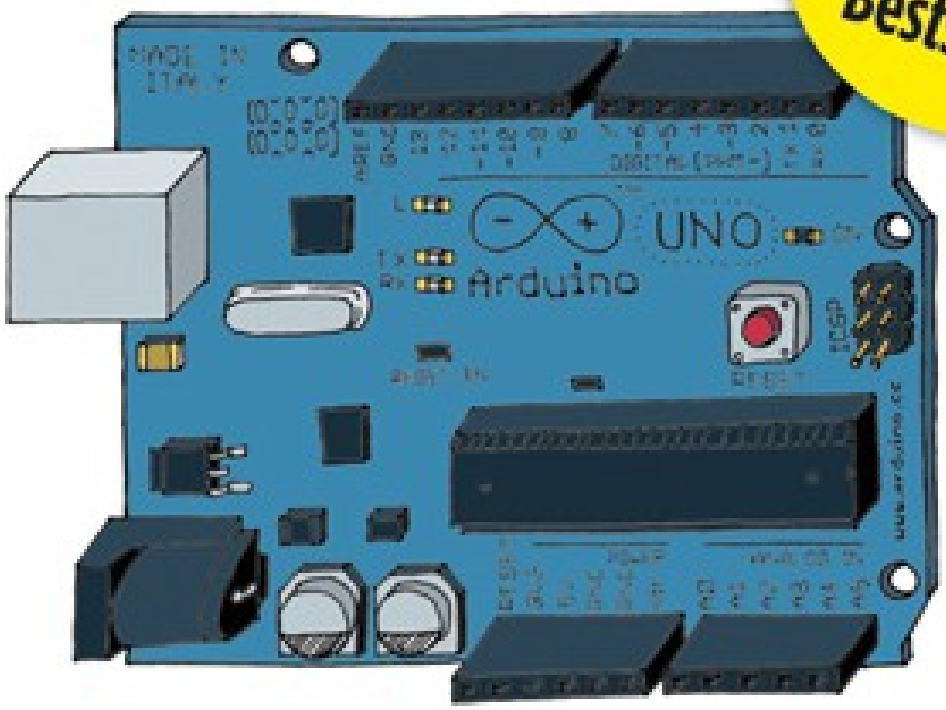
O'REILLY®

3. Auflage

Arduino für Einsteiger

Aktualisierter und erweiterter Bestseller



Die OPEN-SOURCE-PLATTFORM für MAKER

Massimo Banzi, Mitbegründer von Arduino, & Michael Shiloh

Übersetzung von Tanja Feder

Massimo Banzi / Michael Shiloh, Arduino für Einsteiger, O´Reilly, ISBN 978-3-95875-055-5

Inhaltsverzeichnis

4/Die wirklich ersten Schritte mit Arduino

Als Nächstes wirst du erfahren, wie du ein interaktives Gerät bauen und programmieren kannst.

Der Aufbau eines interaktiven Gerätes

Alle Objekte, die wir bauen werden, basieren auf einem simplen Muster, das wir *Interactive Device* nennen. Hierbei handelt es sich um elektronische Schaltungen, die mithilfe von Sensoren (elektronische Komponenten, die Messwerte aus der realen Welt in elektrische Signale umwandeln) die Umgebung erfassen. Das Gerät verarbeitet die Daten, die von den Sensoren geliefert werden, mittels eines Verhaltens, das in der Software beschrieben ist. Das Gerät ist dann in der Lage, mithilfe von *Aktoren* (elektronische Komponenten, die ein elektrisches Signal in physikalisches Verhalten umwandeln können) mit der Welt zu interagieren.

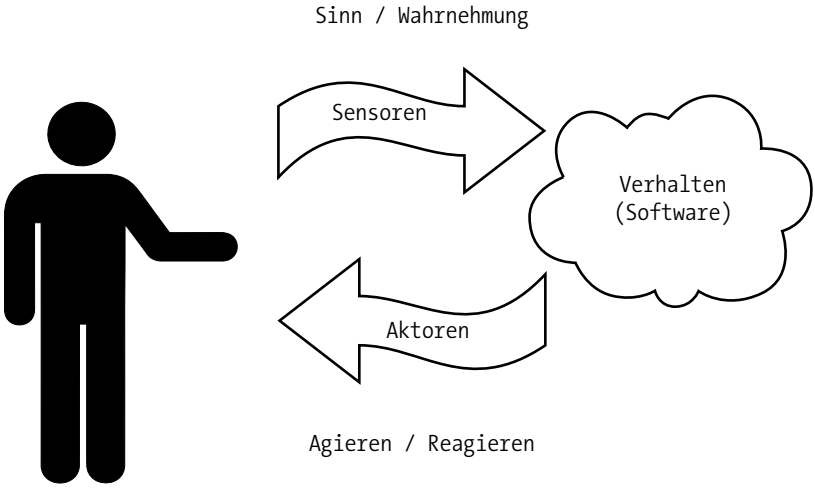


Abbildung 4-1: *Das interaktive Gerät*

Sensoren und Aktoren

Sensoren und Aktoren sind elektronische Komponenten, mit deren Hilfe eine elektronische Komponente mit der Umwelt interagieren kann.

Da es sich bei einem Mikrocontroller um einen sehr einfachen Computer handelt, kann er nur elektrische Signale verarbeiten (ähnlich wie bei elektrischen Impulsen, die zwischen den Neuronen unseres Gehirns übertragen werden). Um Licht, Temperatur oder andere physikalische Größen erfassen zu können, müssen sie in Elektrizität umgewandelt werden. In unserem Körper wandelt das Auge Licht in Signale um, die über Nerven ans Gehirn weitergeleitet werden. In der Elektronik können wird dazu ein spezielles Bauteil verwenden, nämlich einen *lichtabhängigen Widerstand* oder *LDR*, auch als *Fotowiderstand* bekannt, der die auftreffende Lichtmenge messen und als Signal wiedergeben kann, das der Mikrocontroller versteht.

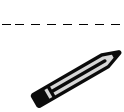
Wenn die Sensoren ausgelesen wurden, verfügt das Gerät über die Informationen, die erforderlich sind, um zu entscheiden, wie es reagieren soll. Der Prozess der Entscheidungsfindung wird vom Mikrocontroller abgewickelt, und die Reaktion erfolgt über die Aktoren. In unseren Körper beispielsweise erhalten die Muskeln elektrische Signale vom Gehirn, die sie dann in Bewegung umsetzen. Im Bereich der Elektronik könnten diese Funktionen z.B. durch Licht oder einen elektrischen Motor ausgeführt werden.

In den folgenden Abschnitten wirst du erfahren, wie unterschiedliche Typen von Sensoren ausgelesen und unterschiedliche Arten von Aktoren gesteuert werden.

Eine LED zum Blinken bringen

Der Sketch, mit dem eine LED zum Blinken gebracht wird, ist der erste Sketch, den du ausführen solltest, um zu testen, ob dein Board einwandfrei arbeitet und richtig konfiguriert ist. Dies ist üblicherweise auch die erste Übung für das Programmieren eines Mikrocontrollers. Eine *Leuchtdiode* (LED) ist eine kleine elektronische Komponente, die einer kleinen Glühbirne ähnelt, jedoch effektiver ist und eine geringere Betriebsspannung benötigt.

Dein Arduino-Board wird mit einer vorinstallierten LED geliefert, die mit einem *L* auf dem Board gekennzeichnet ist. Diese vorinstallierte LED ist mit dem Pin Nummer 13 verbunden. Merke dir die Nummer, denn wir werden sie später noch brauchen. Du kannst auch deine eigene LED hinzufügen. Schließe sie so an, wie es in Abbildung 4-2 dargestellt ist. Achte darauf, sie in das Verbindungsloch mit der Nummer 13 zu stecken.



Wenn die LED über längere Zeit leuchten soll, solltest du einen Widerstand verwenden, wie in Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 beschrieben wird.

