



**9 Awesome Projects**  
written especially for young people!

# Adventures in Arduino



**Get busy with Arduino!**

Guide yourself through the complex activities of programming  
to create these amazing projects!

**Becky Stewart**

**WILEY**

# **Adventures in Arduino**

Becky Stewart

**WILEY**

This edition first published 2015

© 2015 John Wiley and Sons, Ltd.

*Registered office*

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at [www.wiley.com](http://www.wiley.com).

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

**Trademarks:** Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/ or its affiliates in the United States and/or other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Ltd. is not associated with any product or vendor mentioned in the book.

A catalogue record for this book is available from the British Library.

ISBN 978-1-118-94847-7 (paperback); ISBN 978-1-118-94846-0 (ePub); 978-1-118-94845-3 (ePDF)

*To every student ever told they had to choose between the arts and sciences.*

# About the Author

**BECKY STEWART** is an engineer and educator. She works with artists and designers to bring to life often crazy ideas—from shoes that show you how to get home to suspension bridges that can be played like giant harps. After completing a PhD in Electronic Engineering at Queen Mary University of London, Becky helped found Codasign, an education company that creates technology workshops for art galleries and museums. At Codasign she teaches artists and designers how to use electronics and code as creative tools. She documents her projects at <http://theleadingzero.com>.

# Acknowledgments

My first thanks go to Alexandra Deschamps-Sonsino, without whom I would have never started this book. I also offer my sincere gratitude to Alex for her work with Tinker that jumpstarted the Arduino community in London.

None of this would have been possible without the support of the amazing educators that form Codasign. I have learned much from Melissa Coleman and Pollie Barden about how to improve my Arduino teaching, and I am constantly learning from Adam Stark about how to better teach programming concepts. I can't stop thanking Emilie Giles—Codasign would grind to a halt without you; thank you for everything you do. I give a particularly huge thank-you to Liat Wassershstrom for all your feedback and expertise.

I'd like to also thank the artistic and editorial staff that helped shape this book. It has been greatly improved by your guidance.

Lastly, thank you to my family who have supported me in everything I do. Thank you to my parents, who provided a quiet place to sit and write, and to Ben, who has patiently tolerated our wedding planning and vacations being punctuated by writing.



# Publisher's Acknowledgements

Some of the people who helped bring this book to market include the following:

## **Editorial**

Series Creator: Carrie Anne Philbin

VP Consumer and Technology Publishing Director: Michelle Leet

Associate Director—Book Content Management: Martin Tribe

Professional Technology & Strategy Director: Barry Pruett

Acquisitions Editor: Aaron Black

Project Editor: Charlotte Kughen

Copy Editor: Grace Fairley

Technical Editor: Russell Barnes

Editorial Manager: Mary Beth Wakefield

Editorial Assistant: Jessie Phelps

## **Marketing**

Marketing Manager: Lorna Mein

Marketing Assistant: Polly Thomas

# Adventures in Arduino®

## Contents

[Cover](#)

[Title Page](#)

[About the Author](#)

[Introduction](#)

[What Is an Arduino?](#)

[What You Will Learn](#)

[Parts You Will Need](#)

[Tools You Will Need](#)

[Software You Will Need](#)

[Other Useful Materials](#)

[What I Assume You Already Know](#)

[How This Book Is Organised](#)

[Conventions](#)

[The Companion Website](#)

[Reaching Out](#)

[Adventure 1: Setting Up Your Arduino](#)

[What You Need](#)

[Downloading and Installing the Arduino Software on Your Computer](#)

[Using Blink to Test That Everything Is Set Up Correctly](#)

[Building an LED Circuit](#)

[Further Adventures with Arduino](#)

[Adventure 2: Reading from Sensors](#)

[What You Need](#)

[Adding More LEDs](#)

[Printing Messages to the Computer](#)

[Reading Data from a Potentiometer](#)

[Making Decisions in Code](#)

[Building a Status Message Sign](#)

[Further Adventures with Arduino](#)

[Adventure 3: Working with Servos](#)

[What You Need](#)

[Understanding Different Types of Motors](#)  
[Controlling a Servo with Arduino](#)  
[Repeating the Same Thing Over and Over](#)  
[Digital Input with a Push Button](#)  
[Building a Combination Safe](#)  
[Further Adventures with Arduino](#)

## **[Adventure 4: Using Shift Registers](#)**

[What You Need](#)  
[Organising Your Code](#)  
[Getting More Outputs with Shift Registers](#)  
[Building Your Name in Lights](#)  
[Further Adventures with Shift Registers](#)

## **[Adventure 5: Playing Sounds](#)**

[What You Need](#)  
[Making a List](#)  
[Making Noise](#)  
[Building an Augmented Wind Chime](#)  
[Further Adventures with Sound](#)

## **[Adventure 6: Adding Libraries](#)**

[What You Need](#)  
[Analogue Out](#)  
[Capacitive Sensing](#)  
[Building a Crystal Ball](#)  
[Further Adventures with Libraries](#)

## **[Adventure 7: Working with the Arduino Leonardo](#)**

[What You Need](#)  
[Introducing the Arduino Leonardo](#)  
[Sensing Light](#)  
[Building a Game Controller](#)  
[Further Adventures with the Leonardo](#)

## **[Adventure 8: Working with the Lilypad Arduino USB](#)**

[What You Need](#)  
[Introducing the Lilypad Arduino USB](#)  
[Getting Clever with Arrays](#)  
[Passing Data Between Functions](#)  
[Building a POV Hoodie](#)

[Further Adventures with the Lilypad](#)

## **[Adventure 9: The Big Adventure: Building a Marble Maze Game](#)**

[What You Need](#)

[Part One: Scoring Points](#)

[Part Two: Designing Your Maze Game](#)

[Part Three: Writing the Code](#)

[Part Four: Building the Maze Game](#)

[Further Adventures: Continuing Your Adventures with Arduino](#)

## **[Adventure A: Where to Go From Here](#)**

[More Boards, Shields, Sensors and Actuators](#)

[On the Web](#)

[Books](#)

## **[Adventure B: Where to Get Tools and Components](#)**

[Starter Kits](#)

[Brick-and-Mortar Stores](#)

[Online Stores](#)

## **[Glossary](#)**

## **[End User License Agreement](#)**

# List of Illustrations

## Introduction

[FIGURE I-1 Arduino Uno \(top left\), Arduino Leonardo \(bottom left\) and Lilypad Arduino USB \(right\)](#)

[FIGURE I-2 A USB and USB Micro cable](#)

[FIGURE I-3 Breadboards in different sizes and colours](#)

[FIGURE I-4 Jumper wires](#)

[FIGURE I-5 Different types of LED, with a colour-changing LED on the right and below it a Lilypad LED](#)

[FIGURE I-6 Resistors needed for the projects in this book: 100Ω \(top left\), 220Ω \(top right\), 10kΩ \(bottom left\), 1MΩ \(bottom middle\) and 10MΩ \(bottom right\)](#)

[FIGURE I-7 Three different types of potentiometer](#)

[FIGURE I-8 A servo motor](#)

[FIGURE I-9 A tactile push button \(left\) and three different panel mount buttons \(right\)](#)

[FIGURE I-10 A shift register](#)

[FIGURE I-11 A piezo](#)

[FIGURE I-12 A light-dependent resistor](#)

[FIGURE I-13 Male header pins](#)

[FIGURE I-14 Solid core wire \(left\) and stranded wire \(right\)](#)

[FIGURE I-15 Enamelled \(left\) and plastic coated \(right\) wire](#)

[FIGURE I-16 Solder on spools](#)

[FIGURE I-17 A 9V battery-to-DC-barrel connector \(left\) and a UK plug for a USB cable \(right\)](#)

[FIGURE I-18 Alligator clips](#)

[FIGURE I-19 Conductive thread](#)

[FIGURE I-20 A soldering iron](#)

[FIGURE I-21 Different kinds of wire stripper](#)

[FIGURE I-22 Wire cutters](#)

[FIGURE I-23 Pairs of pliers](#)

[FIGURE I-24 A multimeter](#)

[FIGURE I-25 A pair of scissors and a utility knife](#)

## Chapter 1

[FIGURE 1-1 An Arduino Uno and USB cable](#)

[FIGURE 1-2 You can download the Arduino IDE for your computer from the](#)

[Arduino website.](#)

[FIGURE 1-3 Plug the USB cable into the Arduino Uno and then connect it to your computer.](#)

[FIGURE 1-4 The Arduino program icon](#)

[FIGURE 1-5 The important parts of the Arduino IDE](#)

[FIGURE 1-6 The built-in LED on the Arduino board is near the number 13.](#)

[FIGURE 1-7 Opening the Blink sketch, which is located in the examples that are included with the Arduino IDE](#)

[FIGURE 1-8 Selecting the board you are using](#)

[FIGURE 1-9 Selecting the port your Arduino board is plugged into](#)

[FIGURE 1-10 Message in the Arduino IDE after successfully uploading your code](#)

[FIGURE 1-11 A common error when the computer can't talk with the Arduino](#)

[FIGURE 1-12 The electronic components you need to build the circuit](#)

[FIGURE 1-13 The circuit schematic for the LED circuit](#)

[FIGURE 1-14 A breadboard has a series of holes that are connected in rows with two pairs of long rows on the outside and shorter, perpendicular rows in the centre of the board.](#)

[FIGURE 1-15 Basic layout of a breadboard](#)

[FIGURE 1-16 The LED circuit on the breadboard](#)

[FIGURE 1-17 The digital pins on the Arduino board. Digital Pins 0 and 1 are special pins that you learn about later.](#)

## Chapter 2

[FIGURE 2-1 The electronic components you need for the first part of this adventure](#)

[FIGURE 2-2 Building a circuit to control three LEDs](#)

[FIGURE 2-3 The Serial Monitor button](#)

[FIGURE 2-4 The Serial Monitor in the Arduino IDE](#)

[FIGURE 2-5 Different potentiometers](#)

[FIGURE 2-6 Analogue pins on the Arduino Uno](#)

[FIGURE 2-7 Circuit for connecting a potentiometer](#)

[FIGURE 2-8 A status message sign](#)

[FIGURE 2-9 The electronic components you need to make a status message sign](#)

[FIGURE 2-10 Circuit schematic for the sign](#)

[FIGURE 2-11 Prototype circuit on the breadboard for the sign](#)

[FIGURE 2-12 Cutting holes for the LEDs and knob](#)

[FIGURE 2-13 The LED portion of the circuit](#)

[FIGURE 2-14 Soldered potentiometer](#)

[FIGURE 2-15 Power supply that you can use with an Arduino board](#)

[FIGURE 2-16 Completed status message sign](#)

## Chapter 3

[FIGURE 3-1 The electronic components you need for the first part of Chapter 3](#)

[FIGURE 3-2 A servo motor and toy DC motor](#)

[FIGURE 3-3 Opening the Sweep example sketch](#)

[FIGURE 3-4 Circuit to connect a servo to the Arduino board](#)

[FIGURE 3-5 How a tactile push button works](#)

[FIGURE 3-6 Circuit with a tactile push button](#)

[FIGURE 3-7 Circuit with a pull-up resistor](#)

[FIGURE 3-8 Circuit with a push button and internal pull-up resistor on the Arduino board](#)

[FIGURE 3-9 Combination safe](#)

[FIGURE 3-10 The electronic components you need to build your combination safe](#)

[FIGURE 3-11 Circuit schematic for the combination safe](#)

[FIGURE 3-12 Circuit for the combination safe](#)

[FIGURE 3-13 If the lid is not already attached to your box, add a paper hinge.](#)

[FIGURE 3-14 Extend the servo's arm by attaching an object like a paperclip or bamboo skewer.](#)

[FIGURE 3-15 Paper loop so the servo can close the safe](#)

[FIGURE 3-16 Soldered components](#)

[FIGURE 3-17 Completed combination safe](#)

## Chapter 4

[FIGURE 4-1 What you need for the first part of this adventure](#)

[FIGURE 4-2 The anatomy of a function](#)

[FIGURE 4-3 The anatomy of a for loop](#)

[FIGURE 4-4 The CLOCK signal](#)

[FIGURE 4-5 How a shift register works](#)

[FIGURE 4-6 Pin-out diagram for the shift register](#)

[FIGURE 4-7 First connections for the shift register](#)

[FIGURE 4-8 The full circuit for the shift register](#)

[FIGURE 4-9 How to convert from a binary number to a decimal number](#)

[FIGURE 4-10 How would this binary pattern be represented by a decimal number?](#)

[FIGURE 4-11 Adding a second shift register](#)

[FIGURE 4-12 Your name \(or any other word\) in lights!](#)

[FIGURE 4-13 The electronic components you need to build your name in lights](#)

[FIGURE 4-14 Circuit schematic for three shift registers](#)

[FIGURE 4-16 Cardboard letters with holes for LEDs](#)

[FIGURE 4-17 Soldered LEDs and resistors](#)

[FIGURE 4-18 Back of lights](#)

## Chapter 5

[FIGURE 5-1 The electronic components you need for the first part of this adventure](#)

[FIGURE 5-2 Two example arrays](#)

[FIGURE 5-3 The circuit for an array of LEDs](#)

[FIGURE 5-4 How sound is made](#)

[FIGURE 5-5 The circuit for a using a piezo as a speaker](#)

[FIGURE 5-6 An augmented wind chime](#)

[FIGURE 5-7 The electronic components you need to make the wind chime](#)

[FIGURE 5-8 Circuit schematic for the augmented wind chime](#)

[FIGURE 5-9 Breadboard prototype circuit](#)

[FIGURE 5-10 Conductivity test](#)

[FIGURE 5-11 A chime](#)

[Figure 5-12 A chime attached to the base](#)

[FIGURE 5-13 Top of base](#)

## Chapter 6

[FIGURE 6-1 The electronic components you need for the first part of this adventure](#)

[FIGURE 6-2 Analogue and digital signals](#)

[FIGURE 6-3 The pins that support analogwrite\(\)](#)

[FIGURE 6-4 LED circuit for fading an LED](#)

[FIGURE 6-5 Pulse width modulation examples](#)

[FIGURE 6-6 RGB LEDs](#)

[FIGURE 6-7 Circuit connecting an RGB LED to an Arduino board](#)

[FIGURE 6-8 Mixing light versus mixing paint](#)

[FIGURE 6-9 Place the downloaded and unzipped folder in the libraries folder of the Arduino sketchbook.](#)

[FIGURE 6-10 Check for the library and example in the menus.](#)

[FIGURE 6-11 Capacitive sensing circuit](#)

[FIGURE 6-12 A touch-sensitive crystal ball](#)

[FIGURE 6-13 The electronic components you need to make the crystal ball](#)

[FIGURE 6-14 Circuit schematic for the crystal ball](#)

[FIGURE 6-15 Breadboard prototype circuit](#)

[FIGURE 6-16 Mapping a value to a new range](#)

[FIGURE 6-17 Papier maché crystal ball](#)

[FIGURE 6-18 Aluminium foil-covered base](#)

[FIGURE 6-19 Soldered LED circuit](#)

[FIGURE 6-20 Soldered sensor circuit](#)

[FIGURE 6-21 Completed crystal ball circuit](#)

## Chapter 7

[FIGURE 7-1 The electronic components you need for the first part of this adventure](#)

[FIGURE 7-2 Selecting the Arduino Leonardo from Tools=Board in the Arduino IDE](#)

[FIGURE 7-3 USB connectors](#)

[FIGURE 7-4 The Leonardo typing in a word processing program](#)

[FIGURE 7-5 Analogue and digital signals](#)

[FIGURE 7-6 Ohm's Law defines how voltage, current and resistance are related.](#)

[FIGURE 7-7 Two voltage divider circuits, one with an LDR as the top resistance and the other with an LDR as the bottom resistance](#)

[FIGURE 7-8 The equation to calculate how different resistor values in a voltage divider change the output voltage](#)

[FIGURE 7-9 Arduino Leonardo game controller](#)

[FIGURE 7-10 The electronic components you need to make the game controller](#)

[FIGURE 7-11 Circuit schematic for the game controller](#)

[FIGURE 7-12 The game controller circuit](#)

[FIGURE 7-13 Cover without any circuitry](#)

## Chapter 8

[FIGURE 8-1 The electronic components you need for the first part of this adventure](#)

[FIGURE 8-2 The Lilypad Arduino USB](#)

[FIGURE 8-3 An FTDI programming board, which you need if you are using a type of Lilypad Arduino other than a Lilypad Arduino USB](#)

[FIGURE 8-4 The Arduino Lilypad Arduino USB ON switch](#)

[FIGURE 8-5 Select Lilypad Arduino USB from the list of boards](#)

[FIGURE 8-6 Instead of jumper wires to connect components, use alligator clips when prototyping soft circuits.](#)

[FIGURE 8-7 Lilypad LEDs are sewable LEDs that already have current-limiting resistors.](#)

[FIGURE 8-8 A list of integers, also called a one-dimensional array](#)

[FIGURE 8-9 A two-dimensional array of integers stored in rows and columns](#)

[FIGURE 8-10 Circuit for an array of LEDs](#)

[FIGURE 8-11 Iterating over frames of an animation stored in a two-dimensional array](#)

[FIGURE 8-12 Persistence-of-vision hoodie](#)

[FIGURE 8-13 The electronic components you need for the POV hoodie](#)

[FIGURE 8-14 Circuit schematic for the POV hoodie](#)

[FIGURE 8-15 Prototyping the circuit with alligator clips](#)

[FIGURE 8-16 Persistence of vision message captured with a long-exposure photograph](#)

[FIGURE 8-17 Bending the legs of components to make them sewable](#)

[FIGURE 8-18 First connections for sewing the Lilypad circuit](#)

[FIGURE 8-19 Continuing to sew the LEDs into the circuit](#)

[FIGURE 8-20 The sewn POV circuit](#)

## Chapter 9

[FIGURE 9-1 A completed big adventure marble maze game](#)

[FIGURE 9-2 The electronic components you need to build your maze game](#)

[FIGURE 9-3 Circuit to use a piezo as a sensor](#)

[FIGURE 9-4 Circuit for five piezos as sensors and one piezo as a speaker](#)

[FIGURE 9-5 Guidelines for designing your maze](#)

[FIGURE 9-6 How the code works when a game is played](#)

[FIGURE 9-7 Circuit schematic of the maze game](#)

[FIGURE 9-8 Maze game prototype circuit on a breadboard](#)

[FIGURE 9-9 How a `loop\(\)` works](#)

[FIGURE 9-0 Glue strips of card to guide the marble after it drops through a hole.](#)

[FIGURE 9-1 Lid of the maze game fitted to bottom](#)

[FIGURE 9-3 Wiring layout for piezos](#)

[FIGURE 9-4 Solder the negative legs of the LEDs and one contact of the button together.](#)

# Introduction

**ARE YOU AN** adventurer? Do you boldly embark on new endeavours, tackling new skills and mastering new tools? Do you want to learn how to use technology to make your ideas burst into life? Are you curious about how you can combine computer code and electrical circuits with scissors and paper—or even needle and thread? If the answer is an emphatic “yes” then this is the book for you!

# What Is an Arduino?

The Arduino is a tool for building computers that can interact with the physical world around you. You can use it to connect sensors that detect sound, light or vibration, then turn on a light, change its colour, move a motor and much more. The Arduino is the magical device that sits in the midst of all of these things. It reads in from sensors measuring the real world, makes decisions based on that data and then makes something happen in the real world, whether light, sound or movement.

