweite Argument ist `OUTPUT`, denn wenn Arduino mit einem Aktor wie einer LED spricht, schickt es Informationen *hinaus*.

```
}
```

Diese schließende geschweifte Klammer kennzeichnet das Ende der `setup()`-Funktion.

```
void loop()

{
```

`loop()` ist die Stelle, an der du das wesentliche Verhalten deines interaktiven Gerätes bestimmst. Sie wird ständig wiederholt, bis du dem Board die Stromzufuhr entziehst.

```
digitalWrite(LED, HIGH);        // turns the LED on
```

Wie der Kommentar schon sagt, kann `digitalWrite()` jeden Pin ein- bzw. ausschalten, der als Ausgang konfiguriert worden ist. Wie wir gerade bei der `pinMode()`-Funktion sahen, erwartet auch `digitalWrite()` zwei Argumente, und wie bei der `pinMode()`-Funktion teilt das erste Argument

`digitalWrite()` `mit`, auf welchen Pin wir uns beziehen, und wiederum wie bei der Funktion `pinMode()` verwenden wir den Konstantennamen `LED` und beziehen uns so auf den Pin mit der Nummer 13, an den die vorinstallierte LED angeschlossen ist.

Das zweite Argument ist anders: In diesem Fall sagt das zweite Argument `digitalWrite()`, ob das Spannungsniveau auf 0 V (`LOW`) oder 5 V (`HIGH`) einzustellen ist.

Stell dir vor, dass jeder Ausgangs-Pin eine kleine Steckdose ist, so wie die in den Wänden deiner Wohnung. Die europäischen haben 230 V, amerikanische 110 V, und Arduino arbeitet mit eher bescheidenen 5 V. Hier besteht der Zauber darin, dass Software Hardware steuern kann. Wenn du `digitalWrite(LED, HIGH)` schreibst, stellt sie den Ausgangs-Pin auf 5 V ein, und wenn du eine LED anschließt, leuchtet sie auf. An diesem Punkt deines Codes sorgt eine Anweisung in der Software dafür, dass in der physischen Welt etwas passiert, indem der elektrische Strom zum Pin gesteuert wird. Das Ein- und Ausschalten des Pins lässt uns diese nun in etwas übersetzen, das für das menschliche Auge sichtbarer ist; die LED ist unser *Aktor*.

Auf dem Arduino bedeutet HIGH, dass der Pin auf 5 V eingestellt wird, während LOW bedeutet, dass der Pin auf 0 V eingestellt wird.

Du magst dich wundern, warum wir HIGH und LOW statt ON und OFF verwenden. Es stimmt, dass HIGH oder LOW in der Regel ein beziehungsweise aus entsprechen, aber das hängt davon ab, wie der Pin genutzt wird. So schaltet sich zum Beispiel eine zwischen 5 V und einem Pin angeschlossene LED ein, wenn der Pin LOW ist, und aus, wenn der Pin HIGH ist. In den meisten Fällen kannst du aber annehmen, dass HIGH für EIN und LOW für AUS steht.

```
delay(1000);  // waits for a second
```

Obwohl der Arduino erheblich langsamer ist als dein Laptop, so ist er immer noch sehr schnell. Wenn wir die LED ein- und sofort wieder ausschalten würden, würden deine Augen dies nicht erkennen. Wir müssen die LED eine Weile eingeschaltet lassen, damit wir es sehen, und das erreichen wir, indem wir Arduino auffordern, vor dem nächsten Schritt einen Moment zu warten. Im Grunde lässt `delay()` den Mikrocontroller innehalten und für die von dir als Argument angegebene Anzahl an Millisekunden nichts tun. Millisekunden sind Tausendstelsekunden, somit entsprechen 1000 Millisekunden einer Sekunde. Hier bleibt die LED also 1 Sekunde lang eingeschaltet.

```
digitalWrite(LED, LOW);         // turns the LED off
```

Diese Anweisung schaltet die LED, die wir zuvor eingeschaltet hatten, nun aus.

```
delay(1000);    // waits for a second
```

Hier verzögern wir für eine weitere Sekunde. Die LED wird 1 Sekunde lang ausgeschaltet sein.

```
}
```

Diese schließende geschweifte Klammer kennzeichnet das Ende der `loop()`-Funktion. Wenn Arduino hier ankommt, beginnt es am Anfang von `loop()` von neuem.

Zusammenfassend führt dieses Programm Folgendes aus:

- macht Pin 13 zu einem Ausgang (nur einmal am Anfang)
- begibt sich in eine Schleife
- schaltet die an Pin 13 angeschlossene LED ein
- wartet eine Sekunde
- schaltet die an Pin 13 angeschlossene LED aus
- wartet eine Sekunde
- geht zurück zum Beginn der Schleife

Das war jetzt hoffentlich nicht zu kompliziert. Wenn du nicht alles verstanden hast, lass dich nicht entmutigen. Wie wir zuvor erwähnten, dauert es eine Weile, bis diese Vorgehensweisen für dich als Neueinsteiger einen Sinn ergeben. Du wirst beim Durchgehen weiterer Beispiele mehr zum Programmieren lernen.