K kennzeichnet die Kathode (negativ), bei der es sich um den kürzeren Anschlussdraht handelt, und A kennzeichnet die Anode (positiv), die den längeren Anschlussdraht aufweist.

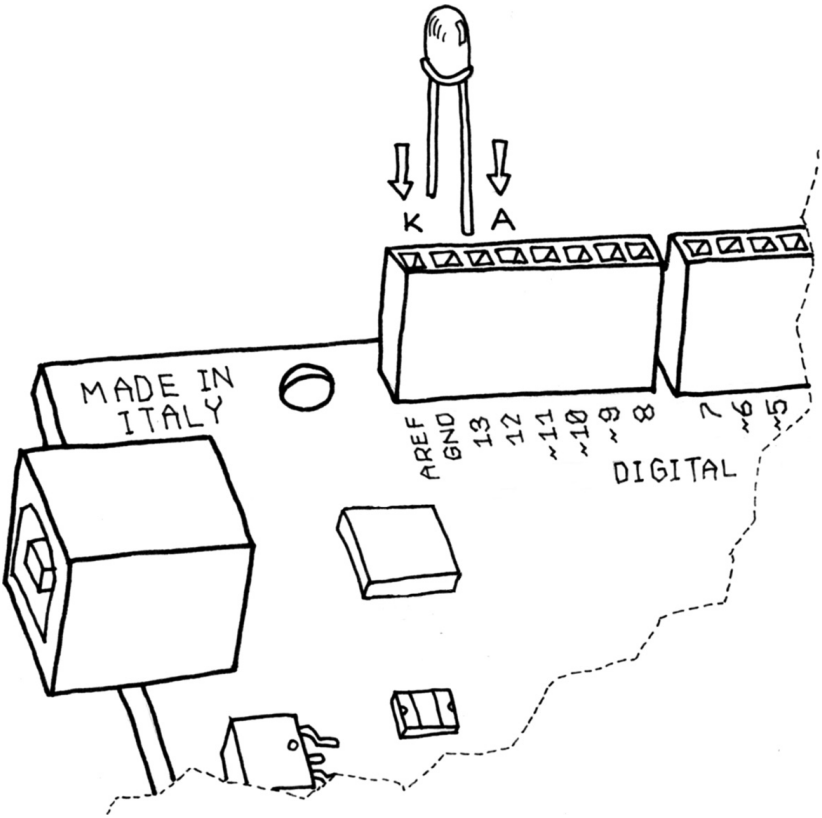


Abbildung 4-2: *Anschließen einer LED an das Arduino-Board*

Wenn die LED angeschlossen ist, muss du dem Arduino mitteilen, was zu tun ist. Dies erfolgt mittels Code, einer Liste von Anweisungen, die du wiederum dem Mikrocontroller übermittelst und mit der du ihn dazu bringst, das zu tun, was du möchtest. (Die Begriffe *Code*, *Programm* und *Sketch* beziehen sich alle auf dieselbe Liste mit Anweisungen).

Öffne auf deinem Computer die Arduino-IDE (beim Mac befindet sie sich im Ordner Applications, unter Windows findest du einen entsprechenden Shortcut entweder auf dem Desktop oder im Startmenü). Starte die IDE mit einem Doppelklick auf das entsprechende Symbol. Wähle File → New aus. Du wirst nun aufgefordert, einen Ordnernamen für den Sketch anzugeben. Hier wird der Sketch dann gespeichert. Nenne ihn *Blinking_LED* und klicke auf OK. Gib dann den folgenden Sketch (Beispiel 4-1) in den Sketch-Editor von Arduino (das Hauptfenster der Arduino-IDE) ein. Du kannst ihn auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Eine dritte Möglichkeit besteht darin, den Sketch einfach durch Klicken auf File → Examples → 01.Basics → Blink zu laden; den größten Lerneffekt erzielst du aber, wenn du ihn selbst eingibst.

Er sollte wie in Abbildung 4-3 aussehen.

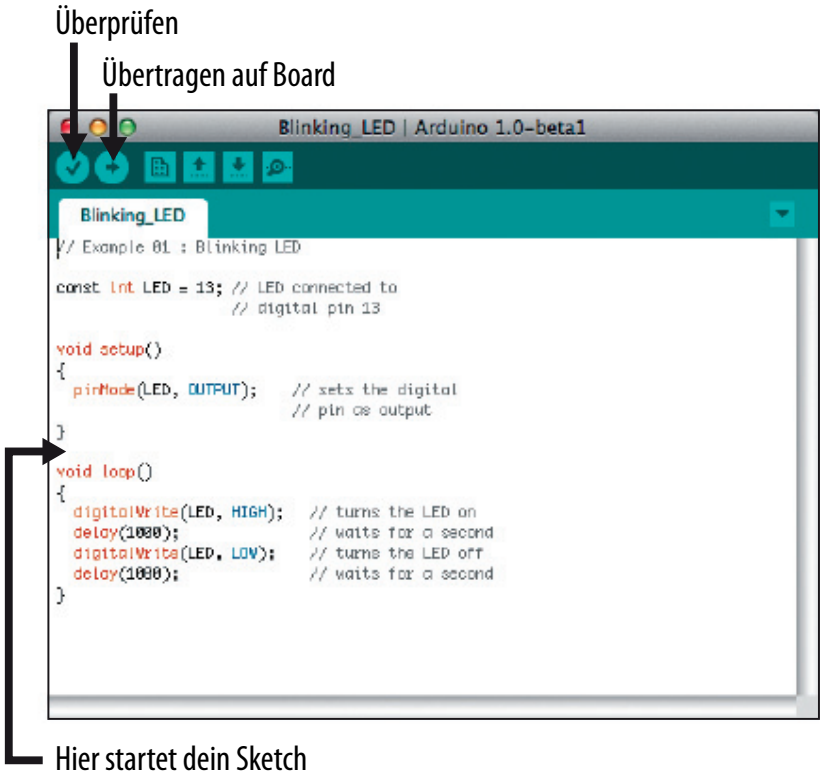


Abbildung 4-3: *Die Arduino-IDE mit dem ersten geladenen Sketch*

Beispiel 4-1: *Eine LED zum Blinken bringen*

```
// 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
}
```

Nun, da sich der Code in deiner IDE befindet, musst du ihn auf Fehler überprüfen. Klicke auf die Schaltfläche Überprüfen (Abbildung 4-3; wenn alles korrekt ist, wird dir die Nachricht *Done compiling* am unteren Rand der Arduino-IDE angezeigt. Diese Nachricht bedeutet, dass die Arduino-IDE deinen Sketch in ein ausführbares Programm übersetzt hat, das auf deinem Board ausgeführt werden kann, ähnlich wie das bei einer .exe-Datei unter Windows oder bei einer .app-Datei unter Mac der Fall ist.

Wenn du eine Fehlermeldung erhältst, hast du sehr wahrscheinlich einen Fehler bei der Eingabe des Codes gemacht. Schau dir jede Zeile genau an und überprüfe jedes einzelne Zeichen, besonders solche wie runde Klammern, eckige Klammern, Semikolons und Kommata. Vergewissere dich, dass du Groß- und Kleinschreibung sorgfältig übernommen hast und dass du den Buchstaben O und die Zahl 0 korrekt verwendet hast.

Wenn der Code fehlerfrei ist, kannst du deinen Sketch auf dein Board laden: Klicke auf die Schaltfläche Übertragen auf Board (siehe Abbildung 4-3). Dadurch wird die IDE angewiesen, den Upload-Prozess zu starten, wodurch das Arduino-Board zurückgesetzt und somit alle laufenden Prozesse beendet werden. Dann wartet das Board auf Instruktionen, die vom USB-Port kommen. Anschließend sendet die Arduino-IDE den Sketch zum Arduino-Board, das ihn wiederum in seinem permanenten Speicher speichert. Sobald die IDE den gesamten Sketch übertragen hat, führt das Arduino-Board den Sketch aus.