The Arduino is usually a blue board about the size of your hand. It has white writing on it labelling its different sections and has all its chips and circuits exposed. There are different types of Arduino boards, and they aren't all blue, but you will learn more about that later in the "Parts You Will Need" section and also in [Adventures 7](#) and [8](#).

The Arduino is a microcontroller. A microcontroller is a simple computer. It can't do many things at the same time but it does what it is told to do really well. You already interact with lots of microcontrollers every day because they control things like microwaves and washing machines.

There are a lot of different types of microcontroller, but the special thing about Arduino is that it is designed for people who are just starting out. So, if you are new to code or electronics, that's okay because the Arduino is great for beginners. But don't underestimate it—it can still take on big projects.

# What You Will Learn

After completing these adventures, you will have learned how to set up the Arduino programming environment on your computer and how to write and upload code to your Arduino board. You will find out how to work with three different Arduino boards: the Uno, Leonardo and the Lilypad USB.

You will learn basic programming concepts that you can use beyond working with the Arduino. The Arduino language is based on the C/C++ language. This means that as you learn how to code Arduinos, you are also learning about how programming works on computers like a laptop or a Raspberry Pi.

Alongside programming, you will be introduced to circuits and electronics. You will learn how to use sensors to detect real-world signals like light or movement, and you will learn how to generate actions in the real world, such as playing a sound or turning on a light.

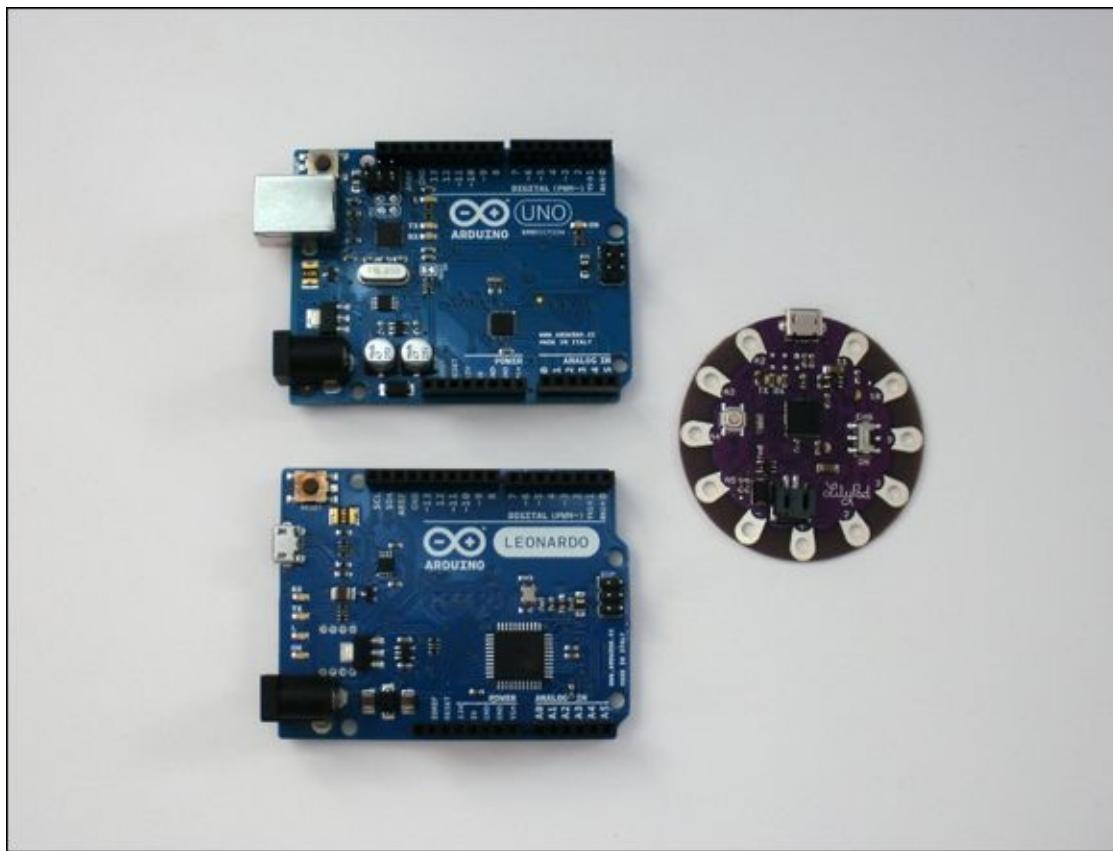
By the end of this book, you will have a broad understanding of what you can do with an Arduino and be ready to start designing and building project ideas of your own!

# Parts You Will Need

It's becoming easier to buy Arduino boards in stores. Popular retail chains like Maplin in the UK now stock Arduinos. Both of those stores also sell the electronic components that you need for the projects in this book. If it's not convenient for you to get to a store there are also many online retailers to choose from, and some of these are listed in Appendix B.

This section explains all the parts you need to make all the projects in this book. Many of the projects use the same core parts.

Of course, the most important thing you need is an Arduino board. There are many different kinds of Arduino boards, but the Arduino Uno is the most common one and the one you use the most in this book. You also need an Arduino Leonardo for [Adventure 7](#) and a Lilypad Arduino USB for [Adventure 8](#). All three boards are shown in [Figure I-1](#).



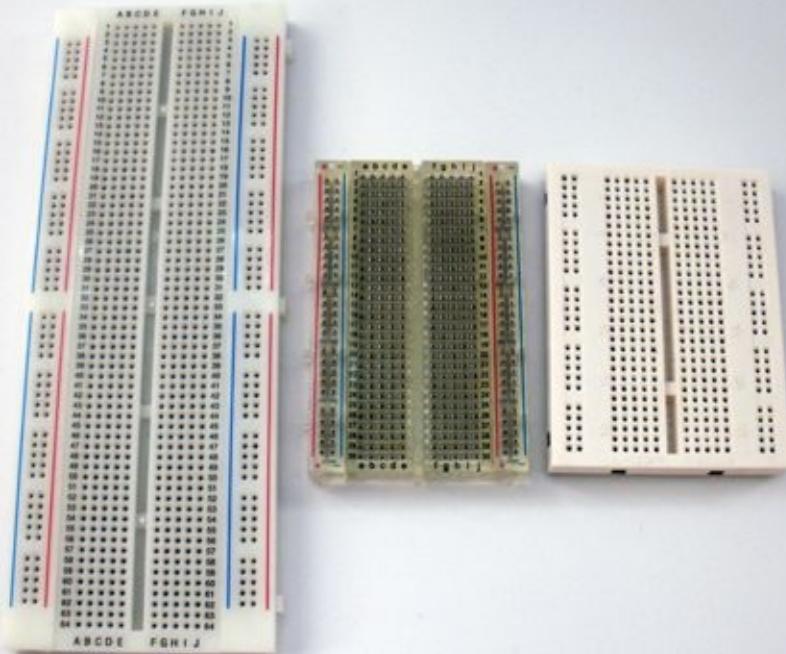
**FIGURE I-1** Arduino Uno (top left), Arduino Leonardo (bottom left) and Lilypad Arduino USB (right)

You will need a USB cable to connect your Arduino board to your computer. For the Arduino Uno you need a “normal” USB cable, but for the Arduino Leonardo and Lilypad Arduino USB you need a USB Micro cable. Both are pictured in [Figure I-2](#).



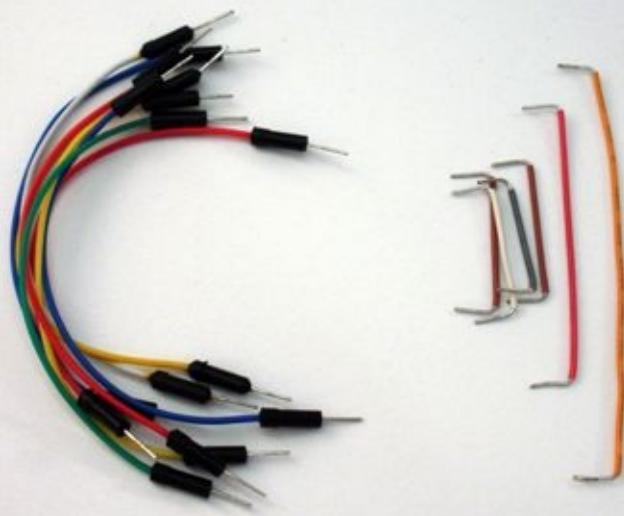
**FIGURE I-2** A USB and USB Micro cable

You use breadboards to build circuits. Breadboards let you connect components easily without having to use solder. They come in different colours and sizes. The larger ones are useful for more complicated projects with lots of parts, whereas the smaller ones are good for projects that you want to fit inside a small space. Two different sizes of breadboards made from two different types of plastic are shown in [Figure I-3](#). [Adventure 3](#) is the only project that uses a breadboard in the completed project; the other adventures use a breadboard only to test a circuit. A larger breadboard will be easier to work with, but if you can only find smaller ones, that's perfectly okay.



**FIGURE I-3** Breadboards in different sizes and colours

Jumper wires are wires you use when you build prototype circuits to try out new concepts. They may be short pieces of stiff wire like those shown on the right in [Figure I-4](#), or they may be more flexible wire with pins on either end like the ones on the left.



**FIGURE I-4** Jumper wires

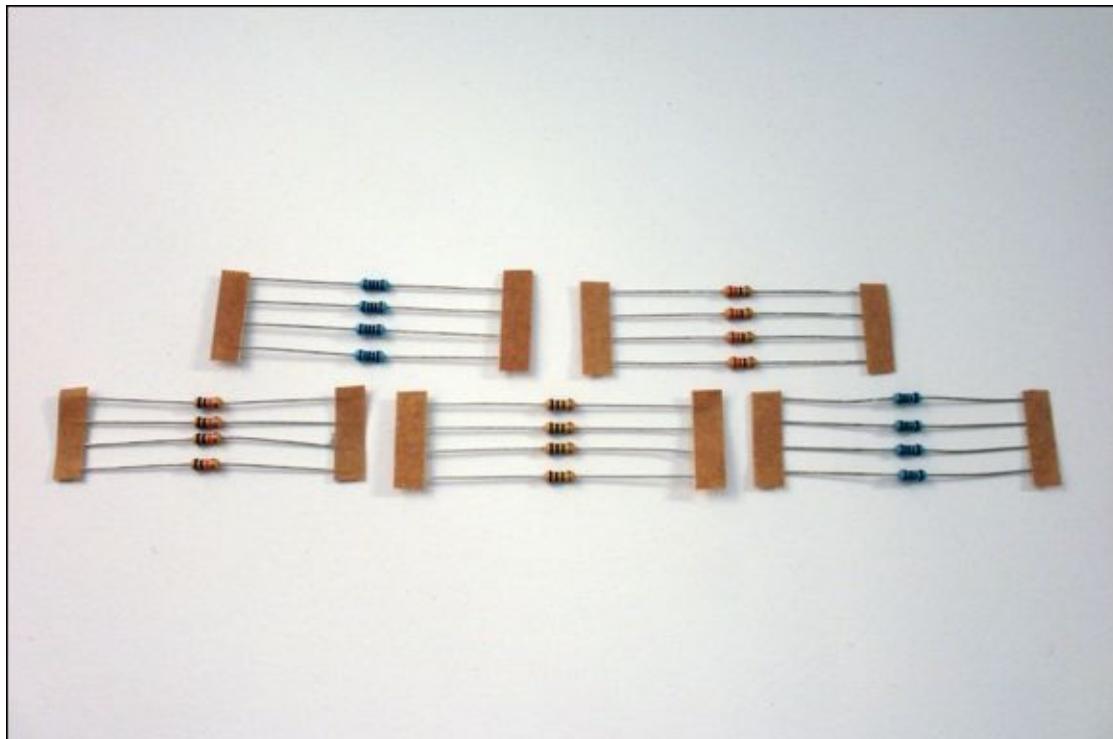
LEDs are a particular sort of light (LEDs stands for light-emitting diodes) that come in a

big selection of sizes and colours. For most of the projects in this book you can use whatever size and colour of LEDs you like. The most common size is 5 mm, but the larger 10 mm LEDs can be great fun to use too. Most LEDs are single-colour, but you use an LED in [Adventure 6](#) that has four legs instead of only two and can change colour. In [Adventure 8](#) you use something called a Lilypad LED, which is made especially for sewing circuits. All the different types of LED used in the projects are shown in [Figure I-5](#).



**FIGURE I-5** Different types of LED, with a colour-changing LED on the right and below it a Lilypad LED

Resistors are a component you read more about in the adventures. They come in different values of resistance, which is measured in ohms ( $\Omega$ ). You don't need many different resistances for the projects in the book but as resistors are small and quite cheap it's a good idea to buy extra. You need resistors of 68 or 100  $\Omega$ , 220  $\Omega$ , 10k (10,000)  $\Omega$ , 1M (1,000,000)  $\Omega$  and 10M (10,000,000)  $\Omega$ . [Figure I-6](#) shows the different resistors.



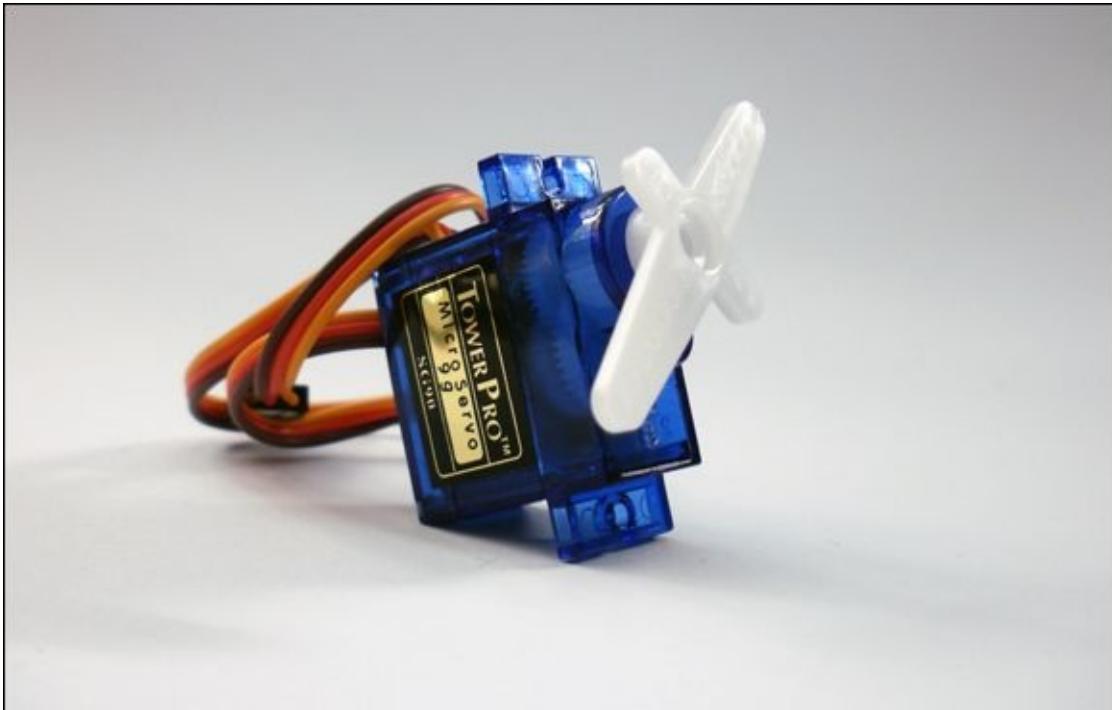
**FIGURE I-6** Resistors needed for the projects in this book: 100 $\Omega$  (top left), 220 $\Omega$  (top right), 10k $\Omega$  (bottom left), 1M $\Omega$  (bottom middle) and 10M $\Omega$  (bottom right)

Potentiometers are the electronic components behind volume knobs or dials on a stereo. They come in many different sizes and shapes. Some fit into a breadboard on their own, like the blue one in [Figure I-7](#), whereas others need wires soldered to them that can connect to a breadboard, like the one in the middle in [Figure I-7](#). Larger ones are easier to mount in a project and may be called panel-mount potentiometers.



**FIGURE I-7** Three different types of potentiometer

A servo, shown in [Figure I-8](#), is a motor that you use in [Adventure 3](#).



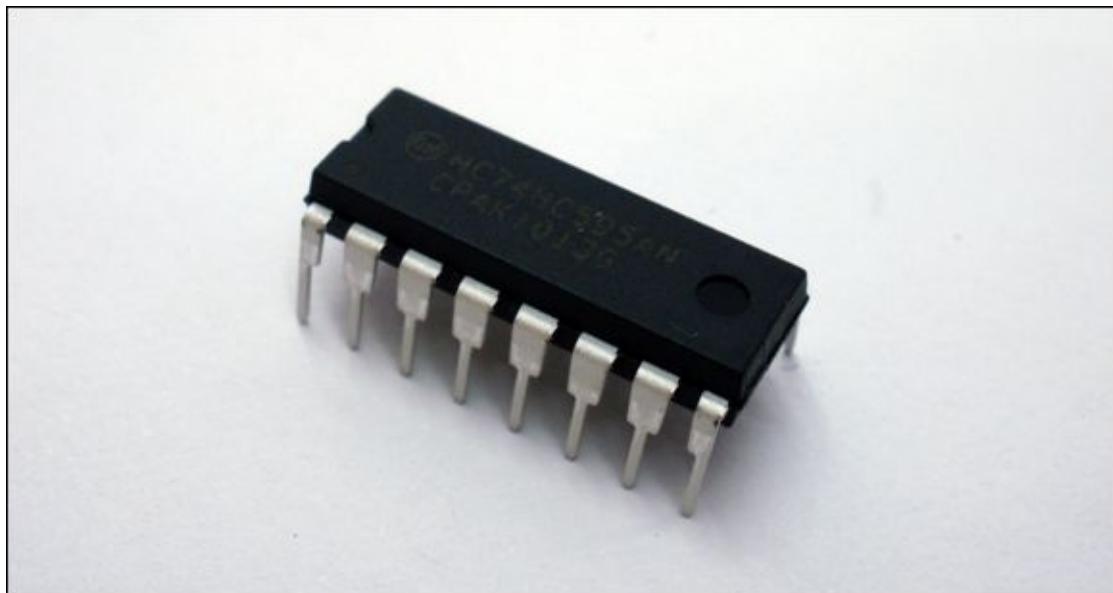
**FIGURE I-8** A servo motor

Buttons are another component that come in many shapes and sizes. You might have never noticed this before, but there are many different kinds of button! All the projects in this book use push-to-make (the opposite of push-to-break) buttons so those are the ones to buy; as long as they are push-to-make, you can use any kind of button you would like. Tactile push buttons are very little buttons that fit in a breadboard, so they are good to have when you are testing your circuit. For your actual projects, panel mount push buttons are better. Both are shown in [Figure I-9](#).