Bevor wir uns dem nächsten Abschnitt zuwenden, sollst du ein bisschen mit dem Code spielen. Reduziere zum Beispiel die Dauer der Verzögerung, indem du andere Zahlen für die Ein- und Aus-Impulse eingibst, um unterschiedliche Blinkmuster zu sehen. Du solltest insbesondere sehen, was passiert, wenn du sehr kleine Verzögerungen einstellst, dabei aber unterschiedliche Verzögerungen für Ein und Aus verwendest. Es gibt einen Moment, an dem etwas Merkwürdiges passiert; dieses »etwas« wird sehr nützlich sein, wenn du die *Pulsweitenmodulation* in »Lichtsteuerung mit PWM« auf Seite 60 kennenlernst.

Was wir bauen werden

> *Mich hat schon immer Licht fasziniert und die Möglichkeit, verschiedene Lichtquellen mittels Technik zu steuern. Ich hatte das Glück, an einigen interessanten Projekten arbeiten zu dürfen, die eine Steuerung von Licht und dessen Interaktion mit Menschen beinhalteten. Arduino ist dafür sehr gut geeignet*.
>
> *– Massimo*

In diesem Kapitel, in Kapitel 5 und in Kapitel 6 werden wir uns damit beschäftigen, *interaktive Lampen* zu entwickeln. Dabei nutzen wir Arduino, um die Grundlagen des Aufbaus interaktiver Geräte zu erlernen. Denk aber daran, dass Arduino nicht wirklich versteht bzw. sich nicht darum schert, was du an den Ausgangs-Pins anschließt. Arduino schaltet einfach den Pin auf HIGH oder LOW, der vielleicht eine Lampe steuert oder einen Elektromotor oder den Motor deines Autos.

Im nächsten Abschnitt erläutern wir die Grundlagen der Elektrizität auf eine Art und Weise, die einen Ingenieur langweilen würde, einen neuen Arduino-Programmierer aber nicht direkt abschreckt.

Was ist Elektrizität?

Wenn du zu Hause bereits irgendwelche Klempnerarbeiten durchgeführt hast, dann wirst du keine Probleme haben, Elektronik zu verstehen. Um zu verstehen, wie Elektrizität und Schaltungen funktionieren, ist es am besten, etwas einzusetzen, das sich *Wasseranalogie* nennt. Lass uns ein simples Gerät betrachten, wie den in Abb. 4–3 dargestellten batteriebetriebenen tragbaren Ventilator.