Dies geschieht recht schnell. Wenn du die Schaltfläche der Arduino-IDE im Auge behältst, wirst du einige Nachrichten im schwarzen Bereich am unteren Rand des Fenster sehen, und genau über diesem Bereich die Meldungen

Compiling, dann Uploading und schließlich Done uploading, mit denen du darüber informiert wirst, dass der Prozess erfolgreich abgeschlossen wurde.

Es sind zwei LEDs mit den Bezeichnungen RX und TX auf dem Arduino-Board vorhanden. Diese flackern jedes Mal auf, wenn ein Byte vom Board geschickt oder empfangen wird. Während des Upload-Prozesses flackern sie kontinuierlich. Auch dies geschieht recht schnell, und wenn du nicht zur richtigen Zeit auf dein Arduino-Board schaust, wirst du diesen Vorgang verpassen.

Falls du kein Flackern der LEDs erkennen kannst oder anstelle der Nachricht Done Uploading eine Fehlermeldung erhältst, besteht ein Kommunikationsproblem zwischen deinem Computer und dem Arduino. Vergewissere dich, dass du den richtigen seriellen Anschluss (siehe Kapitel 3, *Die Arduino-Plattform*, auf Seite 15) unter Tools→Serial Port ausgewählt hast. Überprüfe außerdem, ob unter Tools→Board das richtige Arduino-Modell ausgewählt wurde.

Wenn weiterhin Probleme bestehen, schau dir das Kapitel Kapitel 9, *Troubleshooting*, auf Seite 171 an.

Sobald der Code auf dein Arduino-Board übertragen wurde, verbleibt er dort, bis du ihn mit einem anderen Sketch überschreibst. Der Code bleibt gespeichert, wenn du beim Board einen Reset durchführst oder es ausschaltest, das ist ähnlich wie bei den Daten auf deiner Computer-Festplatte.

Wenn der Sketch korrekt geladen wurde, wird die LED L für eine Sekunde aufleuchten und dann für eine Sekunde dunkel bleiben. Wenn du eine separate LED installiert hast, wie vorher in Abbildung 4-2 zu sehen ist, wird auch diese LED blinken. Das, was du geschrieben hast, ist ein *Computer-Programm* oder auch *Sketch*, wie ein Arduino-Programm genannt wird. Wie schon erwähnt, handelt es sich beim Arduino um einen kleinen Computer, der sich nach Bedarf programmieren lässt. Um eine Serie von Anweisungen in die Arduino-IDE einzugeben, wird eine Programmiersprache verwendet, die die Anweisungen so umwandelt, dass sie vom Arduino-Board ausgeführt werden können.

Als Nächstes möchten wir ein Verständnis für den Code vermitteln. Zunächst ist zu erwähnen, dass Arduino den Code der Reihe nach von oben nach unten ausführt. Die erste Zeile oben ist also die, die zuerst gelesen wird. Dann wird der Prozess nach unten fortgesetzt. Dies erinnert ein wenig an die Statusanzeige bei einen Video-Player, z. B. Quick Time Player oder Windows Media Player, bei dem die Statusanzeige allerdings nicht von oben nach unten, sondern von links nach rechts verläuft, um anzuzeigen, wo im Film du dich gerade befindest.

Reich mir den Parmesan

Achte auf die geschweiften Klammern, die dazu dienen, Code-Zeilen zusammenzufassen. Sie sind insbesondere dann sehr nützlich, wenn du eine Gruppe Anweisungen mit einem Namen versehen möchtest. Mit der Aufforderung "Bitte reich' mir den Parmesankäse" beim Abendessen wird beispielsweise eine ganze Reihe von Aktionen angestoßen, die in diesem kleinen Satz zusammengefasst sind. Weil wir Menschen sind, erfassen wir das ganz selbstverständlich, beim Arduino hingegen müssen alle einzelnen kleinen Aktionen ausformuliert werden, weil die Plattform nicht so leistungsfähig wie unser Gehirn ist. Um also mehrere Anweisungen in einer Gruppe zusammenzufassen, platzierst du ein { vor dem Code-Block und ein } hinter dem Code.

Du siehst, dass in unserem Beispiel zwei Blöcke auf diese Weise definiert wurden. Vor jedem dieser Blöcke sind ein paar merkwürdige Wörter angeführt:

```
void setup()
```

Mit dieser Zeile wird dem Code-Block ein Name zugewiesen. Wenn du eine Liste mit Anweisungen schreiben würdest, die Arduino beibringen, dir den Parmesankäse zu reichen, würdest du `void passTheParmesan()` am Anfang des Blocks schreiben – und dieser Block würde zu einer Anweisung, die du von jeder beliebigen Stelle im Arduino-Code aus aufrufen könntest. Solche Blöcke werden als Funktionen bezeichnet. Wenn du also aus diesem Code-Block eine Funktion erstellt hast, kannst du anschließend `passTheParmesan()` an jeder beliebigen Stelle im Sketch schreiben, und der Arduino wird zur Funktion `passTheParmesan()` springen, die betreffenden Anweisungen ausführen, dann zurückspringen und an der Stelle fortfahren, an dem der Code vor den Anweisungen verlassen wurde.

Dies zeigt einen wichtigen Aspekt eines jeden Arduino-Programms. Beim Arduino können nicht mehrere Prozesse gleichzeitig ablaufen, das bedeutet, es wird zu jedem Zeitpunkt immer nur eine Anweisung ausgeführt. Da der Arduino dein Programm Zeile für Zeile ausführt, wird zum gegebenen Zeitpunkt immer nur diese eine Zeile *ausgeführt*. Nach einem Sprung zu einer Funktion wird diese Funktion Zeile für Zeile ausgeführt, bevor dann eine Rückkehr zum Ausgangspunkt erfolgt. Der Arduino kann nicht zwei Sätze an Anweisungen gleichzeitig ausführen.

Arduino ist nichts für Zauderer

Arduino erwartet immer, dass du zwei Funktionen erstellt hast: Die eine heißt `setup()` und die andere `loop()`.

`setup()` ist die Funktion, in der all der Code untergebracht wird, der zu Beginn des Programms ausgeführt werden soll, und `loop()` enthält das Kernstück des Programms, das kontinuierlich immer wieder ausgeführt werden soll. Dies liegt daran, dass der Arduino sich nicht wie ein normaler Computer verhält: Es können nicht mehrere Programme gleichzeitig ausgeführt werden, und es ist auch nicht möglich, ein Programm abzubrechen. Wenn das Board an eine Stromversorgung angeschlossen ist, wird der Code ausgeführt. Wenn du die Ausführung beenden möchtest, musst du dazu einfach das Board von der Stromversorgung trennen.

Wirkliche Tüftler schreiben Kommentare

Jeglicher Text, der mit `//` beginnt, wird vom Arduino ignoriert. Diese Zeilen sind Kommentare, die du für dich selbst im Programm hinterlässt, um dich daran zu erinnern, was du mit dem Code bezweckt hast, oder die du für andere schreibst, damit sie den Code verstehen.

Es ist absolut üblich (und wir wissen das, weil wir es kontinuierlich tun), einen Code-Abschnitt zu schreiben, ihn auf das Board zu laden und sich dann zu sagen: Okay, diesen Kram werde ich nie wieder anfassen! – nur um sechs Monate später festzustellen, dass der Code aktualisiert werden muss oder noch ein Fehler zu beheben ist. Du wirst dir den Code anzeigen lassen, und wenn du dann keine entsprechenden Kommentare in deinem ursprünglichen Programm eingefügt hast, wirst du sehr schnell denken: "Oh Mann, was für ein Chaos! Wo fange ich denn da bloß an?" Wenn wir in diesem Buch weiter voranschreiten, wirst du noch einige Tricks kennenlernen, wie du dein Programm lesbarer und einfacher im Hinblick auf die Wartung gestaltest.

Wenn du später deine eigene Sketche schreiben wirst und glaubst, dass sie auch für Tüftler interessant sind, die nicht Deutsch verstehen, dann solltest du auf jeden Fall dich darum bemühen, die Kommentare in englischer Sprache zu verfassen. Du erreichst dann viel mehr Leute mit deinem Sketch.