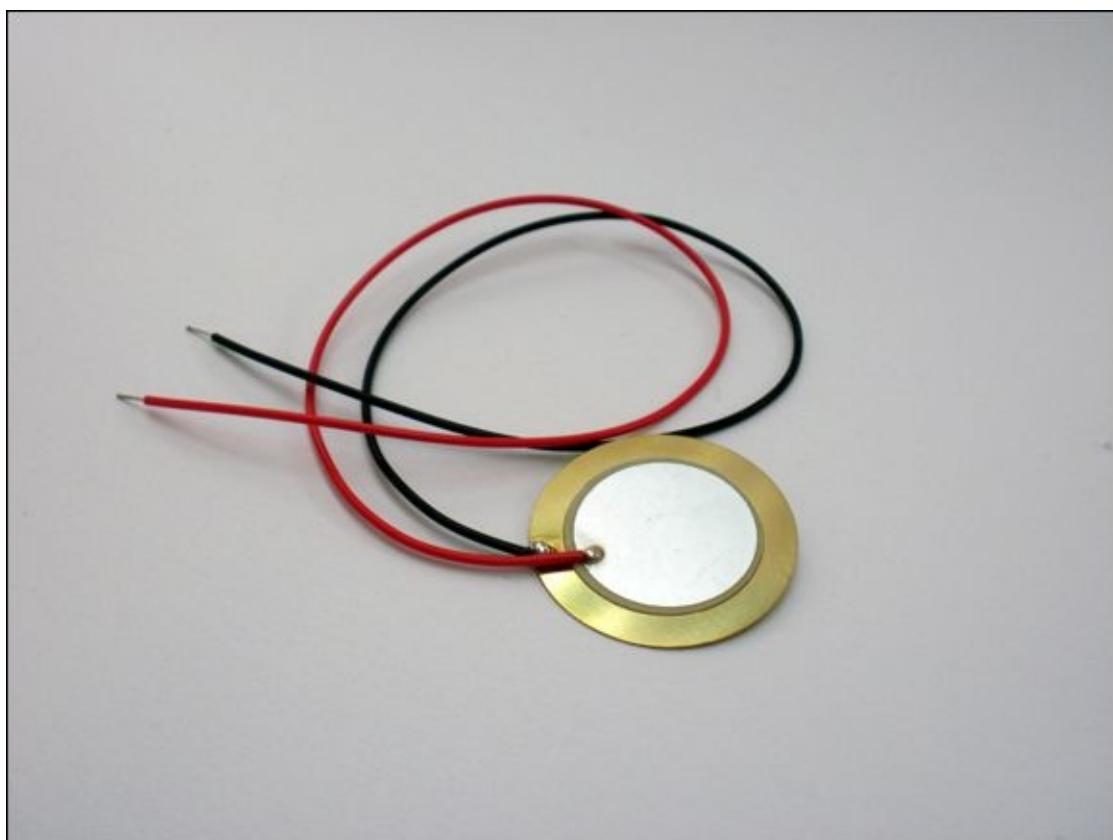
**FIGURE I-9** A tactile push button (left) and three different panel mount buttons (right)

In [Adventure 4](#) you discover how to use shift registers, which are small black chips you can use to control a lot of LEDs. You want a chip that is a 74HC595 shift register—you find out what that means in the adventure. You need to buy a chip with 16 legs on it, as shown in [Figure I-10](#).



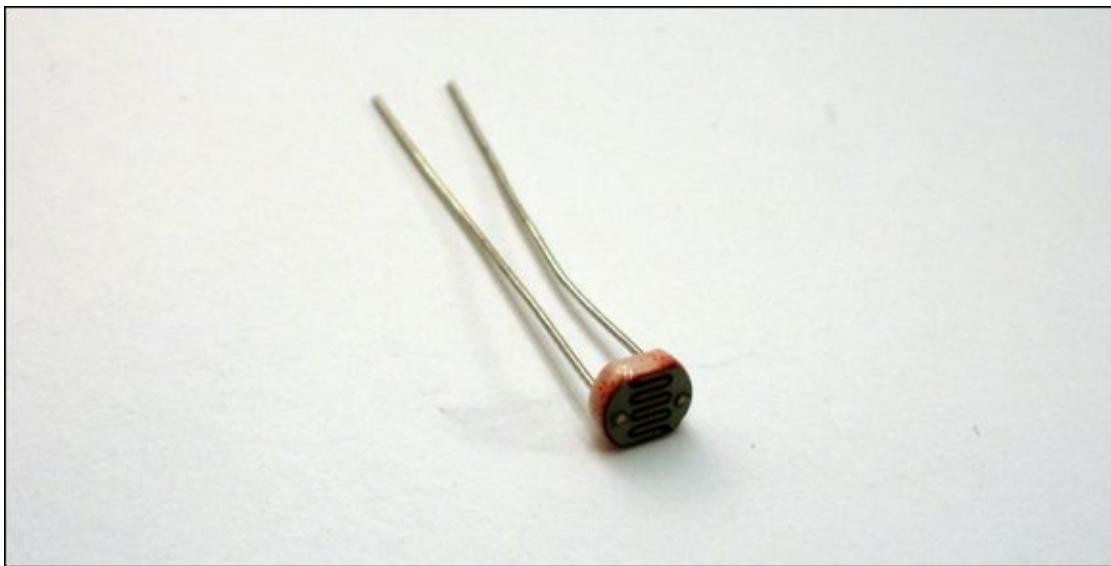
**FIGURE I-10** A shift register

Piezos are used to detect vibrations and can also make sound, like a speaker. You need one piezo for [Adventure 5](#) and six for [Adventure 9](#). They sometimes come inside black plastic housing, which is okay for the one in [Adventure 5](#) but you need at least five without housing (like the one in [Figure I-11](#)) for [Adventure 9](#).



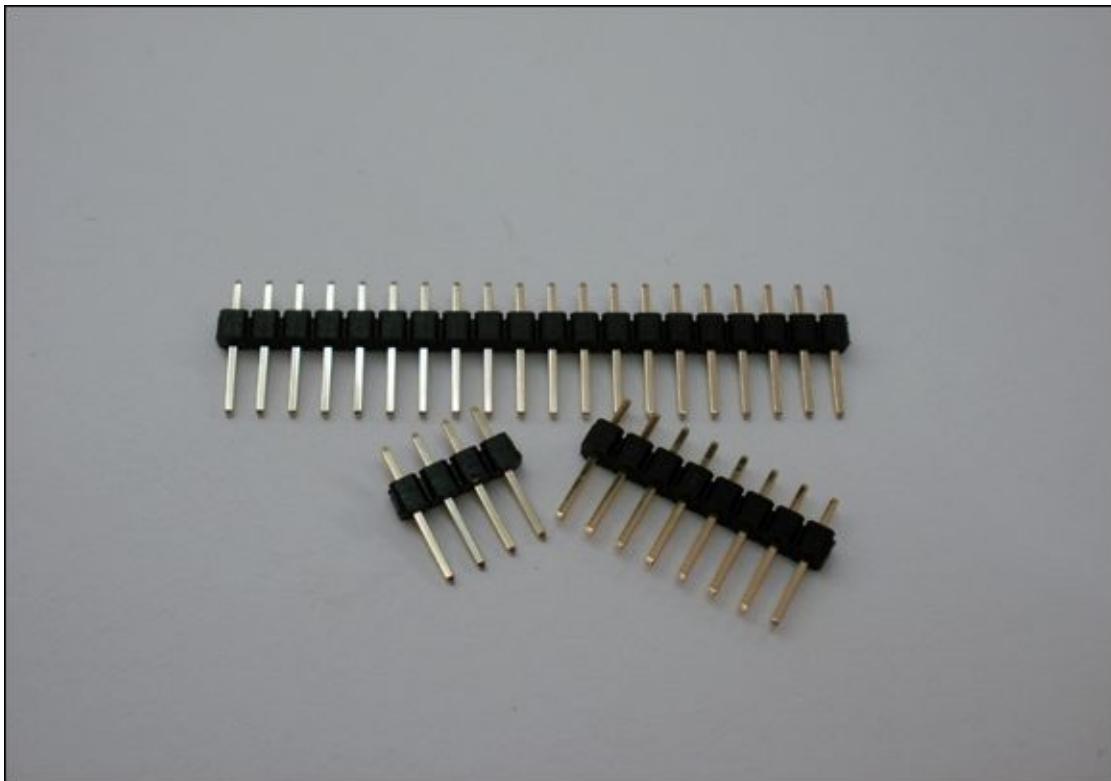
### **FIGURE I-11** A piezo

A light-dependent resistor can tell an Arduino board how bright or dark it is. These look like the one in [Figure I-12](#) or can be a little bigger.



### **FIGURE I-12** A light-dependent resistor

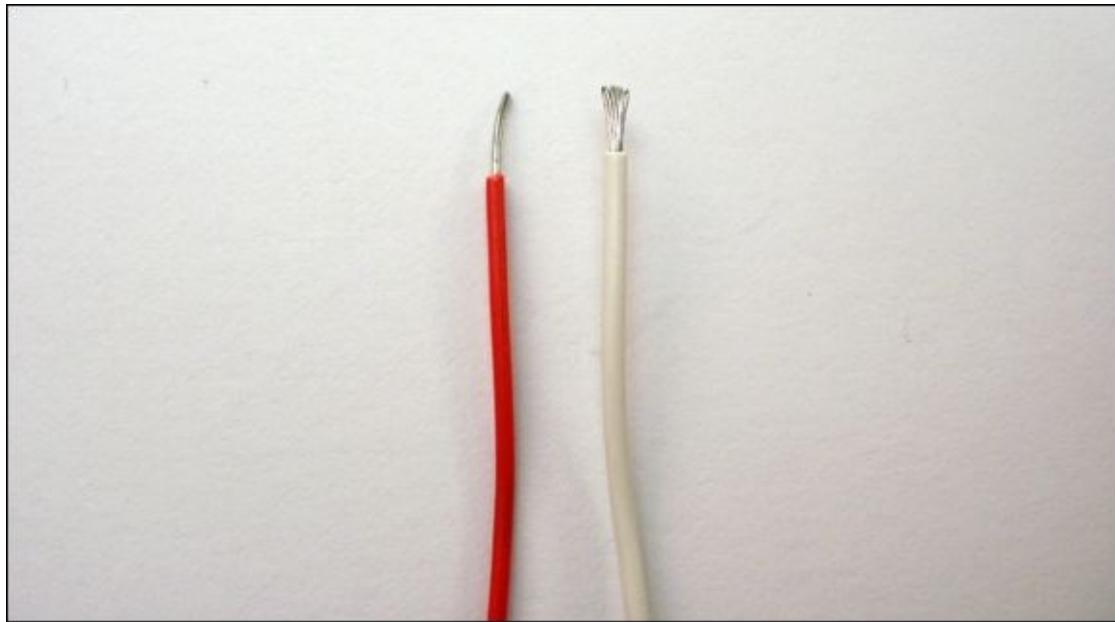
Header pins are small strips of metal that are separated by plastic so that they fit perfectly into the holes on the Arduino Uno. They come in different spacings (called pitches), so you should make sure you get 2.54 mm male header pins, like the ones in [Figure I-13](#). You need a strip of five for [Adventure 5](#), but you can buy them in longer strips and easily break them apart into smaller sections with pliers.



### **FIGURE I-13** Male header pins

When you think about circuits and electricity, one of the first things you picture is

probably wire. But wire isn't a single item; there are many different kinds. Wire can be made of a single piece of metal (called solid core) or a lot of smaller pieces of metal twisted together (called stranded). [Figure I-14](#) shows solid core and stranded wire. Solid core can be useful for breadboards but it's very stiff. Stranded is easier to bend, but you need to solder the end of it in order to get it to fit in a breadboard. You can decide for each project which sort you want to work with—there isn't a right or wrong type to use.



**FIGURE I-14** Solid core wire (left) and stranded wire (right)

Wire usually comes with some kind of coating that doesn't conduct electricity. It may be coloured plastic like the wire on the right in [Figure I-15](#), or it may be enamelled like the wire on the left. You can decide which wire works best in your projects. The enamelled wire works well in the augmented wind chime in [Adventure 5](#) because it's very thin and lets the chimes swing easily. However, you could build the wind chime using a different thin wire.



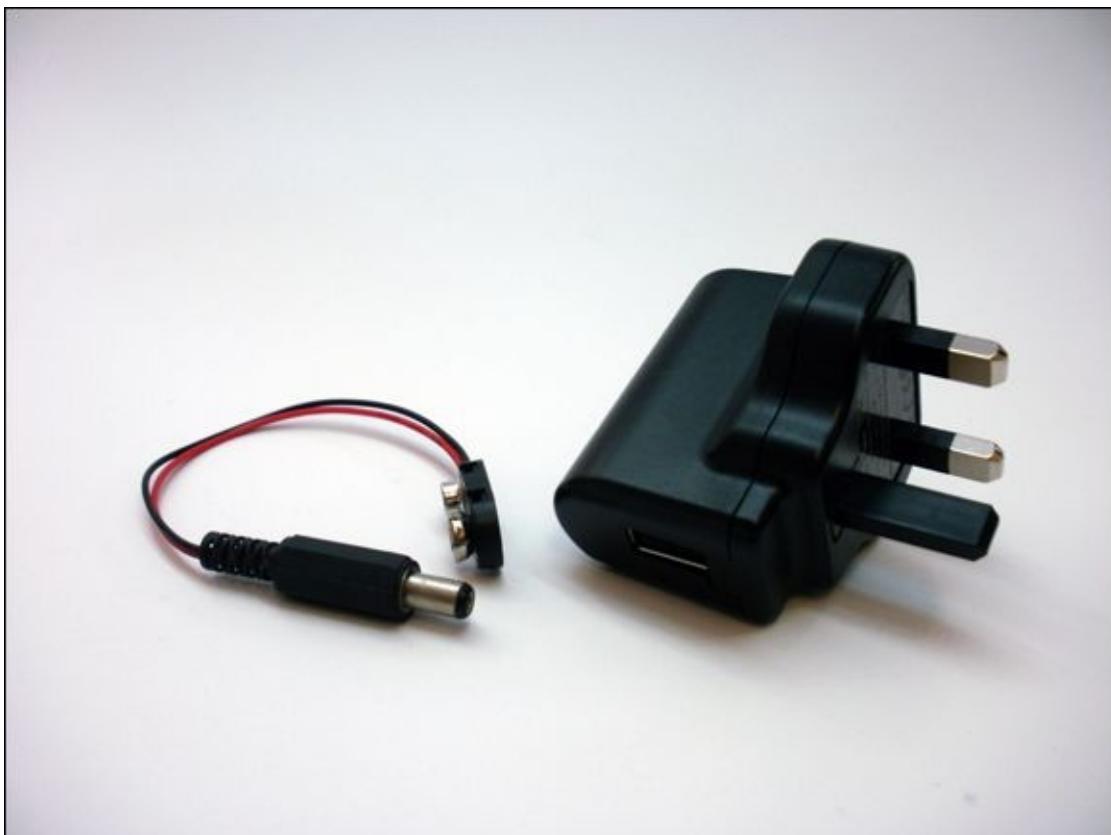
**FIGURE I-15** Enamelled (left) and plastic coated (right) wire

Solder is like a conductive glue for electronics. It sometimes comes on spools in different thicknesses like in [Figure I-16](#). The projects in this book don't require very sophisticated soldering, so you don't have to worry about which thickness to buy. Just about any thickness will work okay. The only important thing to watch out for is to make sure you buy solder for electronics—don't buy solder that's used for plumbing!



## **FIGURE I-16** Solder on spools

After you build your projects, you might want to run them without having to connect them to your computer for power. If so, you can either power your project from a power supply or from a battery. If you use a power supply, it's easiest to buy a USB wall adapter—a power supply that lets you connect a USB cable to a wall socket, with the other end of the USB cable plugged into your Arduino board. If you'd like to use a battery, the best option is to get a 9V battery connector with a DC barrel on the end. There is a black plug socket on your Arduino board where you can plug in the connector. Both options are shown in [Figure I-17](#). For the Lilypad Arduino USB, you can use a LiPo battery, but you read more about that in [Adventure 8](#).



## **FIGURE I-17** A 9V battery-to-DC-barrel connector (left) and a UK plug for a USB cable (right)

When you work with soft circuits in [Adventure 8](#), you need alligator clips like the ones in [Figure I-18](#), which you use instead of jumper wires.



**FIGURE I-18** Alligator clips

In [Adventure 8](#), you also use conductive thread, which is thread spun with conductive fibres. There are different kinds available from different manufacturers, but all the options are a silver colour as shown in [Figure I-19](#).



**FIGURE I-19** Conductive thread

# Tools You Will Need

Just as you need hammers and saws to build something with wood, you need special tools to work with electronics.

When you test your circuits you use a breadboard, but you eventually need to go beyond the breadboard. For example, you might need to add longer wires to a component so it fits inside your housing, or you might want to connect components together in a more permanent way that won't fall apart.

The first thing you need is a soldering iron. Solder is like glue for electronics, but it only works at high temperatures (think of it as a hot glue for electronics). A soldering iron is a tool that gets very hot (much hotter than an oven) so that it can melt solder. Only use a soldering iron when an adult is nearby to help you.

A soldering iron may be a single hand-held tool that plugs into the wall, like the one in [Figure I-20](#). Or it may plug into a box with a temperature dial that plugs into the wall. Either kind is okay. The important thing is to buy one that is meant for small electronics and not plumbing or any other activity.



[\*\*FIGURE I-20\*\*](#) A soldering iron

Wire often comes with a plastic coating that is an insulator that doesn't conduct electricity. You sometimes need to remove this plastic coating from the ends of the wire so you can fit it into a breadboard or solder a component to it. You could always carefully use a knife or cutters to try and remove the plastic, but that can be a very frustrating method. It is well worth buying the right tool for the job. Enter the wire stripper!

Wire strippers come in lots of shapes and sizes, as you can see in [Figure I-21](#). Choose whichever one you like best.



**FIGURE I-21** Different kinds of wire stripper

Wire cutters do what you expect—they cut wires. Be sure to get smaller ones that easily fit in your hand as you will be working with small components and thin wires. [Figure I-22](#) shows the kind of wire cutters you could get.



**FIGURE I-22** Wire cutters

Pliers help you shape and bend wires. They come in different sizes and shapes, but a smaller general purpose pair of pliers is all you need for the projects in this book. Either of the pairs in [Figure I-23](#) would work well.



**[FIGURE I-23](#)** Pairs of pliers

The next tool may seem a bit daunting, but it can be your best friend when working with electronics. It's the multimeter! It measures multiple things (that's how it got its name), with resistance and voltage being the most useful to the beginner. They range from very cheap to extremely expensive. When you are choosing one for yourself, you don't need to spend a lot of money, especially if it is your first multimeter. You probably want one that auto-ranges, though that's not essential, but you definitely need one with a continuity test. (When you look at multimeters in a store, auto-ranging and continuity testing will be listed in their features.) Auto-ranging means that you don't need to know the approximate value of whatever you are testing before you test it. A continuity test is when the multimeter beeps when an electrical connection is made between the probes. [Figure I-24](#) shows a less expensive multimeter, which isn't auto-ranging but does have a continuity test.



**FIGURE I-24** A multimeter

The final tools are not specifically used with electronics but are essential for constructing the housing for your projects: scissors and a utility knife ([Figure I-25](#)). Always take care when using either!



**FIGURE I-25** A pair of scissors and a utility knife

# Software You Will Need

When we talk about Arduino, it is easy to think about the board and nothing else. After all, that's the part you physically place into your project. However, the Arduino needs code in order to do anything. You write that code on another computer first and then upload the code to the Arduino board.

The company that makes the Arduino board also makes the software that helps you write and upload the code. It's free to download from <http://arduino.cc/en/Main/Software>.

[Adventure 1](#) takes you through the steps to setting up the software on your computer.

The circuit schematics and diagrams of circuits on breadboards in this book are made with a program called Fritzing, which is also free online at <http://fritzing.org/download>. You can even use Fritzing to start designing your own projects!