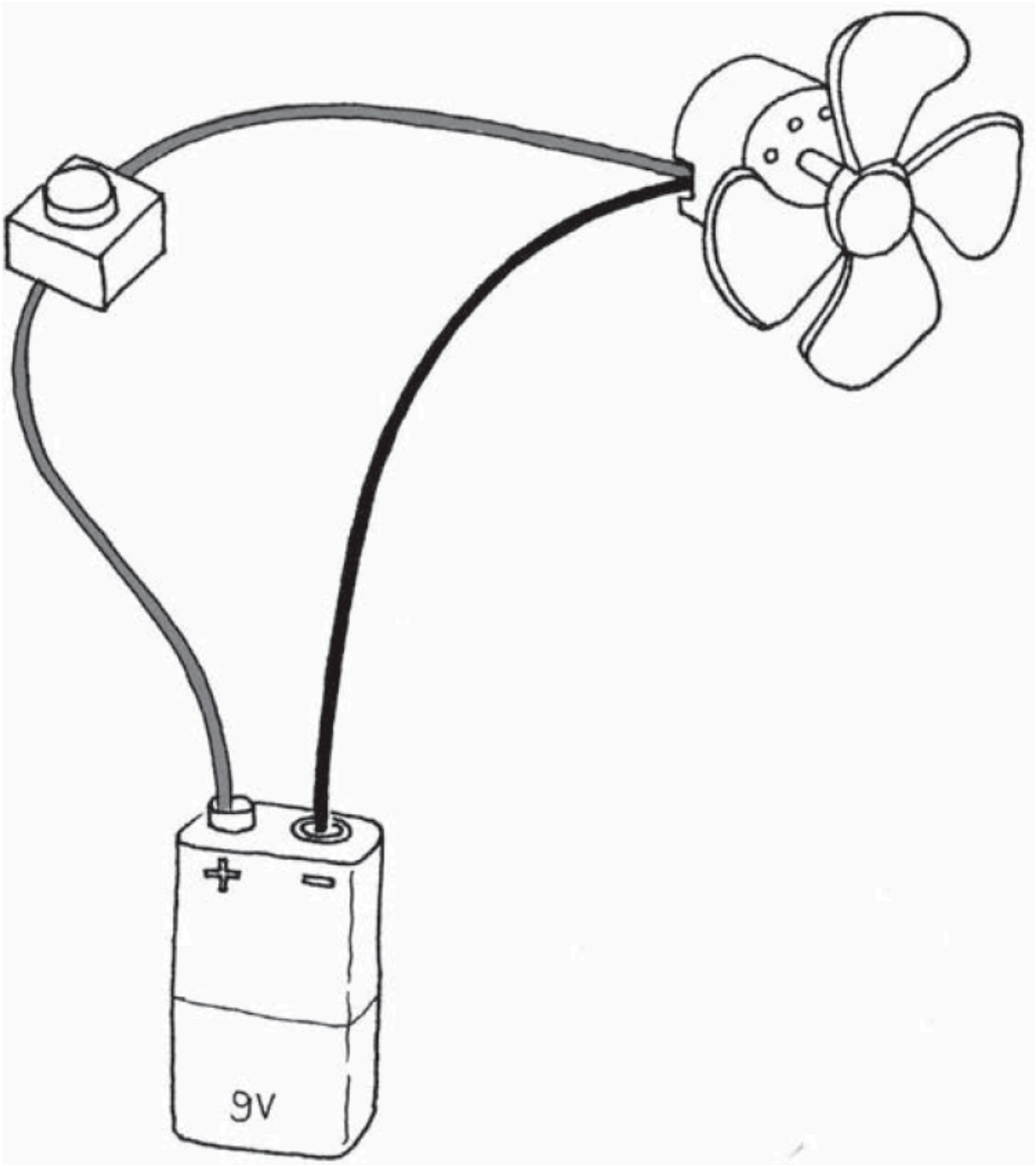


Abb. 4–3 Ein tragbarer Ventilator

Wenn du einen Ventilator auseinandernimmst, wirst du sehen, dass er eine Batterie, ein paar Kabel und einen Elektromotor enthält und dass eins der Kabel zum Motor durch einen Schalter unterbrochen wird. Wenn du den Schalter betätigst, beginnt der Motor sich zu drehen und erzeugt damit den notwendigen Luftstrom, um dir Abkühlung zu verschaffen.

Wie funktioniert das? Nun, stell dir vor, die Batterie ist sowohl ein Wassertank als auch eine Pumpe, der Schalter ist ein Wasserhahn, und der Motor ist eines jener

Räder, die du in Wassermühlen siehst. Wenn du den Wasserhahn öffnest, fließt Wasser von der Pumpe und setzt das Rad in Bewegung.

In diesem einfachen Hydrauliksystem, abgebildet in Abb. 4–4, sind zwei Faktoren wichtig: der Druck des Wassers (dieser wird von der Leistung der Pumpe bestimmt) und die Menge des Wassers, die in den Rohren fließt (das hängt von der Größe der Rohre und dem *Widerstand* ab, den das Rad dem anströmenden Wasser bietet).

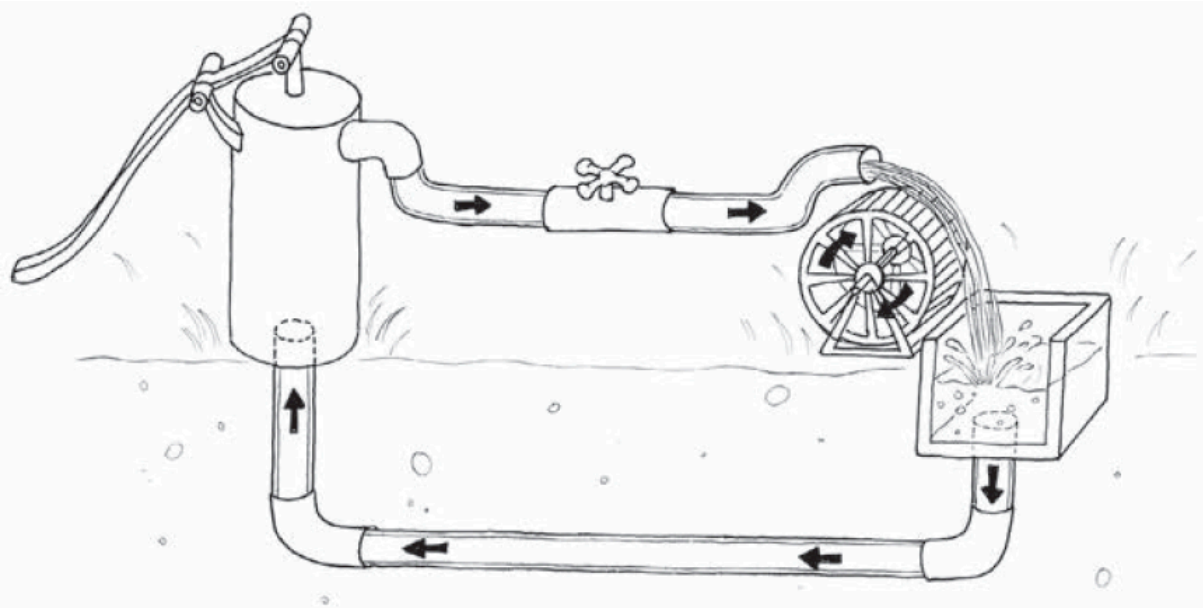


Abb. 4–4 Ein Hydrauliksystem

Du wirst schnell feststellen, dass du, um das Rad schneller drehen zu lassen, die Rohre vergrößern (aber das geht nur bis zu einem gewissen Punkt) und den Druck, den die Pumpe erreichen kann, steigern musst. Eine Vergrößerung der Rohre ermöglicht einen stärkeren Wasserstrom; wenn du sie vergrößerst, reduzierst du effektiv den Widerstand der Rohre gegenüber dem Wasserstrom. Diese Vorgehensweise funktioniert bis zu einem gewissen Punkt, ab dem das Rad sich nicht mehr schneller drehen wird, weil der Wasserdruck nicht stark genug ist. Wenn du diesen Punkt erreichst, benötigst du eine stärkere Pumpe. Diese Methode zum Beschleunigen der Wassermühle geht bis zu dem Punkt, an dem das Rad auseinanderfällt, weil der Wasserstrom zu stark ist und das Rad zerstört. Etwas anderes, das dir auffällt, während sich das Rad dreht, ist, dass sich die Achse ein wenig erhitzt, weil unabhängig von der Befestigung des Rades die Reibung zwischen der Achse und den Löchern, in denen sie montiert ist, Hitze erzeugt. Es ist wichtig zu verstehen, dass in einem System wie diesem nicht die gesamte Energie, die du in das System pumpst, in Bewegung umgesetzt wird – ein Teil davon geht unterwegs verloren und taucht im Allgemeinen als Hitze wieder auf, die aus einigen Teilen des Systems ausstrahlt.