Der Code – Schritt für Schritt

> Womöglich wird dir diese Art von Erläuterung ein wenig überflüssig vorkommen, ähnlich wie in meiner Schulzeit, als ich Dantes *Göttliche Komödie* lesen musste (jeder italienische Schüler muss sie durcharbeiten, genauso wie ein anderes Buch mit dem Titel *Die Brautleute* – oh, was für ein Albtraum!). Für jede Textzeile gab es hundert Zeilen Kommentar. Wenn du beginnst, eigene Programme zu schreiben, sind solche Erläuterungen wesentlich nützlicher.
>
> Massimo

```
// Example 01 : Blinking LED
```

Ein Kommentar ist eine hilfreiche Möglichkeit, kleine Hinweise einzufügen. Der vorangestellte Titelkommentar erinnert uns daran, dass dieses Programm, Beispiel 4-1, eine LED zum Blinken bringt.

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

`const int` bedeutet, dass es sich bei *LED* um die Bezeichnung für eine Ganzzahl handelt, die nicht geändert werden kann (man nennt das auch eine Konstante) und für die der Wert 13 festgelegt wurde. Das ist vergleichbar mit einem automatischen Suchen-&-Ersetzen-Vorgang im Code. In unserem Fall wird der Arduino angewiesen, immer wenn das Wort *LED* erscheint, die Zahl 13 zu schreiben.

Der Befehl wird hier verwendet, um festzulegen, dass die vorinstallierte LED, die wir schon erwähnt haben, an den Arduino-Pin 13 angeschlossen ist. Eine übliche Konvention besteht darin, Großbuchstaben für Konstanten zu verwenden.

```
void setup()
```

Mit dieser Zeile wird dem Arduino mitgeteilt, dass es sich beim nächsten Code-Block um eine Funktion namens `setup()` handelt.

```
{
```

Mit der öffnenden geschweiften Klammer wird ein Code-Block eingeleitet.

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

Endlich eine wirklich interessante Anweisung! Mittels `pinMode()` wird dem Arduino mitgeteilt, wie ein bestimmter Pin konfiguriert werden soll. Digitale Pins können entweder als INPUT oder OUTPUT (Eingang oder Ausgang) verwendet werden. Wir müssen dem Arduino aber mitteilen, wie wir einen Pin benutzen möchten.

In unserem Beispiel benötigen wir einen Ausgangs-Pin, um die LED zu steuern.

Bei `pinMode()` handelt es sich um eine Funktion, und die Wörter (oder Zahlen) innerhalb der Klammern werden als Argumente bezeichnet. Argumente sind jegliche Informationen, die eine Funktion benötigt, um ihre Aufgabe zu erfüllen.

Die Funktion `pinMode()` benötigt zwei Argumente. Mit dem ersten Argument wird `pinMode()` mitgeteilt, auf welchen Pin wir uns beziehen, und mit dem zweiten Argument erhält `pinMode()` die Information, ob wir den Pin als Input oder Output nutzen möchten. `INPUT` und `OUTPUT` sind vordefinierte Konstanten in der Arduino-Sprache.

Erinnere dich daran, dass es sich bei dem Wort *LED* um den Namen der Konstanten handelt, für die die Zahl 13 festgelegt wurde, die Nummer des Pins, an den die LED angeschlossen ist. Das erste Argument ist also `LED`, der Name der Konstanten.

Das zweite Argument ist `OUTPUT`, denn wenn ein Arduino mit einem Aktor kommuniziert, werden Informationen *aus*gesendet.

```
}
```

Die schließende geschweifte Klammer zeigt das Ende der `setup()`-Funktion an.

```
void loop()
{
```

In `loop()` wird das hauptsächliche Verhalten des interaktiven Bauteils festgelegt. Die Funktion wird immer weiter wiederholt, und zwar solange das Board mit Strom versorgt wird.

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

Wie der Kommentar schon sagt, ist `digitalWrite()` in der Lage, jeden Pin, der als OUTPUT konfiguriert wurde, ein- oder auszuschalten. Genau wie wir es bei der Funktion `pinMode()` gesehen haben, erwartet auch `digitalWrite()` zwei Argumente, und wie bei der Funktion `pinMode()` teilt das erste Argument `digitalWrite()` mit, auf welchen Pin wir uns beziehen, und wie bei der Funktion `pinMode()` benutzen 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 unterscheidet sich davon: In diesem Fall teilt das Argument `digitalWrite()` mit, ob der Spannungspegel auf 0 (`LOW`) oder auf 5V (`HIGH`) gesetzt werden soll.

Stell dir vor, dass der Ausgangs-Pin so eine kleine Steckdose ist wie die in den Wänden deiner Wohnung. Europäische Steckdosen liefern 230V, amerikanische 110V und Arduino arbeitet mit gemäßigten 5V. Die Magie offenbart sich hier, wenn die Software die Hardware steuern kann. Wenn du `digitalWrite(LED, HIGH)` schreibst, wird der Ausgangs-Pin auf 5V gesetzt. Schließt du dann die LED an, leuchtet sie. An dieser Stelle im Code bewirkt eine Anweisung in der Software eine Reaktion in der physikalischen Welt, indem der Stromfluss zum Pin gesteuert wird. Das Ein- und Ausschalten des Pins lässt sich in etwas für den Menschen besser Sichtbares übertragen; die LED ist unser *Aktor*.

Auf dem Arduino bedeutet HIGH, dass 5V am Pin anliegen, wohingegen bei LOW der Pin auf 0V gesetzt ist.

Du fragst dich vielleicht, warum wir HIGH und LOW anstelle von ON und OFF verwenden. Es stimmt, dass HIGH oder LOW normalerweise ON bzw. OFF entsprechen, dies hängt aber davon ab, wie der Pin benutzt wird. Eine LED beispielsweise, die zwischen 5V und einem Pin angeschlossen ist, wird eingeschaltet, wenn dieser Pin LOW ist, und ausgeschaltet, wenn er HIGH ist. In den meisten Fällen kannst du allerdings nur vorspiegeln, dass HIGH gleichbedeutend ist mit ON und LOW mit OFF.

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

Obwohl der Arduino sehr viel langsamer als dein Laptop ist, ist er doch immer noch sehr schnell. Wenn wir die LED schnell an- und sofort wieder ausschalten würden, könnten deine Augen dies nicht sehen. Wir müssen die LED eine Weile eingeschaltet lassen, damit wir es sehen können. Hierzu müssen wir den Arduino anweisen, eine Zeitlang zu warten, bevor mit dem nächsten Schritt fortgefahren wird. Mit `delay()` weist du im Grunde genommen den Prozessor an, zu pausieren und nichts zu tun, und zwar für die Zeitdauer in Millisekunden, die du als Argument übergibst. Eine Millisekunde ist ein Tausendstel einer Sekunde, also sind 1.000 Millisekunden eine Sekunde. In unserem Beispiel wird die LED demnach eine Sekunde lang leuchten.

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

Mit dieser Anweisung wird die LED, die wir vorher eingeschaltet haben, ausgeschaltet.

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

An dieser Stelle bauen wir eine weitere Verzögerung von einer Sekunde ein. Die LED bleibt eine Sekunde lang ausgeschaltet.

```
    }
```

Die schließende geschweifte Klammer zeigt das Ende der loop-Funktion an. Wenn der Arduino hier angelangt ist, wird wieder beim Beginn von `loop()` gestartet.

Zusammengefasst macht das Programm Folgendes:

- Pin 13 wird als Ausgangs-Pin definiert (nur einmal zu Beginn).
- Es erfolgt der Eintritt in eine Schleife.
- Die LED, die mit Pin 13 verbunden ist, wird eingeschaltet.
- Es folgt eine Wartezeit von einer Sekunde.
- Die LED, die mit Pin 13 verbunden ist, wird ausgeschaltet.
- Es folgt eine Wartezeit von einer Sekunde.
- Es wird ein Sprung zurück an den Anfang der Schleife durchgeführt.