# Other Useful Materials

Writing code and building a circuit is only one half of completing a project. Your project doesn't come alive until it is surrounded by some kind of housing. Whether it's a game or an interactive light, when it is just a circuit on a breadboard it hasn't yet reached its full potential.

So, to make the projects in this book, you use many low-tech techniques alongside your newly acquired high-tech skills. Scissors, paper and glue form the basis of many of your projects. It's good to have the following items to hand, but it's never a bad decision to add decorative items like glitter that allow you to let your imagination run riot! In particular, you need the following things:

- Small cardboard boxes or shoeboxes
- Card, cardboard and paper
- Paint for decorating
- String or yarn
- White craft glue, glue-stick or a hot glue gun
- Paintbrush
- A balloon (for [Adventure 6](#))
- A marble (for [Adventure 9](#))

# **What I Assume You Already Know**

Because you've started reading this book, I'm going to assume you're already interested in technology! You don't need to have done any computer programming previously or built any circuits (that's what this book is explaining!), but I do assume that you have used a computer before.

You need a computer to work with the Arduino but it doesn't really matter what operating system your computer uses—Mac OS X, Windows or many different Linux distributions (see <http://playground.arduino.cc/Learning/Linux> for guidance on which Linux distributions you can use). I assume that you are comfortable going online and downloading files and that you know how to find and open applications on your computer.

You may need an administrator password to install some of the software, so if you don't know the password for your computer it will help if someone who does know the password is nearby when you install it.

# How This Book Is Organised

This book guides you through programming your Arduino board and constructing circuits in nine adventures. Each adventure starts by introducing the new skill you need in order to complete the standalone project at the end of the adventure. The new skill might be learning how to use a new component such as a motor, for example, or how to do something clever in code.

The most important adventure for you to start with is [Adventure 1](#). It helps you install the software needed to upload your code onto your Arduino board. After all, if you don't do that, the rest of the adventures won't be much fun!

[Adventures 2 to 9](#) build on previous adventures, so I recommend that you follow the adventures in order. Of course, if you prefer you can throw caution to the wind and do them in any order you want. You can always look up more guidance on a particular topic from an earlier adventure if you come across something you don't know.

As well as helping you set up your computer so you can program your Arduino board, [Adventure 1](#) also guides you through your first Arduino program, called a sketch. You even build your first circuit on a breadboard and control an LED.

In [Adventure 2](#) you learn how to control more than one LED, how to print messages from your Arduino board to your computer and how to use your first sensor—a potentiometer. You get to put your new skills into practice by building a status message sign that lights up to show that you don't want to be disturbed.

In [Adventure 3](#) you are introduced to your second sensor: a push button. You combine it with potentiometers to control a motor. You also learn how to use loops in computer code to repeat the same thing over and over again. Putting it all together, you build a combination safe that opens only when the correct combination is dialed. The “safe” is only a cardboard box, so it's probably not fit for storing the family jewels, but it's good enough to protect your favourite sweets.

[Adventure 4](#) shows you how to break up your code into bite-sized pieces using functions. You then use functions to control multiple LEDs using special chips called shift registers. In the final project of the adventure, you make letters with embedded LEDs in the style of old carnival signs.

[Adventure 5](#) adds sound to the growing list of actions you can control with your Arduino. You find out how to make lists in code to play short tunes over a new component—a piezo. You then make an augmented wind chime that puts an electronic twist on a traditional instrument.

[Adventure 6](#) introduces more subtle controls by showing how you can fade an LED and not just turn it on and off. You also expand the abilities of your Arduino by installing new libraries that don't come with the Arduino software. You then combine your new skills with a three-colour LED to create a crystal ball that magically changes colour!

In [Adventure 7](#) you are introduced to a new Arduino board, the Arduino Leonardo. You

master one of the exciting features of the Leonardo: making a computer think the Arduino is a keyboard. You add a new sensor that detects light and make a computer game controller that lets you play a game with a wave of your hand.

In [Adventure 8](#) you get to work with another Arduino board and build circuits using a needle and thread instead of wire and a soldering iron. This adventure helps you become a master of arrays and create a hoodie that displays a secret message.

And finally, [Adventure 9](#) is the big adventure! You have to chance to put together all the skills you've gained over the earlier adventures to create a marble maze game that automatically keeps track of your score, counts down the remaining time and plays sound effects. You use a familiar component in a new way, using a piezo to detect vibrations as well as play sound effects.

Appendices A and B prepare you for further adventures beyond this book. Appendix A points you toward other Arduino resources in print and online, and Appendix B shows you where to buy tools and components for your projects.

# Conventions

Throughout this book there are boxes to help you out:



These boxes explain concepts or terms you might not be familiar with.



These boxes give you hints to make your coding and building easier.



These boxes contain important warnings to keep you and your computer safe when completing a step or project.



These boxes feature quick quizzes for you to test your understanding or make you think more about a topic.



In these boxes I explain things or give you extra information I think you'll find useful.



These boxes point you to videos on the companion website that take you through the steps.

You will also find two types of sidebar in the book. The *Challenge* sidebars ask you how might expand on the new skills you are learning or add new features to your projects. The *Digging into the Code* sidebars go deeper into the programming concepts used in Arduino programming.

When you are following the instructions in the book, you should type in the code exactly as you see it—every ; is very important! However, the spaces between words don't matter. Spaces are used to make the code easier to read, but it doesn't matter to the Arduino.

For example, both of the following lines mean the same thing:

```
if(i<4)  
if ( i < 4)
```

Your sketch could be written as a single very long line of text and it would still run on the Arduino! But it would be very difficult for another programmer to understand what is

happening. Adding spaces and notes to explain what is happening in the code is the best way to program.

Sometimes a line of code is too long to fit on one line of this book. If you see the symbol ↪ at the end of a line of code, that means that line and the next line should be typed as a single line of code in your Arduino software.

For example, the following line should be typed on one line, not two:

```
Serial.println( "Hello, from the setup() function in ↪  
your Arduino Uno!" );
```

To help you keep track of the new coding concepts you learn, there is a Quick Reference Table at the end of each adventure, which lists any new functions, data types or other programming commands that have been introduced in that adventure.

When you complete an adventure, you unlock an achievement and collect a new badge. You can collect badges to represent these achievements from the *Adventures in Arduino* website ([www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino) ).

# The Companion Website

Throughout the book, you'll find references to the *Adventures in Arduino* companion website, [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino). Here, you'll find tutorial videos to help you through the physical making of your projects along with the code used. It can be very frustrating to track down a mistake after you've typed in code from a book by hand. The important thing is understanding what the code is doing and not just how to type it all out yourself (or at least not when you are first starting out)!

# Reaching Out

You'll find a lot of tips in Appendix A about where to go for help but the first place you should always look is the Arduino website ([www.arduino.cc](http://www.arduino.cc)). It has lots of useful information in the Learning section, and you can always ask questions in the Forum.

You can also contact me by sending me a message through the website  
[www.adventuresinarduinobook.com](http://www.adventuresinarduinobook.com).

Time to start your adventures!



**YOU WILL SOON** be creating exciting projects that bridge the physical and digital worlds! You'll learn how to write code that triggers sound, controls motors and flashes lights. The Arduino is the perfect tool for combining circuits and code!

You will use the same three steps for each project you build with your Arduino:

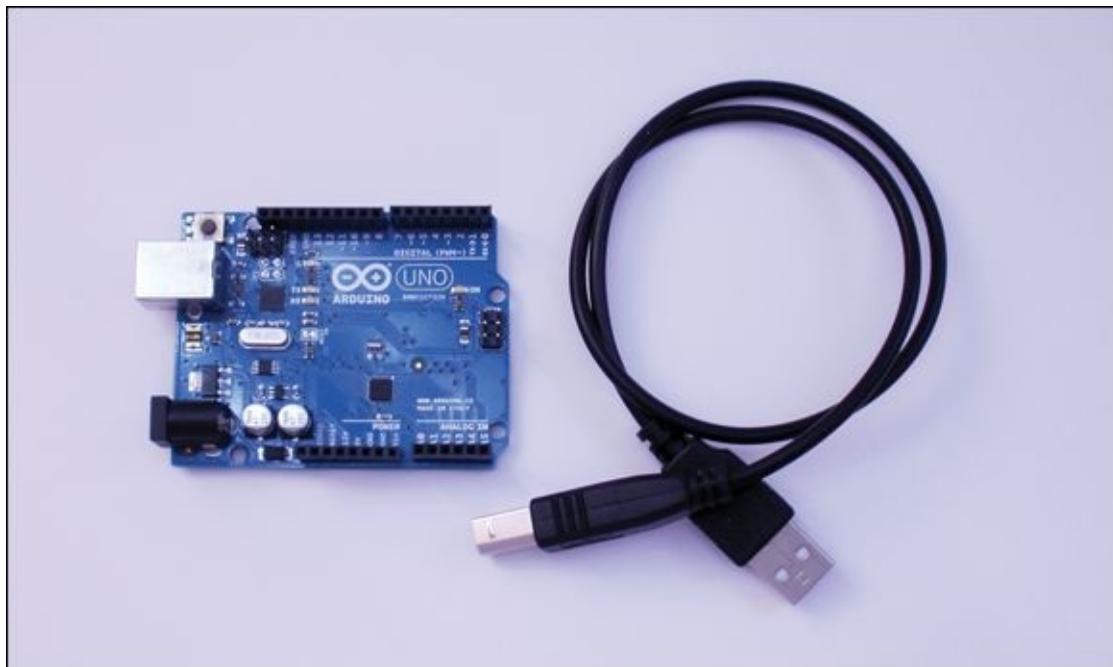
1. Write the code that tells the Arduino Uno what to do on your computer using the Arduino software.
2. Connect your Arduino Uno to your computer, and upload your code onto the board.
3. Build and connect your circuit to your Arduino Uno.

But first things first. Before you can do anything else, you need to download and install the Arduino software and set up your computer to program your Arduino Uno. That's what you will be doing in your first adventure. Then, when you've got everything working as it should, you're going to start your first Arduino project—controlling when a light turns on and off.

# What You Need

To get started, you need the following things. [Figure 1-1](#) shows the electronic components you will need.

- A computer
- An Arduino Uno
- A USB cable
- An Internet connection so you can download the Arduino software



[FIGURE 1-1](#) An Arduino Uno and USB cable

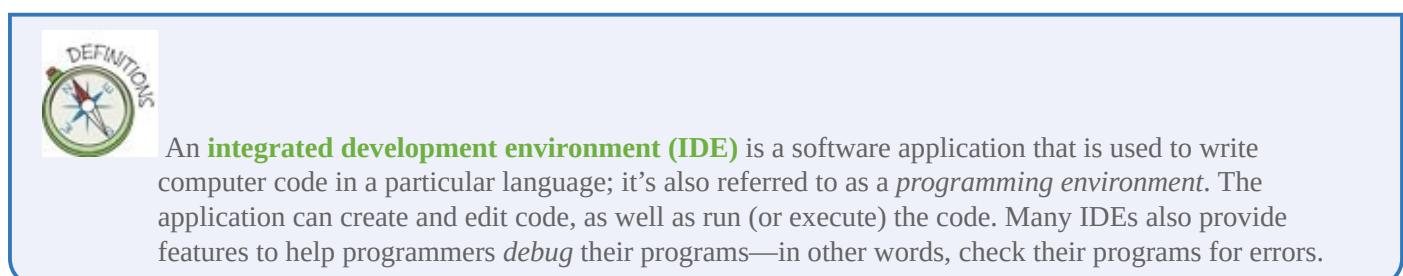
# Downloading and Installing the Arduino Software on Your Computer

In order to run Arduino programs, in addition to an Arduino Uno you need a computer and some special software to make the Arduino work. You will be writing the code that runs on the Arduino Uno on another computer first, and will then upload it to the board. Sounds complicated, doesn't it? Don't worry; you'll be guided through the process step by step. And it's not as difficult as it sounds.

You're going to use a piece of software to write the code and then upload it. This piece of software is called the Arduino environment, or **integrated development environment (IDE)**. It is available for free from the Arduino website at <http://arduino.cc/en/Main/Software> (see [Figure 1-2](#)).



[FIGURE 1-2](#) You can download the Arduino IDE for your computer from the Arduino website.



You are now going to download and install the latest version of the Arduino software designed for your particular computer's operating system, using the steps outlined here. When this book was written, 1.0.6 was the current version of the software. You can see what the current version is by visiting <http://arduino.cc/en/Main/Software>. You can

find the current version towards the top of the page. After you have installed the software, you can see what version you are using by reading the title bar of the window that appears when you launch the Arduino IDE.

The software can run on Windows, Mac or Linux computers, but depending on your computer's operating system (OS), you may need to install both the IDE and another piece of software, called a **driver**. This adventure describes what you need to do, but you can also visit <http://arduino.cc/en/Guide/HomePage>, which has lots of guidance on how to get the software installed.



Have your Arduino Uno and USB cable to hand as you install the software, because you might need them for some of the installation steps, depending on your computer's operating system.



A **driver** is a piece of software that lets your computer communicate with an external device, such as a printer or a keyboard.

# Installing Arduino Software on a Mac

It is quite simple to install the software on a Mac. You don't need to install a driver, only the Arduino IDE. Just follow these steps:

1. In your Internet browser, open the Arduino download page at <http://arduino.cc/en/Main/Software>.
2. Select the Mac OS X zip file from the list of current Arduino downloads.
3. Find the file called `arduino-1.0.6-macosx.zip` and unzip it by double-clicking it. Now move the `Arduino.app` file into your Applications folder.
4. After you've installed the software, plug one end of your USB cable into your Arduino board (shown in [Figure 1-3](#)) and the other end into your computer. A message about a new network device may appear on screen. If that happens, you can just cancel or close the message window.



**FIGURE 1-3** Plug the USB cable into the Arduino Uno and then connect it to your computer.



If you have any problems, visit <http://arduino.cc/en/Guide/MacOSX> for more help.



To see a video of how to install the Arduino IDE on a computer running Mac OS X, visit the companion

site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

# Installing Arduino Software on a Windows PC

You need to follow a number of steps to install the Arduino software on Windows 7, Vista and XP (see the Tips and Tricks box for Windows 8). You will be installing two things: the software and the driver.

1. In your Internet browser, open the Arduino download page at <http://arduino.cc/en/Main/Software>.
2. Start by downloading the file called **Windows ZIP file**.
3. Find the downloaded zip file and unzip it to the **Program Files** folder. It should contain multiple folders and files, including a folder called **drivers** and a file called **arduino.exe**. If you would like, you can right-click **arduino.exe** and create a shortcut to place on your Desktop.
4. Plug one end of the USB cable into your Arduino board and the other end into your computer as shown in [Figure 1-3](#). You should see lights illuminate on your Arduino Uno. This just means it has power.
5. Your computer will now start to look for a driver to use with the Arduino board. Your computer may find the driver on its own, but, depending on the version of Windows your computer is running, you may need to follow different steps to finish installing the driver. You may need to know an administrator password for the computer and might need some help with someone with more computer experience. You can always visit <http://arduino.cc/en/Guide/Windows> for more detailed instructions.
6. Click the Start menu and open the Control Panel.
7. While in the Control Panel, navigate to System and Security. Click System, and when the System window appears, open the Device Manager.
8. Inside the Device Manager look under Ports (COM & LPT). You should see a port named **Arduino UNO (COMxx)**. If you don't see a COM & LPT section, look under Other Devices for Unknown Device.
9. Right-click Arduino UNO (COMxx) port and choose the Update Driver Software option.
10. If you are running Windows XP or the Hardware Update Wizard appears, go to <http://arduino.cc/en/Guide/UnoDriversWindowsXP> and follow the screenshots to install the drivers located in the folder you downloaded and unzipped.
11. If the Hardware Update Wizard doesn't appear and instead you see a window with the options Search Automatically for Updated Driver Software and Browse My Computer for Driver Software appears, click Browse My Computer for Driver Software.
12. Navigate to the folder you downloaded and unzipped. Go to the folder inside it called **drivers** and select **arduino.inf**.

Wow, that was complicated. Luckily, you don't have to do that again. You only need to do it once. The steps might change when new versions of the Arduino IDE are released or if

there are updates to your operating system. You can always go to the Arduino forum board for installation problems at <http://forum.arduino.cc/index.php?board=2.0>. You can look over the questions asked by others and even ask your own if you can't find a question similar to the problems you are having.



If you run into any problems, first visit <http://arduino.cc/en/Guide/Windows> for more help.

You can also visit <https://learn.adafruit.com/lesson-0-getting-started/installing-arduino-windows>, [www.dummies.com/how-to/content/how-to-install-arduino-for-windows.html](http://www.dummies.com/how-to/content/how-to-install-arduino-for-windows.html), or <https://learn.sparkfun.com/tutorials/installing-arduino-ide/windows> for even more tips including extra guidance for installing on Windows 8.

If your computer runs Windows XP, you can follow along with the screenshots at <http://arduino.cc/en/Guide/UnoDriversWindowsXP> to install the drivers.

# Installing Arduino Software on a Linux Machine

If your computer runs Linux (if you are using a Raspberry Pi, for example), you should first visit the online documentation for Linux and Arduino at

<http://playground.arduino.cc/Learning/Linux>. There are many different types of Linux, so I haven't listed them all here, and if your computer runs a Linux distribution, you likely already know how to install new software. It is probably easiest to install the Arduino environment by using a package manager. Here's how to do the installation on a Debian variant of Linux, such as Raspbian on Raspberry Pi. On a command line, enter the following command:

```
sudo apt-get install arduino
```

This command downloads and installs the software. Alternatively, you can download the 32- and 64-bit applications directly from the Arduino download page at

<http://arduino.cc/en/Main/Software>. You don't even need to worry about compiling source code, but don't forget to read the guidelines for your distribution at

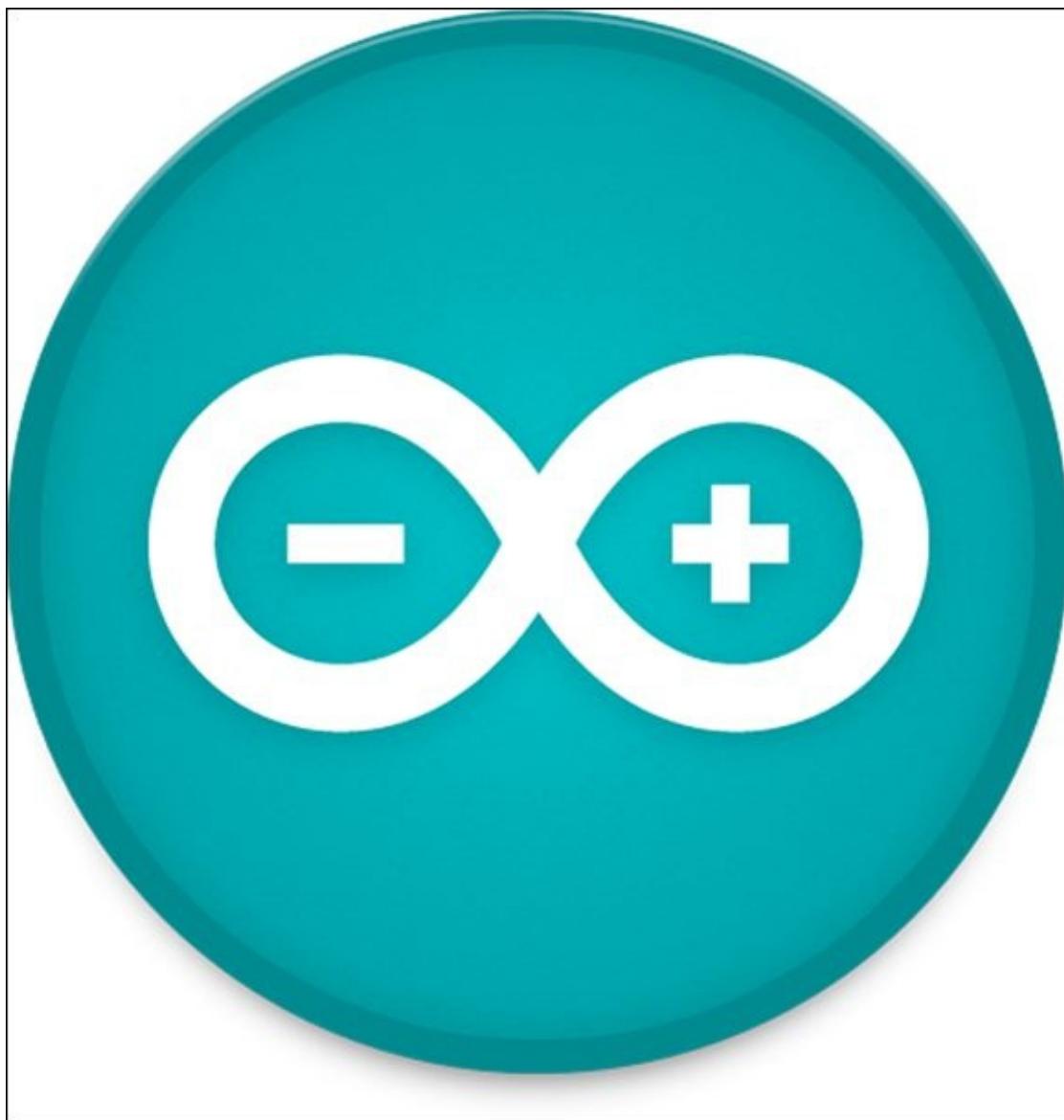
<http://playground.arduino.cc/Learning/Linux>.