Was sind also die wichtigen Teile des Systems? Der von der Pumpe erzeugte Druck ist eines; der Widerstand, den die Rohre und das Rad dem Wasserstrom entgegensetzen, sowie der Wasserstrom selbst (dieser wird sozusagen durch die Anzahl der Liter Wasser, die in einer Sekunde fließen, repräsentiert) sind die anderen.

Elektrizität funktioniert ein bisschen wie Wasser. Du hast eine Art Pumpe (jegliche Stromquelle, wie eine Batterie oder eine Wandsteckdose), die eine elektrische Ladung (stell sie dir als »Tropfen« von Strom vor) durch Rohre – hier Drähte – drückt. Verschiedene elektrische Geräte können diese Tropfen des Stroms zum Erzeugen von Hitze (die Heizdecke deiner Oma), Licht (die Lampe in deinem Zimmer), Ton (deine Stereoanlage), Bewegung (dein Ventilator) und vielem mehr nutzen.

Wenn du liest, dass die Batteriespannung 9 V beträgt, dann stell dir diese Spannung als den Wasserdruck vor, der potenziell von dieser kleinen »Pumpe« erzeugt werden kann. *Spannung* wird in Volt gemessen, benannt nach Alessandro Volta, dem Erfinder der ersten Batterie.

So wie Wasserdruck ein elektrisches Äquivalent hat, hat die Flussrate von Wasser auch eines. Dieses nennt sich *Strom* oder *Stromstärke* und wird in Ampere gemessen (nach André-Marie Ampère, Pionier des Elektromagnetismus). Kehren wir wieder zum Wasserrad zurück, um die Beziehung zwischen Spannung und Strom zu illustrieren: Mit einer höheren Spannung (Druck) kannst du das Rad schneller drehen, mit einer höheren Flussrate (Strom) kannst du ein größeres Rad drehen.

Abschließend wird der Widerstand gegenüber dem Fließen des Stroms auf jedem von ihm genutzten Weg – du hast es sicher bereits erraten – *Widerstand* genannt, und dieser wird in Ohm gemessen (nach dem deutschen Physiker Georg Ohm).

Herr Ohm war auch für die Formulierung des wichtigsten Gesetzes der Elektrizität verantwortlich –die einzige Formel, die du wirklich im Kopf behalten musst. Er konnte nachweisen, dass in einer Schaltung Spannung, Strom und Widerstand alle zueinander in Beziehung stehen und insbesondere, dass der Widerstand einer Schaltung die Höhe des durch ihn fließenden Stroms bei einer bestimmten Spannungsversorgung bestimmt.

Das ist sehr intuitiv, wenn man darüber nachdenkt. Nimm eine 9-V-Batterie und schließe sie an einer simplen Schaltung an. Beim Messen des Stroms wirst du feststellen: Je mehr Widerstände du dem Schaltkreis hinzufügst, desto weniger Strom fließt durch ihn hindurch. Kehren wir noch einmal zur Analogie mit dem in Rohren fließenden Wasser zurück: Nehmen wir eine Pumpe und installieren ein

Ventil (vergleichbar einem variablen Widerstand in der Elektrizität). Je mehr wir nun das Ventil schließen – Erhöhung des Widerstands gegenüber dem Wasserstrom –, desto weniger Wasser fließt durch die Rohre. Ohm fasste sein Gesetz in diesen Formeln zusammen:

```
R (Widerstand) = U (Spannung) / I (Strom)
```

```
U = R * I
```

```
I = U / R
```

An diesem Gesetz ist wichtig, dass es intuitiv verstanden wird, und deshalb bevorzugen wir die letzte Version (`I = U / R`), da der Strom etwas ist, was sich ergibt, wenn man eine gewisse Spannung (der Druck) auf eine gewisse Schaltung (der Widerstand) ausübt. Die Spannung existiert, egal ob sie genutzt wird oder nicht, und der Widerstand existiert, egal ob ihm Strom zugeführt wird oder nicht, aber der Strom entsteht nur, wenn diese beiden zusammengeführt werden.

Einsatz eines Drucktasters zur Steuerung der LED

Eine LED zum Blinken zu bringen, war einfach, aber wir glauben nicht, dass du bei klarem Verstand bleiben würdest, wenn deine Schreibtischlampe ständig blinkt, während du versuchst, ein Buch zu lesen. Deshalb musst du lernen, sie zu steuern. Im vorangegangenen Beispiel war die LED dein Aktor, und der Arduino hat diesen gesteuert. Was zur Vervollständigung des Bildes fehlt, ist ein Sensor.

In diesem Fall werden wir die einfachste Form eines Sensors verwenden, die es gibt: einen Schalter in Form eines Drucktasters.