Wir hoffen, dass dir dieser Code noch keine allzu großen Kopfschmerzen bereitet hat. Lass dich nicht entmutigen, wenn du nicht alles verstanden hast. Dir sind diese Konzepte ja noch neu. Es wird eine Weile dauern, bis sich ihr

Sinn wirklich erschließt. Du wirst in den späteren Beispielen noch mehr zum Thema Programmierung erfahren.

Bevor wir zum nächsten Abschnitt kommen, wollen wir noch ein wenig mit dem Code spielen. Wir könnten zum Beispiel die Anzahl der Verzögerungen reduzieren und dabei verschiedene Zahlen für die Ein- und Ausphasen verwenden, so dass wir unterschiedliche Blinkmuster beobachten können. Insbesondere solltest du darauf achten, was geschieht, wenn die Verzögerungen sehr klein sind und sich in den Ein- und Ausphasen unterscheiden ... du kannst dabei nämlich für einen Moment etwas beobachten, das später in diesem Buch, wenn wir zum Stichwort *Pulsweitenmodulation* in Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 kommen, noch sehr nützlich sein wird.

Was wir bauen werden

> Ich war immer fasziniert von Licht und der Möglichkeit, verschiedene Lichtquellen mittels Technologie zu steuern. Ich hatte das Glück, an einigen interessanten Projekten zu arbeiten, die damit befasst waren, Licht zu steuern und es mit lebenden Personen interagieren zu lassen. Arduino bietet diesbezüglich wirklich gute Möglichkeiten.
>
> Massimo

In diesem Kapitel sowie in den Kapiteln 5 und 7 werden wir uns damit befassen, wie sich *interaktive Lampen* herstellen lassen. Mithilfe des Arduino, den wir hier verwenden, werden wir die Grundlagen kennenlernen, die erforderlich sind, um interaktive Geräte zu bauen. Denk dabei immer daran, dass der Arduino nicht wirklich versteht und es ihm gleichgültig ist, was du am Ausgangs-Pin anschließt. Der Arduino setzt einen Pin einfach auf HIGH oder LOW, wodurch sich ein Licht, ein elektrischer Motor oder dein Automotor steuern lässt.

Im nächsten Abschnitt werden wir versuchen, die Grundlagen der Elektrizität auf eine Art und Weise zu erläutern, die zwar einen Ingenieur sicherlich langweilen würde, aber dafür auch einen Einsteiger in die Arduino-Programmierung nicht sofort abschreckt.

Was ist Elektrizität?

Wenn du zu Hause schon einmal Klempnerarbeiten durchgeführt hast, wirst du in puncto Elektronik keine Verständnisschwierigkeiten haben. Der beste Weg, zu vermitteln, wie Elektrizität und elektrische Schaltungen funktionieren, ist die "Wasser-Analogie". Nehmen wir ein einfaches Gerät wie den batteriebetriebenen, tragbaren Ventilator, der in Abbildung 4-4 gezeigt wird.

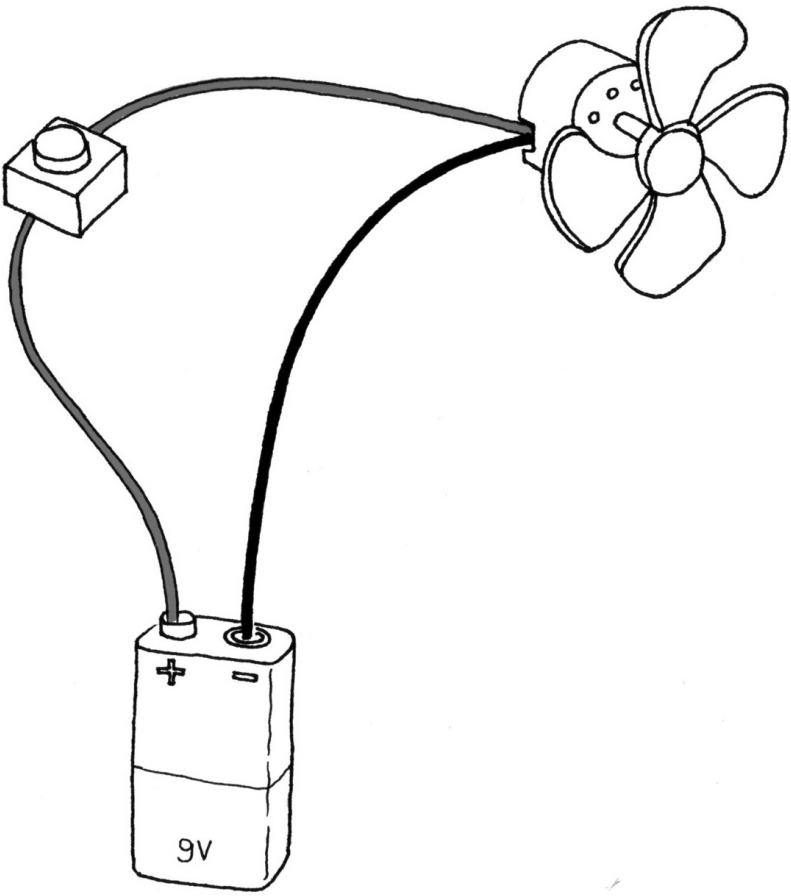


Abbildung 4-4: *Ein portabler Ventilator*

Wenn du den Ventilator auseinanderbaust, wirst du sehen, dass er eine kleine Batterie, einige Drähte und einen elektrischen Motor enthält und dass einer der Drähte, die zum Motor führen, durch einen Schalter unterbrochen ist. Wenn du den Schalter betätigst und den Motor einschaltest, beginnt er, sich zu drehen, und sorgt so für die Luftzirkulation, die dir dann die gewünschte Abkühlung bringt.

Wie funktioniert das? Stell dir einfach vor, die Batterie sei zugleich ein Wasserreservoir und eine Pumpe, der Schalter ein Ventil und der Motor eines von diesen Wasserrädern, die du sicher schon bei Windmühlen gesehen hast. Wenn du das Ventil öffnest, fließt das Wasser von der Pumpe zum Wasserrad und treibt es an.

Bei diesem einfachen Beispiel, das in Abbildung 4-5 dargestellt ist, sind zwei Faktoren wichtig: der Wasserdruck (der von der Leistung der Pumpe bestimmt wird) und die Wassermenge, die durch die Leitung fließt (die vom Durchmesser der Leitung und vom Widerstand, den das Wasserrad dem auftreffenden Wasserstrom entgegensetzt, abhängt).