If you are using a Raspberry Pi or other Linux-based system, I am assuming that you know how to use the command-line interface to install software for whatever version of Linux you are running. If you need more information or a refresher on using the Raspberry Pi, check out *Adventures in Raspberry Pi* by Carrie Anne Philbin (John Wiley & Sons, Inc., 2014).

## Exploring the Arduino IDE

Well done! Now that you have the software installed, you're ready to start using the Arduino IDE! It's time to launch the software. If your computer is a Mac, go to Applications and double-click Arduino. If your computer is running Windows, either double-click the shortcut on the Desktop (if you made one as described earlier in the adventure), or go to the folder you downloaded earlier and double-click the Arduino application. If your computer is running Linux, start the Arduino IDE either from the command line or by double-clicking the program icon. The Arduino program icon (see [Figure 1-4](#)) looks the same on Mac, Windows and Linux.



**FIGURE 1-4** The Arduino program icon

When the Arduino IDE has finished starting, a window similar to the one shown in [Figure 1-5](#) will appear.



**FIGURE 1-5** The important parts of the Arduino IDE

First, you’re going to find out about some of the basic functions of the Arduino IDE. There are six buttons at the top of the Arduino window (see [Figure 1-5](#)), and most of them are easy to figure out. The middle three buttons are New, Open and Save. Clicking the New button (surprise!) starts a new Arduino file, the Open button opens an existing Arduino file, and the Save button saves the current the file. You’ll notice there’s a button on its own on the far right—you will be finding out about this in [Adventure 2](#). That just leaves the two buttons on the left.

The button that looks like a tick is the Verify button. When you click this button, the Arduino **compiles** the code; in other words, it takes the code you have given it and translates it into something the Arduino board can understand. There is a second thing this button does. If there are parts of the code that Arduino’s compiler doesn’t understand—perhaps a semicolon was missed out or a typo was made when the code was typed—it prints out the error at the bottom of the Arduino IDE window. It tries to be helpful with what it prints, but sometimes it doesn’t make sense! Later this adventure goes over some of the common error messages and what they might mean.



**Compiling** code is the process of taking code written by a human and turning it into instructions that can be understood by a machine.

Don’t worry too much about how this happens yet. All you need to know for now is that the Verify button is used to turn your code into something the Arduino can understand, and determine that the code is free from simple errors. After you have done that, you click the final button, which is the Upload button. This compiles and uploads the code to the Arduino Uno. This is the button you will probably use most, as it puts your code onto the Arduino Uno. The Verify button can be useful if you are writing code and don’t have your Arduino Uno nearby because it means you can at least check if you have any simple errors in your code, although you won’t know if your code completely works until you can upload it onto a board.

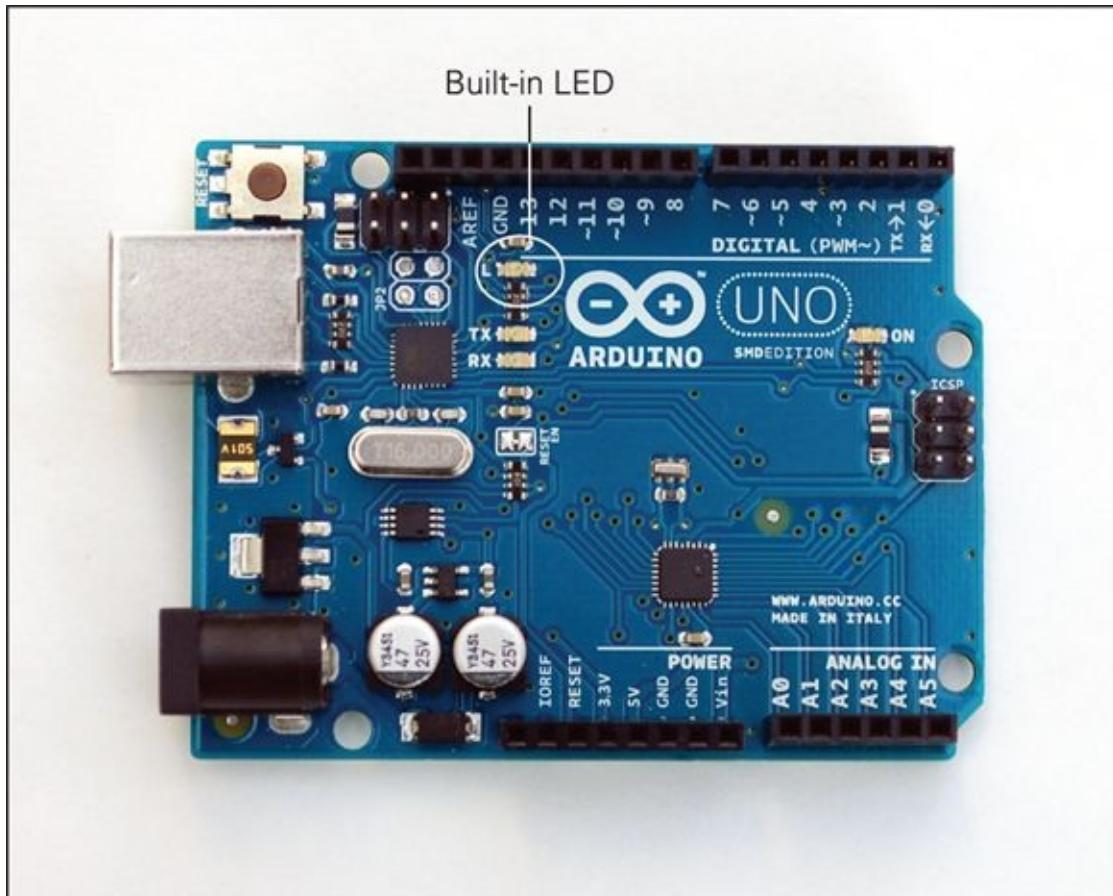
# Using Blink to Test That Everything Is Set Up Correctly

You can write computer code for a device like a laptop or Raspberry Pi and then run it on the same computer you've written it on. With Arduino, it's a bit different because an Arduino board can't program itself, so you have to write the Arduino code on a computer that can run the Arduino IDE. The IDE then takes the code that humans can write and read and translates (or **compiles**) it, turning it into code that the Arduino board understands. The IDE then copies and uploads the compiled code onto the Arduino board.

Software developers use basic programs to test that their computer is working as they would expect. These are called *Hello World* programs.

A *Hello World* program is very simple. If you were learning a new programming language, you would write a program that would just print the phrase "Hello world" to the screen, which would show you that everything was set up properly. But wait—the Arduino doesn't have a screen! After you load the code on the Arduino from your computer, it doesn't talk to the computer anymore; all the USB cable does is provide the Arduino with power (though you find out in the next adventure how to send messages to the computer through the USB cable).

What the Arduino board *can* do is blink a light. It even has one built into the board for exactly this purpose. There's a tiny yellow or orange **light-emitting diode (LED)** near the number 13 on the board (see [Figure 1-6](#)). This LED is useful because you can program it to turn on and off in order to quickly check that the Arduino board is working. You don't need anything besides your computer, a USB cable and your Arduino board.



**FIGURE 1-6** The built-in LED on the Arduino board is near the number 13.

The Arduino IDE comes with some example code to help get you started. One of these examples is the Blink program, or **sketch**, which is used as the *Hello World* program for Arduino. You're now going to use that to check everything is working as it should.



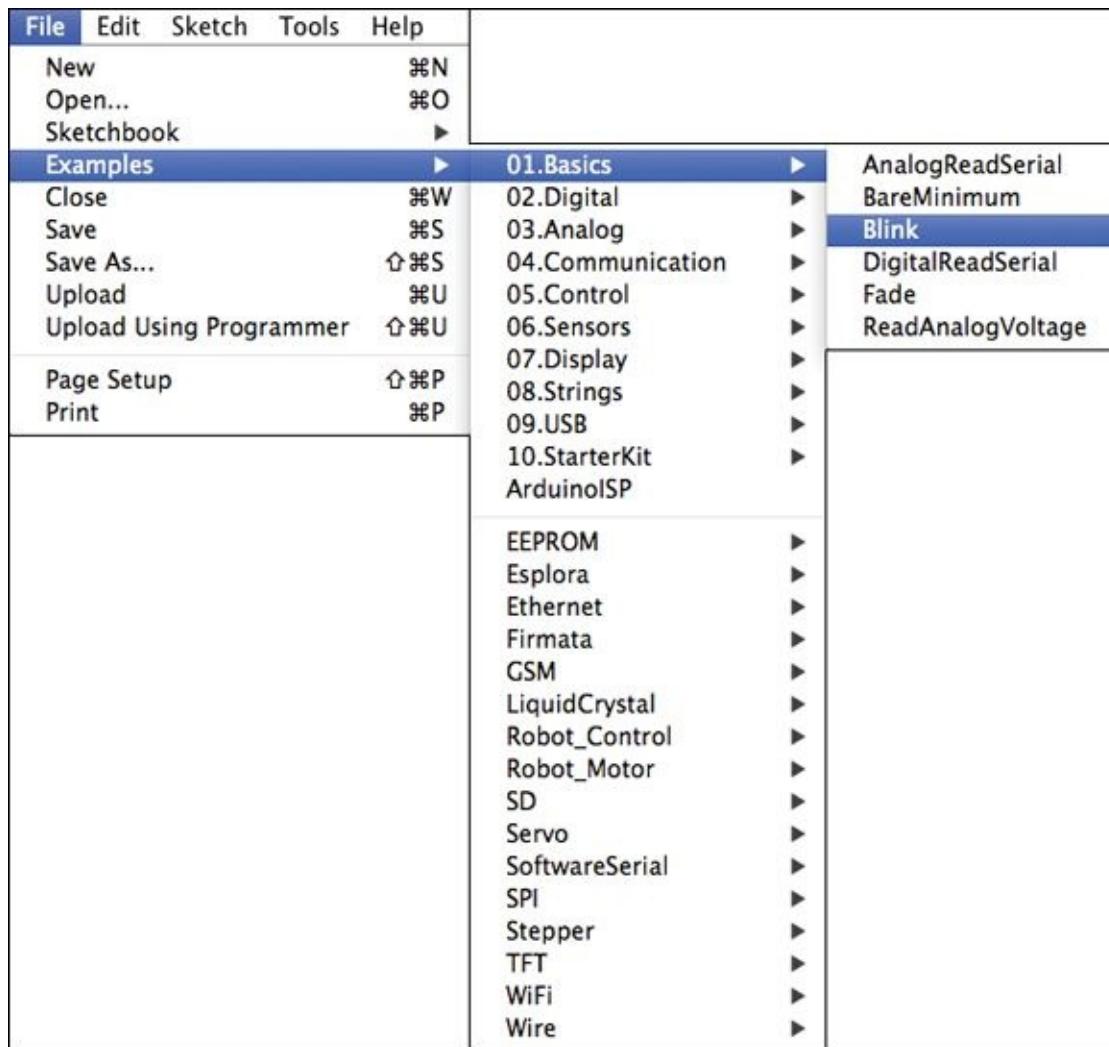
An **LED**, or **light-emitting diode**, is an electrical component that lights up when electrical current flows through it. A diode only lets electricity flow in one direction, so an LED lights up only when the long leg is connected to the positive side of a power source and the short leg is connected to the negative side. If they are switched, the LED won't light up.



Arduino programs are often referred to as **sketches**, like quick drawings artists make.

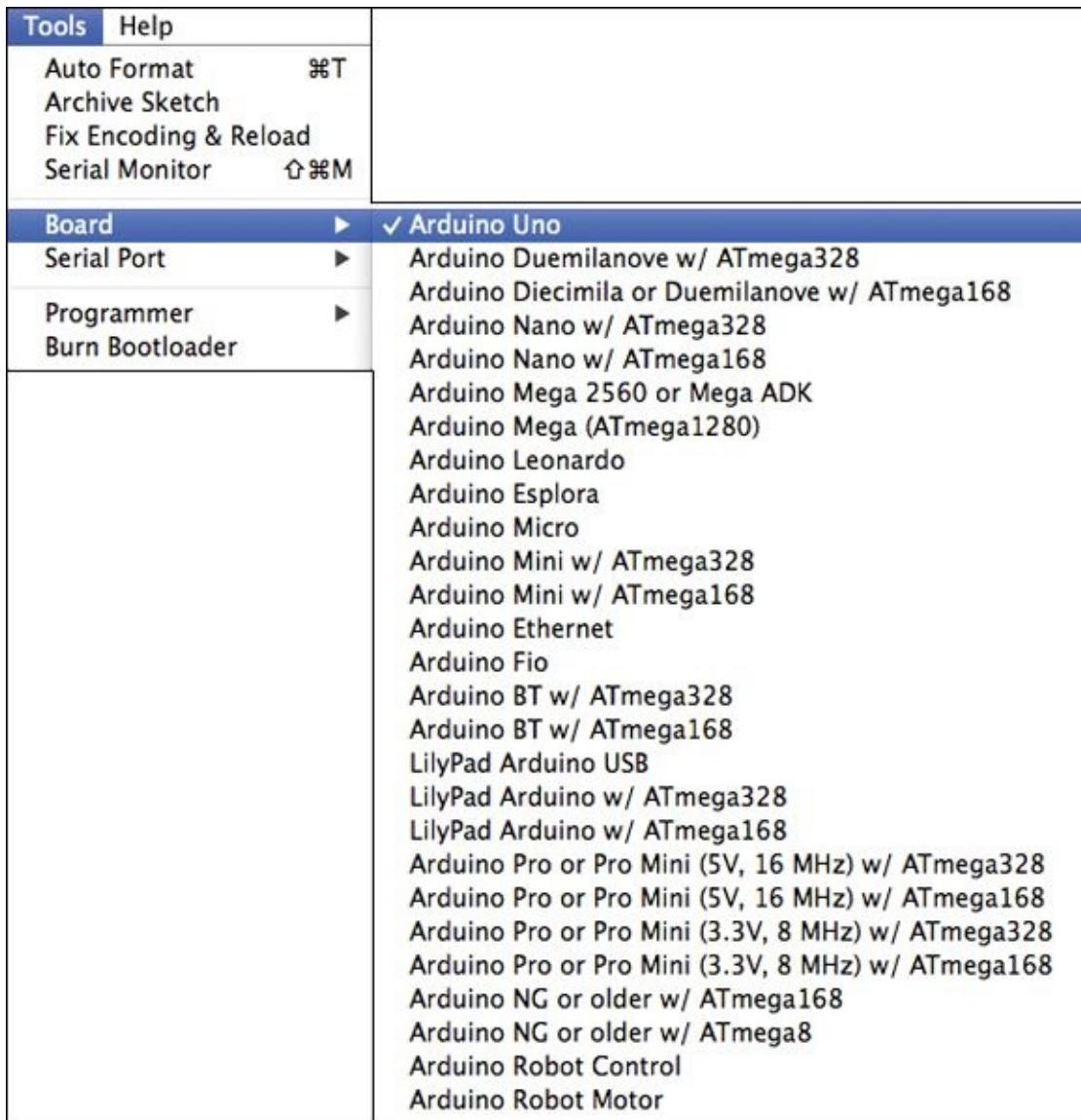
# Uploading Blink

To test that your Arduino Uno can receive messages and new programs from the computer without any problems, you need to compile and upload the Blink sketch. Select File ⇒ Examples ⇒ 01.Basics ⇒ Blink (see [Figure 1-7](#)) to open a new window with the Blink sketch.



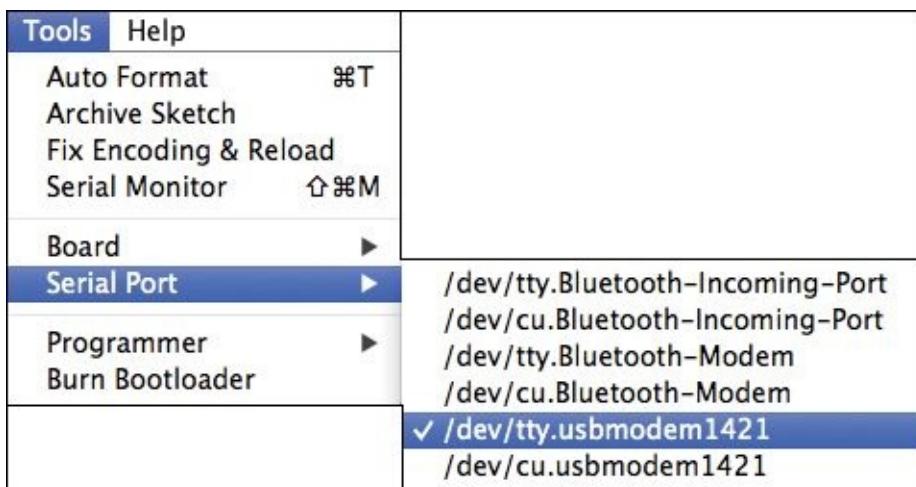
**FIGURE 1-7** Opening the Blink sketch, which is located in the examples that are included with the Arduino IDE

Before you upload your code, you first need to check two settings: the board and the port. You need to make sure these settings are correct each time you launch the IDE. After you have set them, you don't need to change them again until you quit and start the IDE again at a later time. To find the board, select Tools ⇒ Boards (see [Figure 1-8](#)). You should see a list of all the different Arduino boards. Make sure you select the board you are using. (This will be an Uno, but if you're not sure, it is written on the board itself.)