Wenn du einen Drucktaster auseinandernehmen würdest, würdest du sehen, dass das eine ganz einfache Vorrichtung ist: zwei Metallteile, die durch eine Feder auseinandergehalten werden, und eine Kunststoffkappe, die beim Drücken die beiden Metallteile miteinander in Kontakt bringt. Wenn die beiden Metallteile getrennt sind, fließt kein Strom im Drucktaster (so ähnlich wie bei einem geschlossenen Wasserhahn); wenn du ihn drückst, stellst du eine Verbindung her.

Alle Schalter sind im Grunde genommen nur das: zwei (oder mehr) Metallteile, die, wenn sie miteinander in Kontakt gebracht werden, einen elektrischen Strom

vom einen zum anderen fließen lassen, bzw. wenn sie getrennt werden, den Strom unterbrechen.

Zum Beobachten des Zustands des Schalters gibt es eine neue Arduino-Anleitung, die du kennenlernen wirst: die `digitalRead()`-Funktion.

`digitalRead()` prüft, ob auf den von dir zwischen diesen Klammern angegebenen Pin eine Spannung angewandt wird, und gibt einen Wert HIGH oder LOW wieder, je nach Resultat. Die anderen Anweisungen, die du bisher verwendet hast, haben keine Informationen wiedergegeben – sie haben nur ausgeführt, was du ihnen aufgetragen hast. Diese Art von Funktion ist aber etwas beschränkt, da sie dich zwingt, an sehr vorhersehbaren Abfolgen von Anweisungen festzuhalten, ohne Einfluss von der Außenwelt. Mit `digitalRead()` kannst du Arduino »eine Frage stellen« und eine Antwort erhalten, die irgendwo im Speicher abgelegt und zum unmittelbaren oder späteren Treffen von Entscheidungen verwendet werden kann.

Bau die in Abb. 4–5 dargestellte Schaltung. Dafür benötigst du ein paar Teile[2] (diese werden auch für die Arbeit an anderen Projekten ganz gelegen kommen):

- lötfreie Steckplatine
- Satz vorgefertigter Steckbrücken
- ein 10-kΩ-Widerstand
- Drucktastschalter

Alternativ zum Kauf vorgefertigter Steckbrücken kannst du auch Massivdraht vom Typ 22 AWG, der auf kleinen Spulen aufgewickelt wird, verwenden und dann mithilfe eines Seitenschneiders und einer Abisolierzange selbst zuschneiden und abisolieren.

GND auf dem Arduino-Board (engl. *GrouND*) steht für *Masse (*oft auch mit *Erde* übersetzt). Der Begriff GND ist historisch bedingt, in unserem Fall bezeichnet er lediglich den Minuspol beim Strom. Wir tendieren dazu, GND als auch Masse synonym zu verwenden. Du kannst es dir wie das im Untergrund verlaufende Rohr in der Wasseranalogie in Abb. 4–4 vorstellen.

In den meisten Schaltungen wird GND oder Masse sehr häufig verwendet. Aus diesem Grund hat dein Arduino-Board drei Pins mit der Kennzeichnung

GND. Sie sind alle miteinander verbunden, und es macht keinen Unterschied, welchen du benutzt.

Der Pin mit der Kennzeichnung *5 V* ist die positive Seite der Stromzufuhr (Plus) und ist stets 5 Volt höher als die Masse.

Beispiel 4–2 zeigt den Code, den wir zum Steuern der LED mit unserem Drucktastschalter verwenden werden.