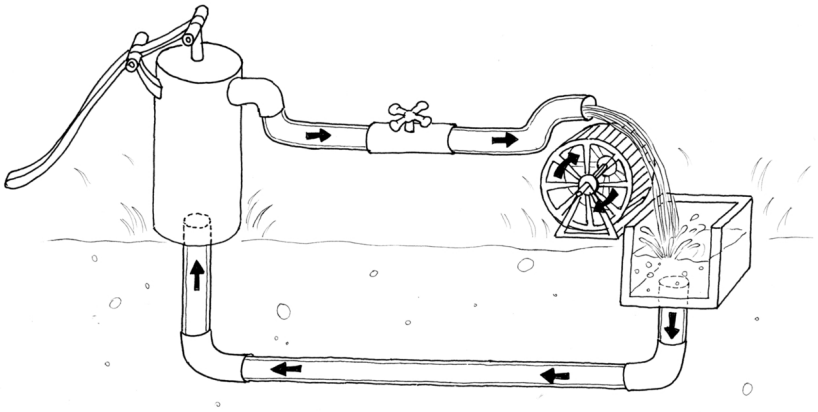


Abbildung 4-5: *Ein Hydrauliksystem*

Du wirst schnell bemerken, dass es zur Erhöhung der Drehgeschwindigkeit des Rades erforderlich ist, den Durchmesser der Leitungen zu vergrößern (was nur bis zu einem bestimmten Punkt funktioniert) und den Druck zu erhöhen, der durch die Pumpe erzielt wird. Durch das Vergrößern des Durchmessers der Leitungen kann mehr Wasser durch sie hindurchfließen. Durch diesen größeren Durchmesser wird der Widerstand in Bezug auf den Wasserdurchfluss verringert. Dieser Ansatz funktioniert bis zu einem bestimmten Punkt, ab dem sich das Rad nicht mehr schneller dreht, weil der Wasserdruck nicht ausreicht. Wenn dieser Punkt erreicht wurde, muss die Pumpleistung erhöht werden. Diese Möglichkeit der Beschleunigung des Wasserrades funktioniert so lange, bis das Rad wegen des zu starken Wasserdrucks auseinanderbricht und zerstört wird. Ein anderer Aspekt, der sich beobachten lässt, ist die Wärmeentwicklung an der Achse, die entsteht, wenn sich das Rad dreht. Egal, wie gut das Wasserrad montiert ist, durch die Reibung zwischen der Achse und der Vorrichtung, in der sie montiert ist, wird Wärme erzeugt. Es ist wichtig, zu verstehen, dass bei einem System wie diesem nicht die gesamte zugeführte Energie in Bewegung umgewandelt wird, sondern ein Teil der Energie verlorengeht. Sie zeigt sich dann im Allgemeinen als Wärme, die von einzelnen Komponenten im System abgegeben wird.

Was sind also die wichtigen Aspekte bei diesem System? Einer ist der durch die Pumpe erzeugte Druck, die anderen sind der Widerstand, der dem Wasserstrom durch die Leitung und das Wasserrad entgegengesetzt wird, und

der eigentliche Wasserdurchfluss (der dargestellt wird als die Anzahl an Litern, die pro Sekunde fließen). Elektrizität funktioniert ein wenig wie Wasser. Du hast eine Art Pumpe (jede Art von Energiequelle, z.B. eine Batterie oder eine Steckdose in der Wand), die elektrische Ladungen (die du dir am besten als kleine elektrische "Tropfen" vorstellst) durch Leitungen drückt, die in Form von Drähten realisiert sind. Diese elektrischen Tropfen werden von einigen Geräten verwendet, um Wärme zu produzieren (Großmutters Heizdecke), Licht zu erzeugen (deine Nachttischlampe), Sound herzustellen (deine Stereoanlage), Bewegung anzustoßen (unser Ventilator) und für viele weitere Dinge.

Wenn du auf einer Batterie die Angabe 9V liest, stell dir diese elektrische Spannung einfach als Wasserdruck vor, der mittels einer kleinen Pumpe erzeugt wird. Elektrische Spannung wird in Volt gemessen. Diese Einheit wurde nach Alessandro Volta benannt, dem Erfinder der ersten Batterie.

Wie der Wasserdruck hat auch die Durchflussmenge des Wassers ein Äquivalent in der Elektrizität. Sie wird als Strom bezeichnet, der in Ampere gemessen wird (nach André-Marie Ampère, einem Pionier des Elektromagnetismus). Das Verhältnis von elektrischer Spannung und Strom kann wieder anhand des Beispiels mit dem Wasserrad veranschaulicht werden: Ein höherer Wasserdruck (elektrische Spannung) bewirkt eine schnellere Drehung des Rades, mit einer höheren Durchflussrate (Strom) lässt sich ein größeres Rad antreiben.

Der Widerstand schließlich, der dem Stromfluss auf jedem Weg, den er zurücklegt, entgegengesetzt wird, heißt, wie du bestimmt schon erraten hast, auch in der Elektronik Widerstand und wird in Ohm gemessen (nach einem deutschen Physiker). Herr Ohm formulierte auch das wichtigste Gesetz in der Elektrizität, und die betreffende Formel ist auch die einzige, die du dir wirklich merken musst. Er konnte nachweisen, dass in jedem Schaltkreis eine Beziehung zwischen Strom und Widerstand besteht, genauer gesagt, dass bei einer gegebenen Spannung die Strommenge, die durch einen Schaltkreis fließt, vom vorhandenen Widerstand abhängt.

Bei genauerem Nachdenken ist das recht intuitiv zu verstehen. Schließe eine 9V-Batterie an einen einfachen Schaltkreis an. Wenn du nun den Strom misst, wirst du feststellen: Je mehr Widerstände du einbaust, desto geringer wird der Strom, der hindurchfließt. Wenn wir noch einmal auf das Beispiel mit dem Wasserdurchfluss in den Leitungen zurückkommen, heißt das Folgendes: Wenn ich hier ein Ventil einbaue (das sich mit einem variablen Widerstand in der Elektrizität vergleichen lässt) und dieses Ventil immer weiter schließe, erhöhe ich den Widerstand in Bezug auf den Wasserdurchfluss, und es fließt immer weniger Wasser durch die Leitungen. Ohm hat dieses Gesetz in folgenden Formeln zusammengefasst:

```
R (Widerstand) = V (Spannung) / I (Strom)
V = R * I
I = V/R
```

Bei diesem Gesetz ist ein intuitives Verständnis wichtig, daher bevorzuge ich die letzte Version (`I = V/R`), da Strom etwas ist, das als Ergebnis entsteht, wenn du eine bestimmte Spannung (der Druck) zu einem bestimmten Kreislauf (der Widerstand) hinzufügst. Die Spannung existiert, unabhängig davon, ob sie tatsächlich genutzt wird oder nicht, und der Widerstand existiert, egal, ob Elektrizität zum Einsatz kommt oder nicht. Strom entsteht aber erst dann, wenn diese beiden Größen zusammenkommen.

Steuerung einer LED mit einem Drucktaster

Eine LED zum Blinken zu bringen, war recht einfach, aber ich glaube nicht, dass du glücklich wirst, wenn deine Nachttischlampe ständig blinkt, während du versuchst, ein Buch zu lesen. Daher musst du sie irgendwie steuern können. Im vorherigen Beispiel war die LED ein Aktor, der von Arduino gesteuert wurde. Was fehlt, ist ein Sensor.

Für unser Beispiel verwenden wir die einfachste verfügbare Ausführung eines Sensors: einen Schalter in Form eines Drucktasters.

Wenn du einen Drucktaster in seine Einzelteile zerlegen würdest, wäre dir sehr schnell klar, dass es sich um ein sehr einfaches Bauteil handelt. Er besteht aus zwei Metallplättchen, die durch eine Feder voneinander separiert werden, und einer Plastikkappe, die, wenn sie gedrückt wird, die zwei Metallplättchen miteinander verbindet. Wenn keine Verbindung zwischen den Metallplättchen besteht, erfolgt keine Stromzirkulation im Drucktaster (ähnlich wie bei einem geschlossenen Ventil). Wenn du den Taster aber drückst, stellst du eine Verbindung her.

Alle Schalter bestehen grundlegend aus folgenden Elementen: zwei (oder mehr) Metallteile, die miteinander in Berührung gebracht werden können, damit Elektrizität von einem zum anderen Metallteil fließen kann, oder die getrennt werden können, um den Stromfluss zu unterbinden.

Um den Status eines Schalters zu überwachen, möchte ich hier eine neue Arduino-Anweisung einführen: die Funktion `digitalRead()`.

`digitalRead()` überprüft, ob irgendeine Spannung an dem Pin anliegt, den du in den runden Klammern angegeben hast, und gibt einfach den Wert HIGH oder LOW zurück, je nachdem, was die Überprüfung ergeben hat. Die anderen Anweisungen, die du bisher verwendet hast, geben keinerlei Informationen zurück – sie führen nur das aus, was wir von ihnen möchten. Diese Art Anweisungen ist aber ein wenig begrenzt, da du bei einer sehr vorhersagbaren Abfolge von Anweisungen stecken bleibst, ohne Input aus der sie umgebenden Welt. Mittels `digitalRead()` kannst du eine Frage an den Arduino richten, und wir erhalten eine Antwort, die wiederum irgendwo gespeichert und sofort im Anschluss oder auch später zur Entscheidungsfindung herangezogen wird.