**FIGURE 1-8** Selecting the board you are using

You'll find the port under Tools ⇒ Serial Port (see [Figure 1-9](#)). Select the port you are using, the same way you found the board. Make sure you have plugged your board into your computer with the USB cable, or the port won't appear on the list.



**FIGURE 1-9** Selecting the port your Arduino board is plugged into



It may not be obvious which port you should select if there is more than one listed. You can try looking at the list when the Arduino board is not plugged in, then plugging in the board and looking again. If one appeared that wasn't there before, that is probably what you should select.

If you are using a Mac, look for the port that starts with `/dev/tty.usbmodem`. On a Windows computer, the port is just listed as a `COM` port. In Linux, the port starts with `/dev/ttyACM`. It won't hurt anything if you don't select the right port, so if you're not sure, just select a port and click the Upload button. If you get an error message, try selecting the next port in the list, and continue down the list until you find the port that your Arduino board is on.



Remember that you need to plug your USB cable connected to your Arduino board into your computer. Your port won't show up in your list of port options if the board isn't connected to the computer!

After you have selected your board and port, you are ready to upload your code—this is the Blink sketch you have already opened. To do this, simply click the Upload button. If you have forgotten to select the port, a message may pop up asking you to select one.

If you have everything set up correctly, you will see a message towards the bottom of the Arduino IDE window that says Compiling Sketch and then Uploading. If the sketch has been uploaded without any problems, you will see a Done Uploading message as shown in [Figure 1-10](#).

```
void loop() {
    digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000); // wait for a second
    digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
```

Done uploading.

Binary sketch size: 1,082 bytes (of a 32,256 byte maximum)

**FIGURE 1-10** Message in the Arduino IDE after successfully uploading your code

Look at the LED next to the number 13 on your board. Is it flashing on for one second then turning off for one second? If so, success! Well done. If not, or if any orange text has appeared at the bottom of your Arduino IDE, the next section will help you troubleshoot what might be going wrong.

You will be soon making changes to your Blink sketch and even writing new sketches of your own. It's important to remember that the code you write in the IDE has to be uploaded to the Arduino Uno. Every time you make a change to the code, upload it again to your board!

## Troubleshooting Common Problems

When something goes wrong when you're trying to upload code to an Arduino board, a message from something called avrdude might be printed at the bottom of the Arduino IDE (such as the one shown in [Figure 1-11](#)).



The screenshot shows the Arduino IDE interface. The code editor at the top contains a single line of code: `// the loop routine runs over and over again forever:`. Below the code editor is a status bar with a progress bar. The main window displays an error message from the terminal or serial monitor. The message is as follows:

```
Problem uploading to board. See http://www.arduino.cc/en/Guide/Troubleshooting#upload
Binary sketch size: 1,084 bytes (of a 32,256 byte maximum)
avrdude: stk500_recv(): programmer is not responding
```

In the bottom left corner of the main window, there is a small number '1'. In the bottom right corner, it says 'Arduino Uno on /dev/tty.usbmodem1411'.

**FIGURE 1-11** A common error when the computer can't talk with the Arduino

The Arduino board is built around a microcontroller (computer) chip made by a company called Atmel. This chip is a type of microcontroller called an AVR and the program that the Arduino IDE uses to talk to the Arduino board is called avrdude. So when you get messages from avrdude, it means something has gone wrong with the communication between the board and the computer. Usually it's that the computer is trying to use avrdude to send a new sketch to the Arduino Uno, but the computer can't find it. Problems could be caused by selecting the wrong port, but if you have tried all the ports there may be something else going wrong.

The easiest thing to try when you get an error from avrdude is to unplug the Arduino board from the USB cable (this removes the power and turns it off). Then plug it back in again. If you still have problems, try unplugging the Arduino board and then quitting the Arduino IDE like you would any other application. Launch the IDE again and connect the Arduino Uno once more to see if you can upload a new sketch.

If you've done all that and you still can't upload sketches to the Arduino board, try going through the installation process for the IDE and drivers again.



The Arduino website (<http://arduino.cc>) is a great resource with lots of tutorials. It also hosts a forum where you can post questions. You will most likely find questions posted by other people who are having the same problem as you.

# DIGGING INTO THE CODE



Hopefully you now have your code uploaded and running on your Arduino Uno. But what is the code actually doing? You know that the Arduino Uno is turning on and off the LED next to 13, but how does it know to do that?

A great way to start learning about code is by reading it before you write it. After all, you didn't learn how to write in school before you learned how to read! You can use the Blink as an introduction to code. Don't worry about understanding all the details right away—it's a lot to learn. You will be shown a bunch of new terms, but you don't need to remember what they all mean right away. You will get to spend more time understanding them in the other chapters.

If you look again at the Blink sketch on the screen, you'll see that the first section is all in grey. The Arduino IDE helps you understand what is happening in the code by changing the colour of the code according to what it does. The text that turns grey is called a **comment**. Comments are notes to the programmer to help explain what is happening in the code. The long comment at the top of the sketch explains what the sketch does:



**Comments** are notes within your code that explain what a line or section of code is intended to do. Each comment line begins with // or, if you want to write a comment that spans multiple lines, is between /\* and \*/. These **special characters** tell the computer running the program to ignore that line or lines.

```
/*
Blink
Turns on an LED on for one second, then off for one second,
repeatedly.
```

Most Arduinos have an on-board LED you can control. On the Uno and Leonardo, it is attached to digital pin 13. If you're unsure what pin the on-board LED is connected to on your Arduino model, check the documentation at <http://arduino.cc>

This example code is in the public domain.

```
modified 8 May 2014
by Scott Fitzgerald
*/
```

The rest of the lines that don't start with // are lines of code that the computer will execute. The lines starting with // are ignored by the computer and are notes to explain what the code is doing.

Every Arduino sketch has to have two **functions**: **setup()** and **loop()**. A function is a set of lines of code that have a name. The next section of code is the **setup()** function, which runs only once and is for tasks that need to happen only when the Arduino is first turned on. Whenever your Arduino Uno first starts up, it looks for the section of the sketch that is the **setup()** function, and it runs that section first. continued continued



A **function** is a set of lines of code that have a name. A function can be used over and over again. It may take some information as an input and output more information when it is finished, but not all functions need to do that.

```
// the setup function runs once when you press reset or power @@ta  
the board  
void setup() {  
    // initialize digital pin 13 as an output.  
    pinMode(13, OUTPUT);  
}
```

As you can see, there is only one instruction inside the `setup()` function—the function `pinMode()`. Every pin on the Arduino can read in information or output information, but it can't do both at the same time. The `pinMode()` function sets up whether the pin inputs or outputs by taking two **arguments**. The first is the number of the pin you're using on the Arduino board. The LED on the Arduino Uno is connected to Pin 13. The second is a special keyword, `OUTPUT`, which tells the Arduino that you want to output on Pin 13 and not read in on that pin.



An **argument** is a piece of information given to a function, which the function then uses to perform its task. The argument goes inside the brackets that follow the function name. In the following code snippet, for example, the function `delay(1000)` has the argument `1000`, which is the number of milliseconds you want the Arduino to wait before executing the next line.

The remaining code in the Blink sketch is the `loop()` function. After the Arduino Uno executes all the code in the `setup()` function, it looks for a function called `loop()`. It then executes all the code in that function. When it is done, it executes all the code in the `loop()` function again. And then again! And on and on! The `loop()` function repeats forever (or at least until the Arduino Uno no longer has power).

```
// the loop routine runs over and over again forever  
void loop() {  
    digitalWrite(led, HIGH);      // turn the LED on ↵  
    // HIGH is the voltage level  
    delay(1000);                // wait for a second  
    digitalWrite(led, LOW);       // turn the LED off ↵  
    // by making the voltage LOW  
    delay(1000);                // wait for a second  
}
```

The first line of code in the `loop()` function turns on the LED using the function `digitalWrite()`. It takes two arguments: the pin number and whether you are turning the electricity on or off. The argument for the pin number is just like what you saw with `pinMode()`. The second argument is a keyword: `HIGH` or `LOW`. `HIGH` turns on the electricity, and `LOW` turns it off.

The last piece of code that you haven't looked at is the `delay()` function. The Arduino board runs this code very fast—millions of times a second. That's so fast that you wouldn't be able to see the LED turn on and off. So you need to make the Arduino pause so that you can see the light blink. The delay function makes the Arduino wait for the number of milliseconds typed as the argument. In this example the delay is 1000 milliseconds, which is equal to 1 second. Here's a review of what the whole sketch does:

1. The `setup` function uses the `pinMode` function to set the pin the LED is on to be an output.
2. In the `loop()` function, the electricity on the LED's pin is turned on.
3. The Arduino is paused for 1000 milliseconds.
4. The electricity on the LED's pin is turned off.
5. The Arduino is paused for 1000 milliseconds.
6. The `loop()` function starts over again.

One last detail you might have noticed is that code has some strange punctuation. Most of the lines end with a

semicolon (`;`). This is like a full stop at the end of a sentence. A full stop indicates when a sentence is finished—it keeps writing neat and tidy. A semicolon does the same thing for a computer. It helps the computer separate different lines of code.

You may have also noticed that the code has spaces and indentation. These are like comments—they make the code easier to understand for the programmer. The computer just ignores them. The indentation makes it easier to see what lines of code belong to a function. You might have noticed that the lines of code that belong to `loop()` are all between `{` and `}` and are indented. The `{` and `}` tell the computer that those are the lines that belong to `loop()`, whereas the indentation makes that more visually obvious to reader.

The spaces in between parts of code are also only for the programmer; the computer ignores them. For example, `delay(1000)` and `delay( 1000 )` are the same to the computer, but the spaces can make it a little easier for some people to read the code. You can decide how you prefer to write your code in the following chapters!

# CHALLENGE



Try changing how long the LED turns on and off by changing the arguments in the `delay()` functions. Remember you need to upload your code after each time you change it in order for the sketch with the new changes to be on the Arduino Uno.

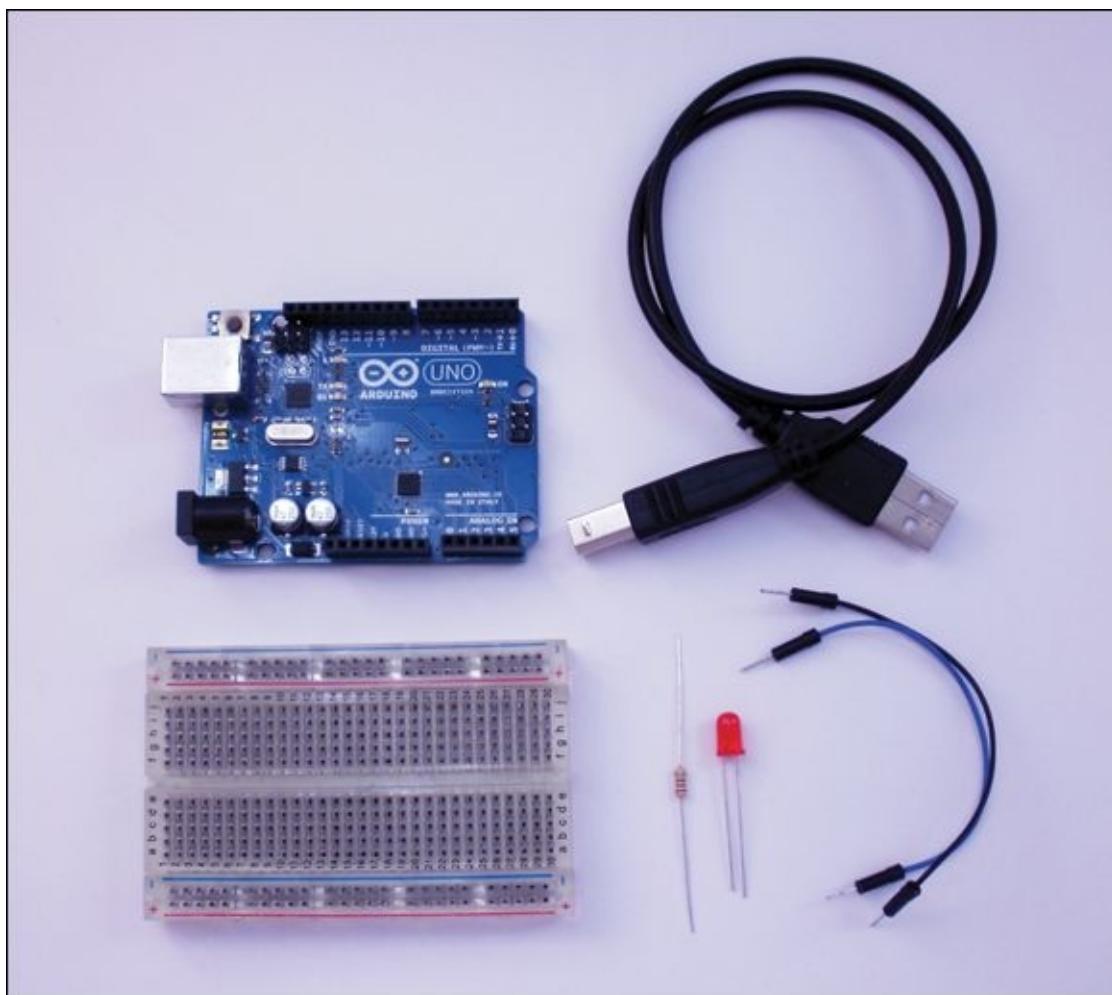
# **Building an LED Circuit**

Now that you have an LED blinking on the Arduino Uno, you are ready to go beyond the Uno's board and build your first circuit! You will use first become familiar with the tools you will use to build and test new circuits: circuit schematics and breadboards. They are the keys to creating your own projects that you can share along with being able to build projects designed by other. In the next chapter you will go a step further and build a housing for your circuit, but it's a good idea to first become comfortable with how your code and circuit come together on the Arduino Uno.

## What You Need

You need the following things to build your LED circuit. [Figure 1-12](#) shows the electronic components you need.

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 1  $220\Omega$  resistor
- 1 LED
- 2 jumper wires

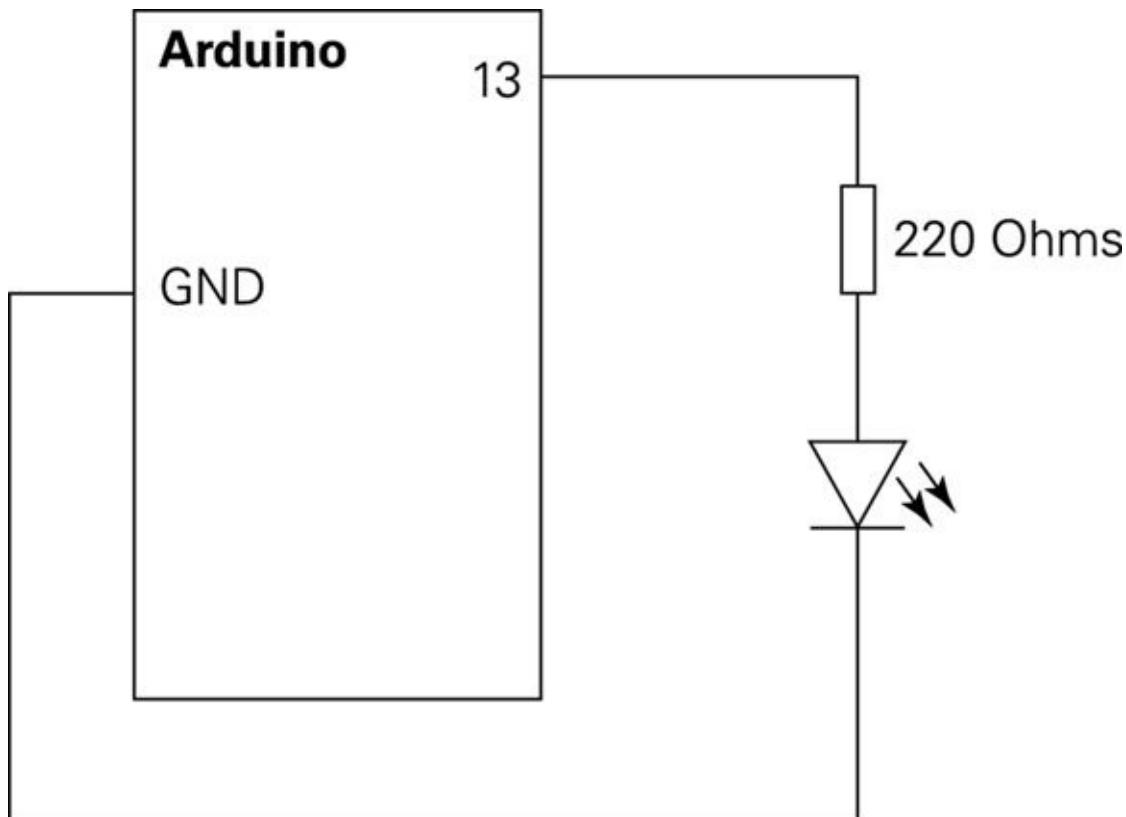


[FIGURE 1-12](#) The electronic components you need to build the circuit

# Understanding Circuit Schematics

Electricity is the flow of electrical charge. You've seen it in nature through lightning or static electricity that occurs when you walk across a carpeted floor and then touch a door handle. You also use circuits every day to control how electricity is allowed to flow. You turn on and off the lights in a room with a light switch. You can turn on a TV and change the channel. This is all done by using circuits to control electricity. You're not ready yet to build the kind of circuits that are inside a TV, but you can build a circuit that turns on and off lights!

Even a simple circuit can be built in many different ways. For example, LEDs come in different sizes and colours. You could power an LED from an Arduino Uno or with a battery. A circuit schematic is simply a diagram showing the important information about a circuit, using symbols. [Figure 1-13](#) is a circuit schematic showing the circuit that you are now going to build. There are three symbols in the schematic each representing the Arduino Uno, the resistor (more about what that is a little later) and the LED (the triangular symbol). You could build the circuit using a large red LED or a small green LED; it's your choice. The important information is that you are connecting that LED to an Arduino Uno and a resistor.



[FIGURE 1-13](#) The circuit schematic for the LED circuit

Electricity can be described and measured in different ways. Because it's invisible, it can be hard to imagine how electricity works, so water is often used as an analogy. The flow of electricity in a wire is like water in a pipe. The water flow moving through the pipe is similar to the electrical current (measured in amps, which is abbreviated A), and the water pressure is like the electrical voltage (measured in volts, which is abbreviated V). The size

of the pipe in combination with how much water is being moved through it affects the water pressure. A smaller pipe creates more water pressure than a larger pipe when the same amount of water is passed through both. The size of the pipe describes the third property used to describe electricity: resistance (measured in Ohms represented and represented with the symbol  $\Omega$ ).

That may seem complicated and difficult to understand, but don't worry about grasping all the details now. By working with electricity and building circuits, you'll figure out how voltage, current and resistance are related. Back to the circuit schematic!



The study of electricity and circuits is called circuit theory. If you are interested in learning more about circuit theory, there are great tutorials online to get you started. I really like Sparkfun's. They have animations to help illustrate the concepts! Start with <https://learn.sparkfun.com/tutorials/what-is-electricity> and <https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law>.