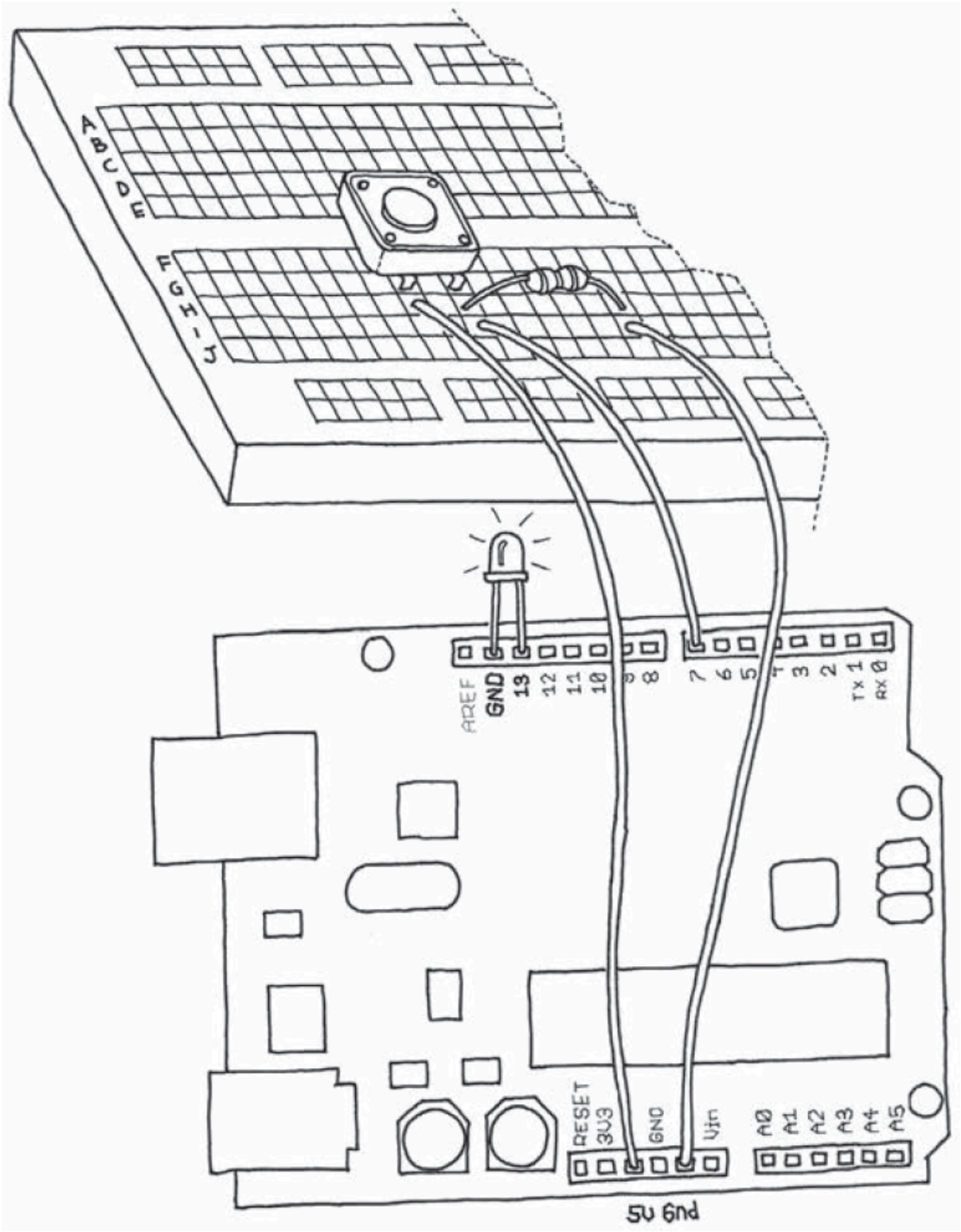


Abb. 4–5 Anschluss eines Drucktasters

Beispiel 4–2 LED bei gedrücktem Schalter einschalten

```
// Turn on LED while the button is pressed
```

```
const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;          // val will be used to store the state
                      // of the input pin
void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}
void loop(){
  val = digitalRead(BUTTON); // read input value and store it
  // check whether the input is HIGH (button pressed)
  if (val == HIGH) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
```

```
}
```

Wähle in Arduino *File* ⟶ *New* (wenn du einen anderen Sketch geöffnet hast, möchtest du ihn vielleicht erst speichern). Wenn Arduino um Eingabe eines Namens für deinen neuen Sketch-Ordner bittet, gib *PushButtonControl* ein. Tippe den Code aus Beispiel 4–2 in Arduino ein (oder lade ihn von der Katalogseite (*https://makezine.com/go/arduino-4e-github/*) der englischen Originalausgabe dieses Buches herunter und lege ihn in der Arduino-IDE ab). Wenn alles korrekt ist, leuchtet die LED auf, wenn du den Taster drückst.

Wie funktioniert das?

Wir haben mit diesem Beispielprogramm zwei neue Konzepte eingeführt: Funktionen, die das Ergebnis ihrer Arbeit wiedergeben, und die `if`-Anweisung.

Die `if`-Anweisung ist möglicherweise die wichtigste Anweisung in einer Programmiersprache, da sie es dem Computer (du erinnerst dich: Der Arduino ist ein kleiner Computer) ermöglicht, Entscheidungen zu treffen. Nach dem `if`-Stichwort musst du in den Klammern eine »Frage« eingeben, und wenn die »Antwort« bzw. das Ergebnis wahr ist, wird der erste Codeblock ausgeführt; andernfalls wird der Codeblock nach `else` ausgeführt.

Beachte, dass das Symbol == etwas völlig anderes bedeutet als das Symbol =. Ersteres wird verwendet, wenn zwei Datensätze verglichen werden, und es gibt `wahr` oder `falsch` wieder; letzteres weist einer Konstanten oder Variablen einen Wert zu. Achte darauf, dass du das richtige Symbol verwendest, denn es passiert schnell, nur = einzugeben, und in dem Fall wird dein Programm nie funktionieren. Wir wissen das, denn auch nach jahrelangem Programmieren machen wir immer noch diesen Fehler.

Du solltest wissen, dass der Schalter nicht direkt mit der LED verbunden ist. Dein Arduino-Sketch inspiziert den Schalter und trifft dann die Entscheidung, ob er die LED ein- oder ausschaltet. Die Verbindung zwischen dem Schalter und der LED erfolgt tatsächlich in deinem Sketch.

Den Finger so lange auf dem Taster zu halten, wie man Licht benötigt, ist unpraktisch. Obwohl es einem klarmachen würde, wie viel Energie man verschwendet, falls man eine Lampe brennen lässt, wenn man aus dem Raum geht, müssen wir einen Weg finden, den Taster »festzuhalten«.

Eine Schaltung, tausend Verhaltensweisen

Der große Vorteil programmierbarer Elektronik gegenüber klassischer Elektronik wird nun offensichtlich: Ich werde dir zeigen, wie man unter Verwendung derselben elektronischen Schaltung aus dem vorhergehenden Abschnitt viele verschiedene »Verhaltensweisen« implementieren kann, indem man einfach die Software ändert.