Bau' die Schaltung, die in Abbildung 4-6 dargestellt ist. Hierzu benötigst du einige Bauteile (sie werden auch bei anderen Projekten nützlich sein):

- Eine lötfreie Steckplatine
- Einen Satz vorgefertigter Steckbrücken
- Einen 10K-Ohm-Widerstand – 10er-Pack)
- Einen Drucktastschalter, 10er-Pack)

Alternativ zum Kauf vorgefertigter Steckbrücken kannst du auch Massivdraht vom Typ 22 AWG, der auf kleinen Spulen aufgewickelt ist, verwenden und ihn mit Drahtschneider und Abisolierzange abisolieren.

GND auf dem Arduino-Board steht für *Masse* (engl. ground). Der Begriff ist historisch, aber in unserem Fall bezeichnet er einfach den Minuspol beim Strom. Wir tendieren dazu, die Begriffe GND und Masse synonym zu verwenden. Du kannst dir die Masse als die Leitung vorstellen, die sich in der Wasseranalogie in Abbildung 4-5 quasi underground befindet.

Bei den meisten Schaltkreisen wird GND oder Masse sehr häufig genutzt. Daher besitzt dein Arduino-Board drei Pins mit der Bezeichnung GND. Sie sind alle miteinander verbunden, daher macht es keinen Unterschied, welchen du benutzt.

Der Pin mit der Bezeichnung *5V* ist der positive Pol beim Strom und immer 5 Volt höher als die Masse.

In Beispiel 4-2 wird der Code gezeigt, mit dem wir die LED mittels unseres Drucktastschalters steuern werden.

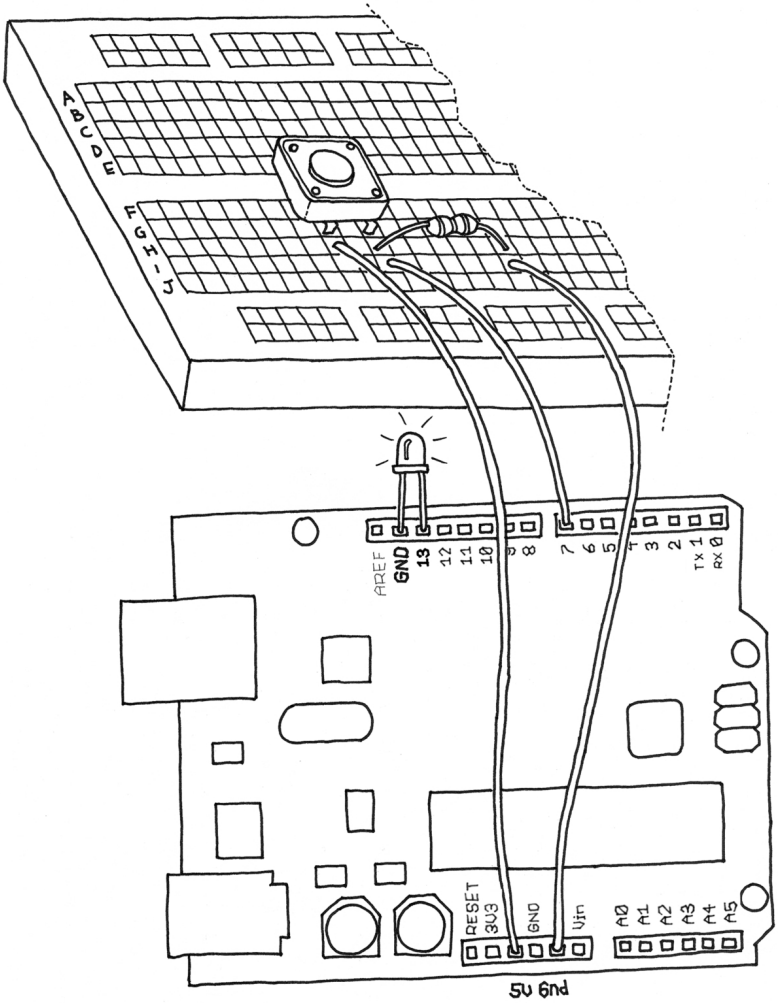


Abbildung 4-6: *Anschließen eines Drucktasters*

Beispiel 4-2: *LED mit Drucktaster anschalten*

```
// 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 im Arduino File→New aus (falls du einen anderen Sketch geöffnet hast, schließe ihn). Wenn du vom Arduino aufgefordert wirst, einen Namen für den neuen Sketch-Ordner anzugeben, gib PushButtonControl ein. Gib dann den Code für Beispiel 4-2 ein oder lade ihn von der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunter und kopiere ihn in die Arduino-IDE. Ist alles korrekt abgelaufen, wird die LED leuchten, wenn du den Taster drückst.

Erläuterung der Funktionsweise

Mit diesem Beispielprogramm habe ich zwei neue Konzepte eingeführt: Funktionen, die das Ergebnis ihrer Arbeit zurückliefern, und die `if`-Anweisung.

Die `if`-Anweisung ist wahrscheinlich die wichtigste Anweisung in einer Programmiersprache, weil sie dem Computer (denk immer daran, dass der Arduino ein kleiner Computer ist) ermöglicht, Entscheidungen zu treffen. Nach dem Schlüsselwort `if` muss eine Frage, die in Klammern eingeschlossen ist, angefügt werden. Wenn die Antwort bzw. das Ergebnis wahr ist, wird der erste Code-Block ausgeführt, anderenfalls der Code-Block nach `else`.

Beachte hier, dass sich das Symbol == von dem Symbol = stark unterscheidet. Das erste Symbol wird verwendet, wenn zwei Dinge miteinander verglichen werden. Es wird dann entsprechend das Ergebnis TRUE oder FALSE zurückgeliefert; mit dem zweiten Symbol wird einer Variablen oder Konstanten ein Wert zugewiesen. Achte darauf, das korrekte Zeichen zu verwenden, denn diese Fehlerquelle ist recht groß, und im Falle eines Fehlers an dieser Stelle wird das Programm niemals funktionieren. Wir wissen, wovon wir sprechen, denn nach jahrelanger Programmiererfahrung unterläuft uns dieser Fehler immer noch.

Es ist wichtig zu wissen, dass der Schalter nicht direkt mit der LED verbunden ist. Dein Arduino-Sketch überprüft den Schalter und fällt dann die

Entscheidung, ob die LED ein- oder ausgeschaltet wird. Die Verbindung zwischen dem Schalter und der LED erfolgt wirklich in deinem Sketch.

Es ist natürlich sehr unpraktisch, mit dem Finger die ganze Zeit den Taster gedrückt halten zu müssen, wenn du Licht benötigst. Auch wenn man bedenkt, wie viel Energie verschwendet wird, wenn du dich von der Lampe fortbewegst und sie nicht nutzt, sie aber eingeschaltet lässt, wollen wir dennoch herausfinden, wie wir bewirken können, dass der Taster im aktivierten Modus fixiert wird.

Ein Schaltkreis – 1.000 Verhaltensweisen

Der große Vorteil von digitaler, programmierbarer Elektronik gegenüber klassischer Elektronik wird hier offensichtlich: Ich werde dir nun zeigen, auf welche Weise viele verschiedene Verhaltensweisen unter Verwendung desselben Schaltkreises wie im vorherigen Abschnitt implementiert werden können, indem einfach die Software entsprechend geändert wird.

Wie ich bereits erwähnt habe, ist es nicht sehr praktisch, die ganze Zeit den Taster mit dem Finger gedrückt halten zu müssen, um Licht zu haben. Wir müssen also eine Art "Gedächtnis" implementieren, und zwar in Form eines Software-Mechanismus, der speichert, wann wir den Taster gedrückt haben, und der die Lampe weiter leuchten lässt, auch wenn wir den Finger vom Taster genommen haben.

Hierzu verwenden wir eine sogenannte Variable. (Wir haben sie bereits einmal verwendet, aber ich habe sie noch nicht erläutert.) Eine *Variable* ist ein Ort im Arduino-Speicher, an dem Daten gespeichert werden können. Du kannst sie dir als Post-it vorstellen, das du verwendest, um dich an etwas zu erinnern, z.B. eine Telefonnummer: Du schreibst beispielsweise "Luisa 02 555 1212" darauf und klebst ihn an deinen Computerbildschirm oder an den Kühlschrank. In der Arduino-Sprache ist das ähnlich einfach: Du entscheidest einfach, welcher Datentyp gespeichert werden soll (z.B. eine Zahl oder ein Text) und vergibst einen Namen. Du kannst diese Daten dann speichern oder abrufen. Hier ein Beispiel:

```
int val = 0;
```

`int` bedeutet, dass die Variable eine Ganzzahl speichert. Dabei ist `val` der Name der Variablen und mit `= 0` wird ein Anfangswert von 0 zugewiesen.

Eine Variable kann, wie der Name schon sagt, überall im Code geändert werden, so dass du später im Programm

```
val =112;
```

schreiben kannst.

Hierdurch wird deiner Variablen ein neuer Wert, nämlich 112, zugewiesen.

Hast du bemerkt, dass beim Arduino jede Anweisung mit einem Semikolon endet? Auf diese Weise wird dem Compiler (dem Teil des Arduino, der deinen Sketch in ein vom Mikrocontroller ausführbares Programm umwandelt) angezeigt, dass eine Anweisung beendet ist und eine neue beginnt. Wenn du ein Semikolon an einer Stelle vergisst, an der es erforderlich ist, wird dein Sketch für den Compiler keinen Sinn ergeben.

Im folgenden Programm wird `val` verwendet, um das Ergebnis von `digitalRead()` zu speichern; alle Informationen, die der Arduino vom Input-Pin erhält, landen in der Variablen und bleiben dort gespeichert, bis sie von einer anderen Code-Zeile geändert werden. Beachte, dass Variablen einen Speichertyp verwenden, der als *RAM* bezeichnet wird. Diese Art Speicher ist recht schnell, doch wenn du dein Board ausschaltest, gehen alle im RAM gespeicherten Daten verloren (d.h. jede Variable wird auf ihren Anfangswert zurückgesetzt, wenn das Board wieder mit Energie versorgt wird). Deine Programme selbst werden in einem Flash-Speicher – dieselbe Art Speicher, wie sie auch bei Mobiltelefonen zum Speichern von Telefonnummern verwendet wird – gespeichert. Hier bleiben die Daten erhalten, auch wenn das Board ausgeschaltet wird.

Nun wollen wir eine andere Variable verwenden, mit der gespeichert wird, ob die LED ein- oder ausgeschaltet bleiben soll, nachdem wir den Finger vom Taster genommen haben. Beispiel 4-3 ist ein erster Versuch, dieses Ziel zu erreichen.

Beispiel 4-3: *LED mit Drucktaster anschalten und angeschaltet 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 sehen, dass er funktioniert ... irgendwie. Das Licht wechselt allerdings so schnell, dass du es gar nicht zuverlässig mit einem Tastendruck ein- oder ausschalten kannst.

Schauen wir uns einmal die interessanten Zeilen des Codes an: `state` ist eine Variable, die entweder 0 oder 1 speichert, um sich so zu merken, ob die LED ein- oder ausgeschaltet ist. Wenn der Taster freigegeben wurde, wird sie auf den Anfangswert 0 (LED aus) gesetzt.

Später lesen wir den aktuellen Zustand des Tasters aus, und wenn er gedrückt ist (`val == HIGH`), ändern wir ihn von 0 in 1 oder umgekehrt. Da `state` immer nur 1 oder 0 sein kann, verwende ich hier einen kleinen Trick. Er beinhaltet einen kleinen mathematischen Ausdruck, der auf der Idee basiert, dass 1-0 = 1 ist, und 1-1 = 0:

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

Die Zeile macht mathematisch gesehen vielleicht keinen Sinn, bei der Programmierung hingegen schon. Das Symbol = bedeutet "weise das Ergebnis von dem, was nach mir folgt, der Variablen zu, die vor mir angeführt ist" – in unserem Beispiel wird `state` als neuer Wert das Ergebnis von 1 minus dem alten Wert von `state` zugewiesen.

Später in diesem Programm kannst du sehen, dass wir `state` verwenden, um zu ermitteln, ob die LED ein- oder ausgeschaltet sein muss. Wie bereits erwähnt, führt das zu eher ungenauen Ergebnissen.

Dies liegt an der Art und Weise, wie der Taster ausgelesen wird. Der Arduino ist wirklich schnell. Die eigenen internen Anweisungen werden mit einer Rate von 16 Millionen pro Millisekunde ausgeführt – das sind einige Millionen Code-Zeilen pro Sekunde. Während du also mit deinem Finger den Taster drückst, liest der Arduino die Position des Schalters möglicherweise einige tausend Male und ändert dabei `state` entsprechend. Das Ergebnis wird also letztendlich unvorhersehbar; es könnte AUS lauten, wenn es eigentlich AN lauten sollte oder umgekehrt. So wie eine falsch laufende Uhr zweimal am Tag die Zeit korrekt wiedergibt, kann auch das Programm gelegentlich ein korrektes Verhalten aufweisen, sehr oft aber wird es falsch sein.

Wie kannst du dieses Problem beheben? Nun, du musst den exakten Zeitpunkt ermitteln, an dem der Taster gedrückt wird – das ist dann der einzige Moment, in dem `state` geändert werden muss. Dazu möchten wir den Wert von `val` speichern, bevor wir einen neuen Wert auslesen. Dadurch wird es

möglich, die aktuelle Position des Tasters mit der vorherigen zu vergleichen und `state` nur dann zu ändern, wenn der Taster von LOW zu HIGH wechselt.

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);
 }
}
```

Probier' das Programm aus, du hast es fast geschafft!

Möglicherweise hast du bemerkt, dass dieser Ansatz nicht ganz perfekt ist, was an einem anderen Problem bei mechanischen Schaltern liegt.

Bei Drucktastern handelt es sich um zwei Metallteilchen, die durch einer Feder auseinandergehalten werden und die miteinander in Berührung kommen, wenn du den Schalter drückst. Das klingt möglicherweise so, als wäre der Schalter vollständig eingeschaltet, wenn du den Taster drückst. Was aber tatsächlich geschieht, ist, dass die Metallteilchen voneinander abprallen wie ein Ball, der auf dem Boden hüpft.

Dieses Abprallen erfolgt zwar nur über eine kurze Distanz und für den Bruchteil einer Sekunde, dennoch verursacht es beim Schalter einen mehrmali-

gen Wechsel zwischen AN und AUS, bis dieses Prellen endet und der Arduino schnell genug ist, dies zu erfassen.

Wenn der Taster prellt, erhält Arduino eine Reihe rasch aufeinanderfolgender Ein- und Aussignale. Es wurden viele Möglichkeiten zum Entprellen entwickelt, aber in diesem einfachen Code-Abschnitt reicht es völlig aus, eine Verzögerung von 10 bis 50 Millisekunden einzubauen. Mit anderen Worten: Du wartest einfach ein wenig, bis das Prellen endet.

Beispiel 4-5 enthält den finalen Code:

Beispiel 4-5: *Eine weitere neue und verbesserte Formel für Tastendrücke – mit einfachem Entprellen!*

```
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);
 }
}
```

Tami (Masaaki) Takamiya hat ein wenig Extra-Code eingefügt, mit dem du möglicherweise ein besseres Ergebnis beim Entprellen erzielst:

```
    if ((val == LOW) && (old_val == HIGH)) {
        delay(10);
    }
```

Index

D

E

F

G

H

I

K

W

X

Z