In the schematic in [Figure 1-13](#), the circuit is connected to Pin 13 and GND on the Arduino. Pin 13 is where the electricity that lights up the LED flows from. GND stands for ground or 0V. When you use a battery to power a circuit, the battery has two terminals: positive and negative. Electricity flows from the positive to the negative. Pin 13 and GND on the Arduino Uno play the same role as the positive and negative terminals of the battery. Electricity flows from Pin 13 to GND when they are connected in a circuit.

The circuit has two components besides the Arduino Uno: a resistor and an LED. The pins on the Arduino Uno all output 5V, which is too much for the LED. Remember the water analogy? Think of the LED like a drinking straw. It's not as strong as a pipe, so forcing too much water into it can cause the straw to burst. A **resistor** helps to control how much current can flow through the LED. In this kind of circuit, the resistor limits the amount of current, so is known as a current-limiting resistor.



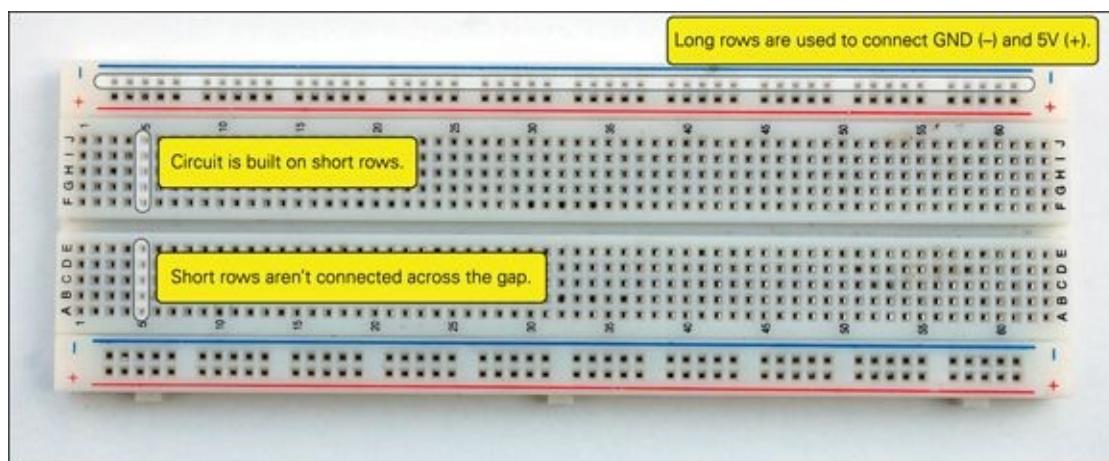
A **resistor** is an electrical component that resists current in a circuit. For example, LEDs can be damaged by too much current, but if you add a resistor with the correct value to the LED circuit to limit the amount of current, the LED is protected. Resistance is measured in Ohms (represented by  $\Omega$ ). You need to pick a resistor with the correct value to limit the current through a circuit; the value of a resistor is shown by coloured bands that are read from left to right.

# Using a Breadboard

After you know what circuit you are building, you need to use something called a **breadboard** to help you build it. A breadboard is a board with rows of holes on it (see [Figure 1-14](#)). If you could peek inside the plastic case of the breadboard, you would see that the holes in each row touch the same piece of metal. Sticking two wires in the same row means they are touching the same piece of metal, and electricity can flow between them.



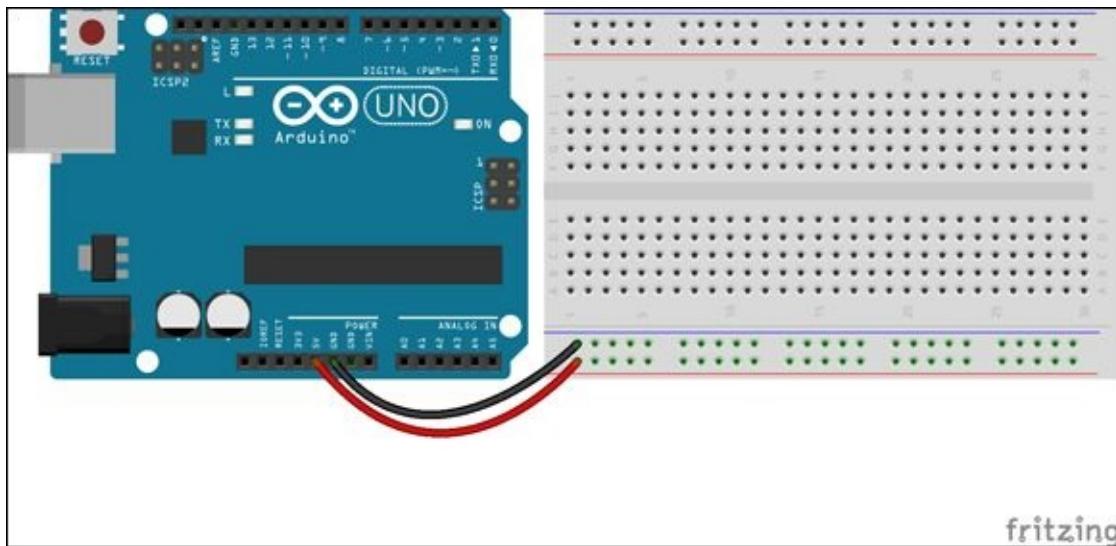
A **breadboard** is a reusable device that allows you to create circuits without needing to solder all the components. Breadboards have a number of holes into which you push wires and components to create circuits.



**FIGURE 1-14** A breadboard has a series of holes that are connected in rows with two pairs of long rows on the outside and shorter, perpendicular rows in the centre of the board.

The long rows on the outside edges of the board are where you can connect 5V and ground (GND on the Arduino board). Some boards may come with labels like + or - or colours like red and black or blue. Red is a colour used to represent the positive voltage, so with the Arduino Uno that would be 5V. Black or blue is used to represent ground or GND on the Arduino Uno.

You can think of the long rows for 5V and ground as expansions for the 5V and GND pins on the Arduino Uno. There is only one 5V pin on the Arduino Uno, so what if you have more than one component that needs to connect to 5V? By using a jumper wire, you can connect 5V to a row on the breadboard. You then have many holes where components can connect to 5V. It's the same for ground. There are more GND pins on the Arduino Uno (there are three, and they all do the same thing, so you can use any of them whenever you need to connect to ground), but you can also use a jumper wire to expand the number of ground connections on the breadboard. [Figure 1-15](#) shows how you can do this.



**FIGURE 1-15** Basic layout of a breadboard

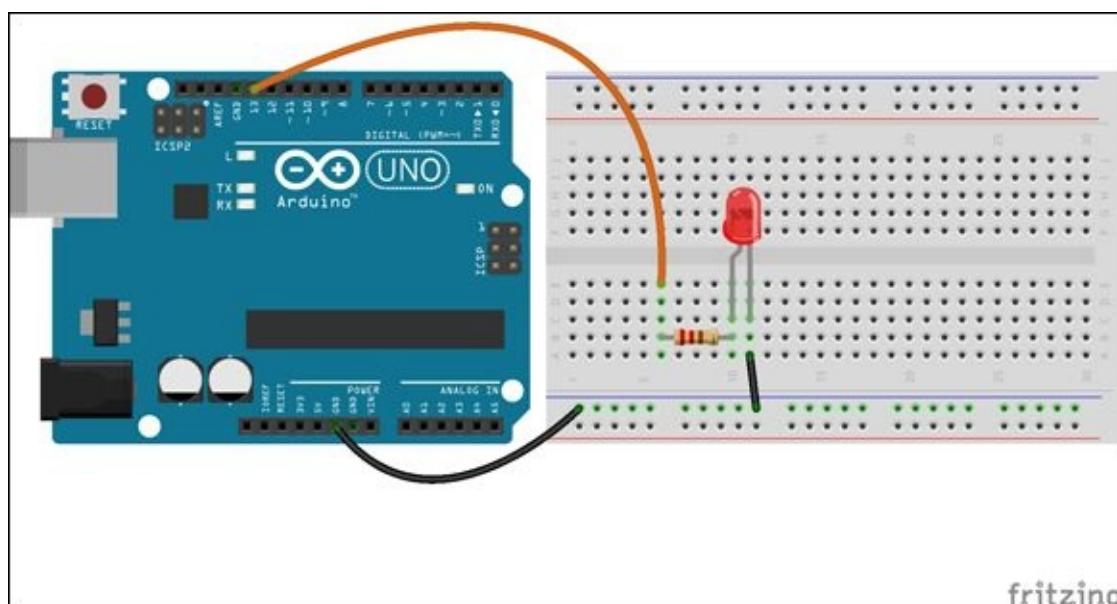
The rows in the middle of the breadboard, between the long outside rows, are what you use to connect your components to each other. If you could see inside the plastic case of the breadboard, you would see that these rows are perpendicular to the long rows on the edges. The holes are in groups of five and don't connect across the gap in the middle of the breadboard.

# Building Your First Circuit

You are now ready to build your LED circuit using your 220-Ohm resistor and LED. An LED is directional—that means you can accidentally put it in a circuit backwards. If you look at the LED closely, you can see that the two legs aren't the same length. The long leg of the LED should connect to the positive or 5V portion of the circuit, and the short leg should connect to ground. The resistor isn't directional, so both legs are the same length, and it doesn't matter which leg is connected to which part of the circuit.

Go through the following steps to build the circuit in [Figure 1-16](#):

1. Connect one end of a jumper wire to one of the GND pins on the Arduino and the other end to a long row on the breadboard. This is the black wire in [Figure 1-16](#), but your jumper wire can be any colour.
2. Put one leg of the resistor into any of the short rows in the middle of the breadboard.
3. Put the other leg into another short row in the middle of the breadboard. It just can't be in the same group of five holes as the other leg!
4. Put the long leg of the LED into a hole in the same row as one of the legs of the resistor. They are now touching the same piece of metal inside the breadboard, so electricity will eventually be able to flow through the resistor and then the breadboard row and then the LED.
5. Connect the short leg of the LED to the long row of the breadboard where the jumper wire is connected.
6. Use a second jumper wire to connect from Pin 13 to the same short row as the leg of the resistor that isn't connected to the LED.



[FIGURE 1-16](#) The LED circuit on the breadboard



Don't ever connect the 5V and GND pins together without a component like a resistor or LED in between them. This creates a short circuit and can damage your Arduino Uno. If you ever do this by

accident, your computer will probably notice that something is wrong on your Arduino Uno and will cut off the power from the computer to the Uno. If this happens, just unplug the Arduino Uno from the computer and then plug it in again.

Your LED should now blink on and off on the breadboard, just like the LED did on the Arduino board. Congratulations! You've built your first Arduino circuit! Your code is controlling electricity and whether a light is on or off. This is just the beginning of your journey to build some exciting Arduino projects.



Visit the companion site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino) to see a video showing how to build this circuit.

# CHALLENGE



Change the pin number that your LED circuit is connected to. You can use any of the pins from 2 to 13 on the section of the board labeled Digital, shown in [Figure 1-17](#). The other pins have special functions that you learn about in the next adventure.

Remember, you need to make a change in your code (and remember to upload your change to your Arduino board), and you need to make a change to the circuit.



**FIGURE 1-17** The digital pins on the Arduino board. Digital Pins 0 and 1 are special pins that you learn about later.

# Further Adventures with Arduino

Congratulations! You have achieved a lot. It might seem like that was a lot of work just to get a single light to flash on and off, but it means you are ready for all kinds of adventures.

Check out projects that others have done to get some inspiration for what is possible:

- <http://makezine.com/category/electronics/arduino/>
- [www.creativeapplications.net/tag/arduino/](http://www.creativeapplications.net/tag/arduino/)

## Arduino Command Quick Reference Table

Command	Description
setup()	Function that runs once when the Arduino Uno first starts. See also <a href="http://arduino.cc/en/Reference/Setup">http://arduino.cc/en/Reference/Setup</a> .
loop()	Function that is repeatedly run after the setup() is completed and until the Arduino is turned off. See also <a href="http://arduino.cc/en/Reference/Loop">http://arduino.cc/en/Reference/Loop</a> .
pinMode()	Sets the pin number entered as the argument to either output electricity or read it in. See also <a href="http://arduino.cc/en/Reference/PinMode">http://arduino.cc/en/Reference/PinMode</a> .
OUTPUT	Keyword set in second argument of pinMode() that says the pin will output electricity. See also <a href="http://arduino.cc/en/Reference/Constants">http://arduino.cc/en/Reference/Constants</a> .
digitalWrite()	Turns on or off the electricity at the specified pin. See also <a href="http://arduino.cc/en/Reference/DigitalWrite">http://arduino.cc/en/Reference/DigitalWrite</a> .
HIGH	Keyword used to turn on the electricity in digitalWrite(). See also <a href="http://arduino.cc/en/Reference/Constants">http://arduino.cc/en/Reference/Constants</a> .
LOW	Keyword used to turn off the electricity in digitalWrite(). See also <a href="http://arduino.cc/en/Reference/Constants">http://arduino.cc/en/Reference/Constants</a> .
delay()	Pauses the Arduino Uno for a specified number of milliseconds. See also <a href="http://arduino.cc/en/Reference/Delay">http://arduino.cc/en/Reference/Delay</a> .



**Achievement Unlocked:** You are making all the right connections and shining bright!

## In the Next Adventure

In the next adventure, you start adding interactivity to your Arduino code and control LEDs using a dial!



## Adventure 2

### Reading from Sensors

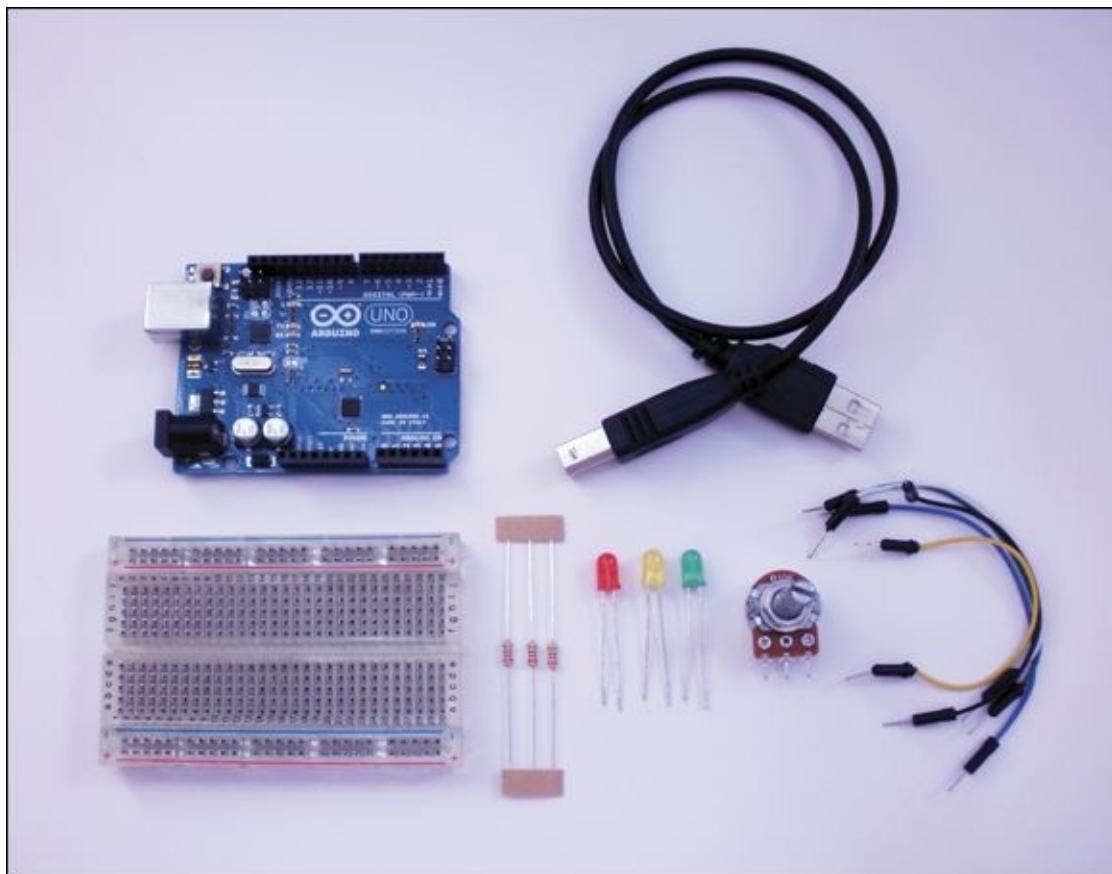
**YOU NOW HAVE** the Arduino software installed and have uploaded your first sketch to make sure everything is set up correctly. (If you haven't done that, it would be best to go to [Adventure 1](#) and do that now!) It's great telling the Arduino to control something like flashing lights, but the real fun with Arduino starts when your projects become interactive. There are a couple things that need to happen before a project can become interactive: first the Arduino needs to know something about what is happening in the real world; then the Arduino code needs to do something based on that information.

You're going to travel a long way in this adventure! You start by controlling multiple LEDs and then you print some messages from the Arduino to the computer. After that, you read in information from a sensor and print that information to the computer. Finally, you put all of that knowledge together to build a terrific status message sign, which will have multiple messages and a control knob you can turn to choose what message you want to display—perfect for welcoming or sending away visitors at your whim.

# What You Need

You first find out how to add more LEDs to your circuit and then how to use a sensor called a potentiometer. The following list tells you what you need, and [Figure 2-1](#) shows the electronic components. Remember, Appendix B includes suggestions of where you can buy everything.

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 3 LEDs (1 green, 1 yellow, 1 red)
- 3  $220\Omega$  resistors
- 1  $10k\Omega$  potentiometer
- 4 jumper wires



**FIGURE 2-1** The electronic components you need for the first part of this adventure

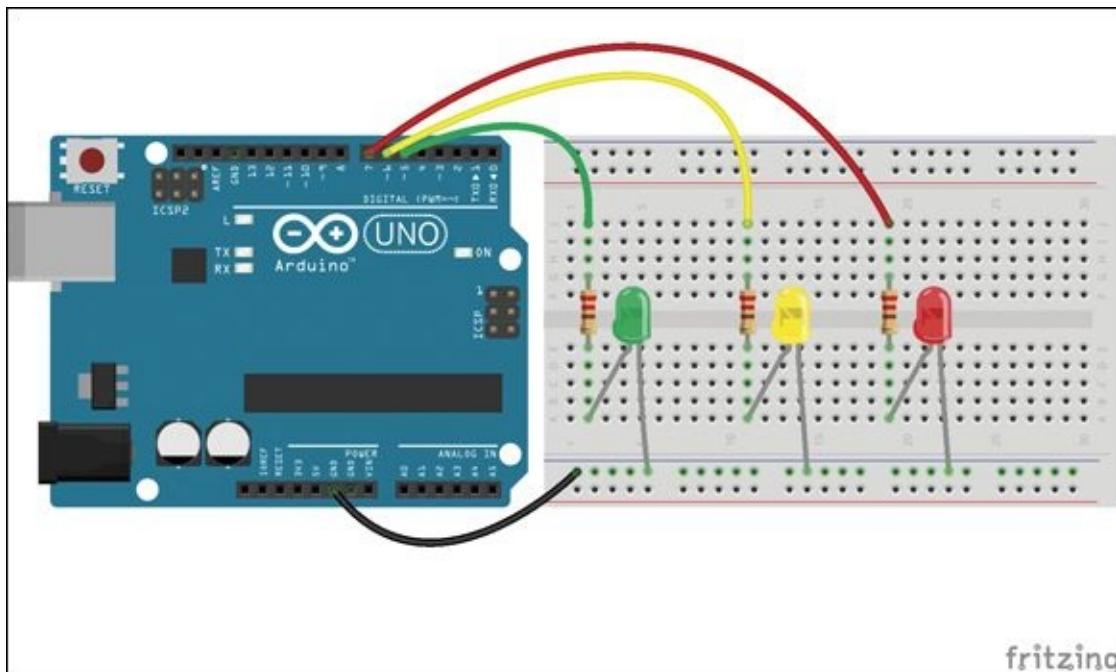
# Adding More LEDs

In Chapter 1, you built a circuit on a breadboard so that the Arduino controlled a single LED. One LED is great, but more LEDs are even better! So what do you need to do to add more LEDs?