Wie ich bereits erwähnte, ist es nicht sonderlich praktisch, deinen Finger auf dem Taster zu halten, damit das Licht brennt. Deshalb musst du eine Art von »Gedächtnis« in Form eines Softwaremechanismus implementieren, das sich merkt, dass du den Knopf gedrückt hast, und das Licht brennen lässt, auch nachdem du den Taster losgelassen hast.

Dafür verwenden wir eine sogenannte *Variable*. (Du hast bereits eine verwendet, aber wir haben es nicht erklärt.) Eine Variable ist ein Ort im Arduino-Speicher, an der du Daten ablegen kannst. Sieh es wie einen dieser selbstklebenden Notizzettel an, der dich an etwas erinnern soll, wie zum Beispiel eine Telefonnummer: Du nimmst einen, notierst darauf »Luisa 0255 51212« und klebst ihn an deinen Bildschirm oder Kühlschrank. In der Arduino-Sprache ist das genauso simpel: Du entscheidest, welche Art von Daten du speichern möchtest (beispielsweise eine Zahl oder einen Text), gibst dem einen Namen, und dann kannst du, wenn du willst, die Daten speichern oder abrufen. Zum Beispiel:

```
int val = 0;
```

`int` bedeutet, dass deine Variable eine Ganzzahl speichert, `val` ist der Name der Variablen, und `= 0` weist einen Anfangswert von Null zu.

Eine Variable kann, wie der Name andeutet, irgendwo in deinem Code modifiziert werden, sodass du später in deinem Programm schreiben könntest:

```
val = 112;
```

was deiner Variablen einen neuen Wert, 112, zuweist.

Hast du bemerkt, dass in Arduino jede Anweisung mit einem Semikolon endet? Dies macht man, damit der Compiler (der Teil von Arduino, der

deinen Sketch in ein Programm umwandelt, das der Mikrocontroller ausführen kann) weiß, dass deine Anweisung abgeschlossen ist, und eine neue beginnt. Wenn du dort, wo ein Semikolon erforderlich ist, dieses vergisst, ergibt dein Sketch für den Compiler keinen Sinn.

Im folgenden Programm speichert die Variable `val` das Ergebnis von `digitalRead()`; was auch immer Arduino vom Eingang empfängt, landet in der Variablen und bleibt dort, bis eine andere Codezeile es ändert. Beachte, dass Variablen einen Speichertyp namens *RAM* verwenden. Der ist ziemlich schnell, aber wenn du dein Board ausschaltest, gehen alle Daten im RAM verloren (was bedeutet, dass jede Variable beim erneuten Einschalten des Boards auf den ursprünglichen Wert zurückgesetzt wird). Deine Programme selbst werden im Flash Memory gespeichert – das ist derselbe Typ wie in deinem Handy, das dort Telefonnummern speichert –, der seinen Inhalt auch nach dem Ausschalten behält.

Lass uns nun eine weitere Variable verwenden, die daran erinnert, ob die LED nach dem Loslassen des Tasters ein- oder ausgeschaltet sein soll. Beispiel 4–3 zeigt einen ersten Versuch.

Beispiel 4–3 LED einschalten, wenn der Taster gedrückt ist, und nach dem Loslassen brennen lassen

```
const int LED = 13;   // the pin for the LED

const int BUTTON = 7; // the input pin where the

                      // pushbutton is connected

int val = 0;    // val will be used to store the state

                // of the input pin

int state = 0;  // 0 = LED off while 1 = LED on

void setup() {

  pinMode(LED, OUTPUT);     // tell Arduino LED is an output
```

```
  pinMode(BUTTON, INPUT);   // and BUTTON is an input
}
void loop() {
  val = digitalRead(BUTTON); // read input value and store it
  // check if the input is HIGH (button pressed)
  // and change the state
  if (val == HIGH) {
    state = 1 - state;
  }
  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Teste nun diesen Code. Du wirst feststellen, dass er funktioniert … einigermaßen. Aber das Licht wechselt so schnell, dass du es mit einem Tastendruck nicht zuverlässig ein- oder ausschalten kannst.

Schauen wir uns die interessanten Teile des Codes an: `state` ist eine Variable, die entweder 0 oder 1 speichert, um sich daran zu erinnern, ob die LED ein- oder ausgeschaltet ist. Nachdem der Taster losgelassen wird, initialisieren wir sie zu 0 (LED aus).

Später lesen wir den aktuellen `state` des Knopfes ab, und wenn er gedrückt ist (`val = = HIGH`), ändern wir den `state` von 0 auf 1 oder umgekehrt. Wir tun dies mithilfe eines kleinen Tricks, da state nur 1 oder 0 sein kann. Der von mir verwendete Trick beinhaltet einen kleinen mathematischen Ausdruck, basierend auf der Vorstellung, dass 1 – 0 gleich 1 ist und 1 – 1 gleich 0:

```
state = 1 - state;
```

Die Zeile mag mathematisch nicht viel Sinn ergeben, beim Programmieren aber schon. Das Symbol = bedeutet »Weise das Ergebnis dessen, was nach mir ist, dem Variablennamen vor mir zu« – in diesem Fall wird dem neuen Wert von `state` der Wert von 1 minus dem alten Wert von `state` zugewiesen.

Später im Programm kannst du erkennen, dass wir `state` verwenden, um herauszufinden, ob die LED ein- oder ausgeschaltet sein soll. Wie schon erwähnt, führt das zu etwas unzuverlässigen Ergebnissen.

Dies liegt an der Art und Weise, wie wir den Taster ablesen. Arduino ist wirklich schnell: Es führt seine eigenen internen Anweisungen mit einer Rate von 16 Millionen pro Sekunde aus – es kann durchaus ein paar Millionen Zeilen Code pro Sekunde ausführen. Das bedeutet, dass Arduino die Position des Tasters, den du gedrückt hältst, mehrere tausend Mal abliest und dementsprechend `state` ändert. Somit führt es zu unvorhersehbaren Ergebnissen: Die Lampe kann ausgeschaltet sein, wenn du sie eingeschaltet haben möchtest, oder umgekehrt. So wie eine kaputte Uhr zweimal am Tag die richtige Uhrzeit anzeigt, könnte das Programm hin und wieder das korrekte Verhalten zeigen, aber meistens wird es falsch sein.

Wie kann man das beheben? Nun, du musst den exakten Moment erfassen, in dem der Taster gedrückt wird – genau in dem Moment musst du `state` ändern. Unser bevorzugter Weg ist, den Wert von `val` zu speichern, bevor wir einen neuen ablesen; so kann man die aktuelle Position des Tasters mit der vorherigen vergleichen und `state` nur dann ändern, wenn sich der Knopf von LOW in HIGH ändert.

Beispiel 4–4 enthält den entsprechenden Code:

Beispiel 4–4 Neue und verbesserte Formel für den Tastendruck!

```
const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;     // val will be used to store the state
                 // of the input pin
int old_val = 0; // this variable stores the previous
                 // value of "val"
int state = 0;   // 0 = LED off and 1 = LED on
void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}
void loop(){
  val = digitalRead(BUTTON); // read input value and store it
                             // yum, fresh
 // check if there was a transition
 if ((val == HIGH) && (old_val == LOW)){
```

```
    state = 1 - state;
  }
  old_val = val; // val is now old, let's store it
  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

In dieser if-Anweisung wird dir etwas Neues aufgefallen sein: Hier gibt es zwei Vergleiche, getrennt durch ein neues Symbol: &&. Dieses Symbol führt die logische AND-Operation durch, das heißt, dass die Verbundanweisung nur dann wahr ist, wenn beide Einzelanweisungen wahr sind.

Jetzt teste diesen Code: Du hast es fast geschafft!

Du hast vielleicht bemerkt, dass diese Vorgehensweise aufgrund eines anderen Problems mit mechanischen Schaltern nicht ganz perfekt ist.

Wie wir bereits erläuterten, sind Drucktastschalter nur zwei Metallteile, die durch eine Feder voneinander ferngehalten werden und erst miteinander in Kontakt kommen, wenn du den Taster drückst. Das klingt möglicherweise so, als wäre der Schalter vollständig eingeschaltet, wenn du den Taster drückst, aber tatsächlich prallen die beiden Metallteile voneinander ab, genau wie ein Ball auf dem Boden aufspringt.

Obwohl das Abprallen nur über eine ganz kurze Distanz und innerhalb eines Bruchteils einer Sekunde erfolgt, lässt es den Schalter mehrere Male zwischen

Aus und Ein wechseln, bevor das Abprallen stoppt, und Arduino ist schnell genug, um dies zu erfassen.

Während der Drucktastschalter hüpft, sieht der Arduino eine sehr schnelle Abfolge von Ein- und Aus-Signalen. Es sind viele Techniken zum *Entprellen* entwickelt worden, aber in diesem einfachen Code reicht es normalerweise aus, eine Verzögerung von 10 bis 50 Millisekunden hinzuzufügen, wenn der Code einen Wechsel erkennt. Mit anderen Worten: Du wartest einfach einen Moment, bis das Hüpfen aufhört.

Beispiel 4–5 zeigt den abschließenden Code.

Beispiel 4–5 Eine neue und verbesserte Formel zum Drücken des Tasters – mit einfacher Entprellung!

```
const int LED = 13;     // the pin for the LED

const int BUTTON = 7;   // the input pin where the

                        // pushbutton is connected

int val = 0;      // val will be used to store the state

                  // of the input pin

int old_val = 0; // this variable stores the previous

                  // value of "val"

int state = 0;    // 0 = LED off and 1 = LED on

void setup() {

  pinMode(LED, OUTPUT);      // tell Arduino LED is an output

  pinMode(BUTTON, INPUT);    // and BUTTON is an input

}
```

```
void loop(){

 val = digitalRead(BUTTON); // read input value and store it

                            // yum, fresh

 // check if there was a transition

 if ((val == HIGH) && (old_val == LOW)){

   state = 1 - state;

   delay(10);

 }

 old_val = val; // val is now old, let's store it

 if (state == 1) {

   digitalWrite(LED, HIGH); // turn LED ON

 } else {

   digitalWrite(LED, LOW);

 }

}
```

Ein Leser, Tami (Masaaki) Takamiya, meldete sich mit ein wenig zusätzlichem Code, der zu einer besseren Entprellung führen könnte:

```
if ((val == LOW) && (old_val == HIGH)) {

    delay(10);

}
```

Index

Symbole

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z