The first thing you need is more LEDs—this adventure uses three. You also need three  $220\Omega$  resistors, as each LED needs its own current-limiting resistor. Resistors help control the flow of electricity in a circuit. In this circuit, the resistors protect the LEDs from becoming damaged from too much current. You can read more about current and resistors in [Adventure 1](#).

Start by building the circuit shown in [Figure 2-2](#):

1. Put one leg of one of the resistors in a short row on the top half of the breadboard towards the left side of the board. Put the other leg of the resistor in the short row across the gap in the middle of the breadboard directly below where you've inserted the first resistor leg. The rows of the breadboard aren't connected across the gap, so each resistor leg is in its own row; they aren't touching the same piece of metal inside the breadboard.
2. Repeat with the second and third resistors. Place one resistor in the centre of the breadboard and the other towards the right side of the breadboard. Each resistor should reach across the gap in the middle of the board and have one leg in a short row above the gap and the other in a short row below the gap.
3. Now add the LEDs. The long leg of each LED connects to the resistor and the short leg connects to ground. Insert the long leg of each LED into the same short row as each resistor. It should be placed just below the resistor. Place the green LED on the left side of the breadboard, the yellow in the middle and the red on the right side.
4. Insert the short leg of each LED into one of the long rows running the entire length of the breadboard at the very bottom. If your breadboard is labelled with a blue or black line or a -, insert the three short legs into that row. If your breadboard isn't labelled then you can use either row—just make sure all three of the legs are in the same row.
5. Your circuit is now built on your breadboard. All that is left is to connect it to your Arduino Uno. Use one jumper wire to connect from a GND pin (there are three of them on the Arduino Uno and you can use whichever you would like, they are all the same) to the long row on the breadboard where your three short LED legs are inserted.
6. Using three more jumper wires, connect one wire from Pin 5 on the Arduino Uno (not A5, but the 5 in the section labelled Digital) to the top of the resistor on the left side of the breadboard connected to the green LED. Use a second jumper wire to connect Pin 6 to the middle resistor connected to the yellow LED, and a third jumper wire to connect from Pin 7 to the last resistor connected to the red LED.



**FIGURE 2-2** Building a circuit to control three LEDs

Finished? Now you’re ready to write the code. It’s going to look a lot like the Blink sketch described in [Adventure 1](#). You use code to control when an LED turns on and off. The big difference is that you start using variables to keep track of the LEDs, and you need to control three LEDs instead of only one. You read more about variables in the next section.

First, launch the Arduino IDE. It opens a new sketch window when the program starts. You can also go to File=New or click the New button to create a new empty sketch. Type in the following sketch exactly as it is written. The spaces and indentation aren’t important; they just make the code easier to read. However, don’t accidentally leave off a ; or your code won’t run! Don’t forget to save it by going to File=Save or clicking the Save button.

```
// Pins
int greenLED = 5;
int yellowLED = 6;
int redLED = 7;

void setup() {
  // set to output to LED pins
  pinMode(greenLED, OUTPUT);
  pinMode(yellowLED, OUTPUT);
  pinMode(redLED, OUTPUT);
}

void loop() {

  // turn on all LEDs
  digitalWrite(greenLED, HIGH);
  digitalWrite(yellowLED, HIGH);
  digitalWrite(redLED, HIGH);
  // wait 1 second
  delay(1000);
```

```
// turn off all LEDs  
digitalWrite(greenLED, LOW);  
digitalWrite(yellowLED, LOW);  
digitalWrite(redLED, LOW);  
  
//wait 1 second  
delay(1000);  
}
```



Typing the sketches by hand is a good way to become more comfortable with all the new coding terms you are learning, but it can be frustrating if it's a really long sketch or you've made a typo that you can't find. For those cases, all the sketches are available to download from companion site [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

After you have finished typing out the sketch and have saved it, you are ready to upload it to your Arduino Uno. Select the board and port from the menus (you can review how to do this in [Adventure 1](#)). Click the Upload button and watch for messages to appear at the bottom of the window of the Arduino IDE.

If there aren't any errors, your three LEDs should start flashing on and off! That's so much more exciting than blinking just one LED.

If you see any errors that say “Expected initializer before 'void'” or “expected ';'...” then you probably have a typo. Look carefully over your code and make sure it matches what you are supposed to type in. If there are any errors from avrdude, check that your Arduino Uno is plugged into your computer and that you've chosen the correct options from the Board and Port menus. If you still have problems, go back through [Adventure 1](#) to make sure your software is installed correctly.

# DIGGING INTO THE CODE



Now that you have seen what the code does, it's time to figure out how it's doing that! The sketch can be broken up into three sections: the lines before the functions; the `setup()` function; and the `loop()` function.

First, the lines of code before the functions.

```
// Pins  
int greenLED = 5;  
int yellowLED = 6;  
int redLED = 7;
```

The first line beginning with `//` is a comment explaining what the following lines mean. The next three lines are **variables** assigned the pin numbers for each of the LEDs. A variable is like giving something in code a name. Here it's giving a name to the pin numbers to make it easier to remember which LED is connected to each pin. The name `greenLED` is much more obvious than the number `5`, isn't it?



A **variable** is a code construct that holds a value that can be changed. For example, the variable `greenLED` stores the number `5`.

In front of each of the variable names is `int`. This is describing what kind of data can be stored in the variable. `int` is short for integer, so the variables can store only whole numbers.

Although giving something a name is convenient, variables become really powerful when the variable changes its value but keeps the same name. That feature of variables isn't being used here, but it is in the next section.



There a few rules to keep in mind when creating new variable names. The first is that they can't have spaces in them, but you can use underscores (`greenLED` and `green_LED` are fine, but `green LED` is not). You also can't start the name with anything besides a letter (`led3` is fine, `31ed` is not). Lastly, it's not a requirement, but variables usually start with a lowercase letter (`greenLED` rather than `GreenLED`).

The next part of the sketch is the `setup()` function. This function is what the Arduino Uno runs right when it starts up. It is only run that one time, so this function is for commands that need to be done only once. The most common thing done in the `setup()` is to set the `pinMode()` of pins being used. The `pinMode()` determines whether the pin will output electricity to control a component like an LED or it will read in a signal from a sensor.

This circuit doesn't have any sensors—only LEDs—so the `pinMode()` is set to `OUTPUT` for each of the pins. Because variables are being used to represent the pin numbers, their names (`greenLED`, `yellowLED` and `redLED`) can be typed instead of `5`, `6` and `7`.

```
void setup() {  
    // set to output to LED pins  
    pinMode(greenLED, OUTPUT);  
    pinMode(yellowLED, OUTPUT);  
    pinMode(redLED, OUTPUT);  
}
```

The final section of code is the `loop()` function. This function is executed repeatedly until the Arduino Uno's power is removed.

The `loop()` function uses only two other functions: `digitalWrite()` and `delay()`. The `digitalWrite()` function turns on or off an LED. The first argument determines which LED is being talked about, and the second argument

determines what is to be done—either turn on the LED if the argument is **HIGH** or turn it off if the argument is **LOW**.

To summarise what the **loop()** does, it turns on each LED, then waits for 1 second so you can see them on, then it turns off each LED and waits for 1 second so you can see that they are off.

```
void loop() {  
    // turn on all LEDs  
    digitalWrite(greenLED, HIGH);  
    digitalWrite(yellowLED, HIGH);  
    digitalWrite(redLED, HIGH);  
  
    // wait 1 second  
    delay(1000);  
  
    // turn off all LEDs  
    digitalWrite(greenLED, LOW);  
    digitalWrite(yellowLED, LOW);  
    digitalWrite(redLED, LOW);  
  
    //wait 1 second  
    delay(1000);  
}
```

# CHALLENGE



Now you have three LEDs that all blink together, make them light up in a sequence one by one. You won't need to change the circuit; you change only the code. Remember to upload your changes in your code to the Arduino Uno.

A few hints:

- You don't have to write more `digitalWrite()` functions; you only need to move around the ones already written.
- You need to add more `delay()` functions.

# Printing Messages to the Computer

After you upload code onto the Arduino Uno, the board doesn't talk to the computer. In fact, it doesn't need the computer at all. The only thing that's happening here is that the Arduino Uno is getting its power from the computer, via the USB cable plugged into the computer. You could even unplug it from the computer and use a battery. (You can find information about powering the board with a battery in [Adventure 5](#).) But your computer can still be useful after you have uploaded your sketch onto the Arduino Uno. For example, the Arduino Uno doesn't have a screen, but the computer does. So if you are **debugging** your code, or just want to read messages to know what the board is doing, the computer and USB cable can be a powerful tool.



**Debugging** is the where you locate the cause of any errors in your computer program code and fix them.

The Arduino Uno can talk with the computer using **serial communication** over the USB cable. This just means that one bit of data is sent at a time. You can think of it as one letter of a word being sent at a time, eventually to spell out a whole word. To send data from the Arduino to the computer, there are just three functions you need to know about.



**Serial communication** is one way that two devices, like a computer and an Arduino board, can send and receive data to each other. One piece of data is sent at a time.

The first function is one that you only need to call once in the **setup()** function:

```
Serial.begin(9600);
```

This function takes only one argument: the speed at which the Arduino Uno is sending out and receiving data. It's important that this number is the same as the speed that the computer is sending and receiving data (you find out how to check that later). If the computer and Arduino Uno are sending and receiving data at different speeds, they won't be able to understand each other. Unless you are explicitly told to use a different number, **9600** is a good number to use.



Remember that you can always look up terms in the glossary at the back of the book.

The other two functions are:

```
Serial.print("Your message goes here");
Serial.println("Your message goes here");
```

These are the functions that send messages from the Arduino board to the computer. The

first one, `Serial.print()`, doesn't send a **newline character** at the end of the message; in other words, the cursor isn't moved down to the next line at the end of the message. The function `Serial.println()` does include a newline; you can imagine this as a message with an Enter at the end of it.



A **newline character** is like pressing the Enter or Return key on your keyboard.

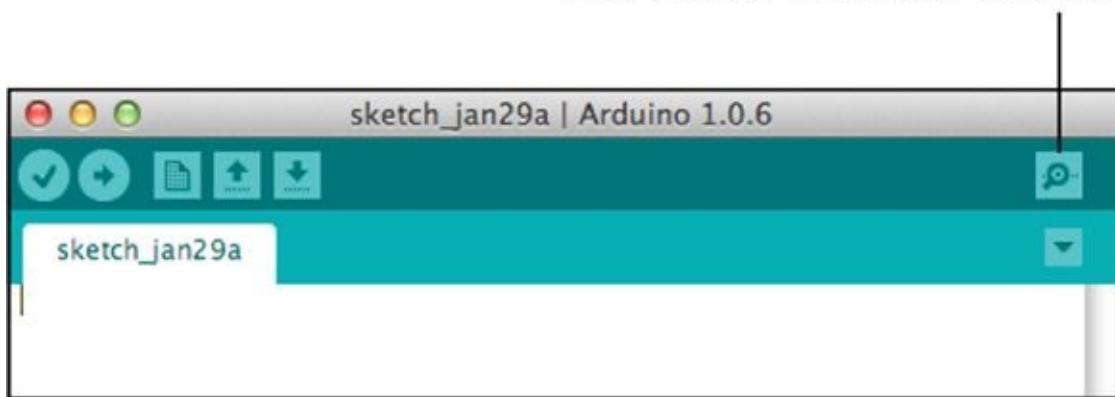
The best way to understand this is to try it out. There's no circuit for this; you just need the Arduino board plugged into your computer. Create a new sketch with the following code and upload it onto the board:

```
void setup() {  
    // to start serial communication  
    // the argument needs to match  
    // the rate you choose in the  
    // Serial Monitor  
    Serial.begin(9600);  
    Serial.println("Hello, this is from setup");  
  
    // a delay so that messages aren't too quick to read  
    delay(3000);  
}  
  
void loop() {  
    // printing a message and then waiting a second  
    Serial.print("This is from loop, with a print. ");  
    delay(1000);  
    Serial.println("And this is from loop with a println.");  
    delay(1000);  
  
    int myVariable = 27;  
    Serial.print("And this is printing a variable: ");  
    Serial.println(myVariable);  
    delay(1000);  
}
```

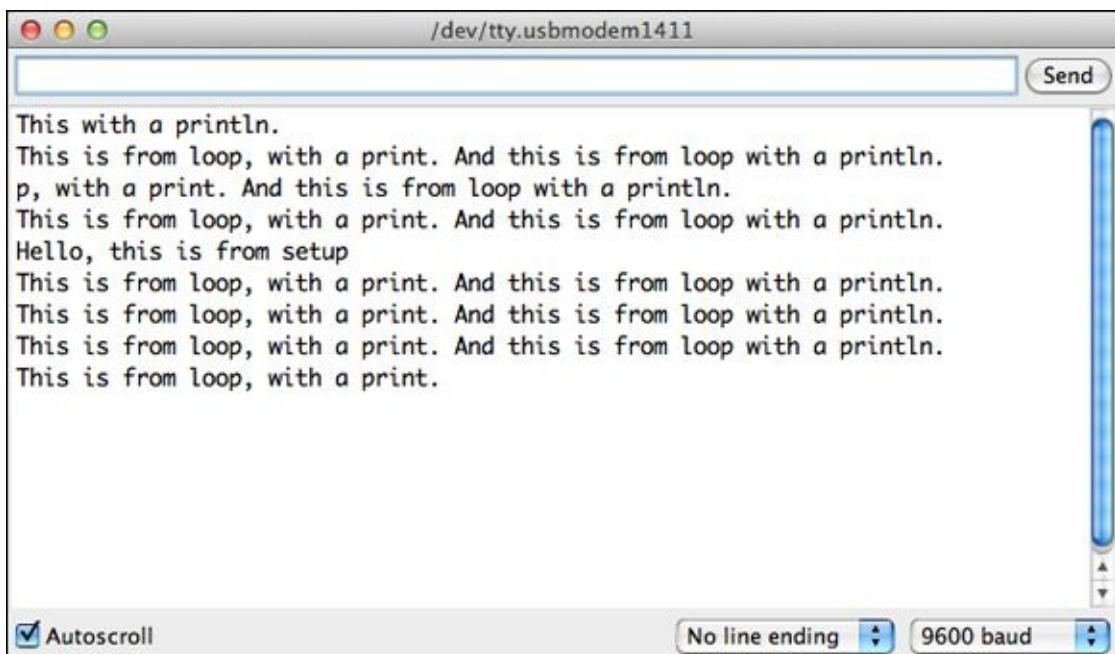


You may have noticed that the messages you send in `Serial.print()` and `Serial.println()` are between quotation marks (""). This is how you write in code a piece of text that you don't want the computer to interpret as code. You can print the value of a variable by replacing the message and the "" with a variable name, such as the variable `myVariable` in the sketch.

After the sketch is uploaded onto the board, open the Serial Monitor by clicking that last Arduino IDE button—the one I didn't cover in [Adventure 1](#). It's the button on its own on the right in [Figure 2-3](#). After you click the button, a window opens like the one shown in [Figure 2-4](#). You may notice that the number 9600 appears in the bottom right of the window (if it doesn't you should click the number that is there and select 9600). This is the number that needs to match the argument in `Serial.begin()` in your Arduino code.



**FIGURE 2-3** The Serial Monitor button



**FIGURE 2-4** The Serial Monitor in the Arduino IDE

You will see the messages from your Arduino Uno appear in your Serial Monitor. The Serial Monitor is a tool built into the Arduino IDE that lets you see the messages sent by the Arduino Uno using serial communication. You will find it a very useful tool in all of your adventures! When you are done with the Serial Monitor, you can just close the window. It may also close on its own when you upload a new sketch to your board. You can reopen the Serial Monitor to see any new messages.

# DIGGING INTO THE CODE



So what's going on in the code? There are the sections that you are probably getting used to: the `setup()` and `loop()` functions. The `setup()` function doesn't have much going on. The first four lines are comments explaining what is happening. The serial communication is then started and a message is sent: `Hello, this is from setup.` The Arduino Uno is then paused for 3 seconds, just so that messages don't print too fast to read.

```
void setup() {  
    // to start serial communication  
    // the argument needs to match  
    // the rate you choose in the  
    // Serial Monitor  
    Serial.begin(9600);  
    Serial.println("Hello, this is from setup");  
  
    // a delay so that messages aren't too quick ↵ to read  
    delay(3000);  
}
```

The `loop()` function then prints messages in three different ways. It first uses `Serial.print()` to print `This is from loop, with a print.` and then waits for 1 second. Because the message used `Serial.print()` and not `Serial.println()`, the next message appears on the same line, with the new message printing right after the previous one. There's another 1-second delay and then a new variable is created to hold the number `27`. A `Serial.print()` function prints a message and then the variable prints at the end of the line. Because the variable is printed without the surrounding quotation marks (not `"myVariable"`), the number `27` is printed.

```
void loop() {  
    // printing a message and then waiting a second  
    Serial.print("This is from loop, with a print. ");  
    delay(1000);  
    Serial.println("And this is from loop with a println.");  
    delay(1000);  
  
    int myVariable = 27;  
    Serial.print("And this is printing a variable: ");  
    Serial.println(myVariable);  
    delay(1000);  
}
```



The `setup()` function is run once when the Arduino Uno is first turned on, but the Arduino Uno calls the `setup()` function a couple other times besides when it is first turned on. For example, there's a reset button on the board that you can use to manually restart the board, so the `setup()` function is called before going on to the `loop()` function. Also, whenever a new serial connection is made, the board restarts. So whenever you open the Serial Monitor, the Arduino board restarts and runs the `setup()` function again. You may notice this happening when you open the Serial Monitor. The message being printed gets interrupted, and the message in the `setup()` function starts printing instead. Try opening the Serial Monitor and then pressing the reset button on the Arduino Uno.

# Reading Data from a Potentiometer

Now you're ready to take a physical action in the real world and use it to make decisions in code. This is exactly what the Arduino was built to do! You will soon be turning a knob to change what LED is on.

You're going to start by using a **potentiometer**. That's a big name for a simple component! It's simply a resistor that adjusts how much resistance it has as you turn a control knob—for example, the knob you use on a stereo to turn up the volume. Potentiometers come in lots of shapes and sizes. [Figure 2-5](#) shows a few potentiometers that change their resistance through a rotating motion. The two bigger ones can have knobs or dials that fit over the end of the shafts. The little one is sometimes called a trimpot, and it already has a small knob attached that you turn. Trimpots like the one shown in [Figure 2-5](#) work well when building circuits on breadboards as they fit into the breadboard holes. Larger potentiometers like the one on the right in [Figure 2-5](#) can also fit into breadboards. Potentiometers like the one in the middle are easier for soldering wires to.



A **potentiometer** is a type of resistor with an adjustable knob to vary the resistance of current.

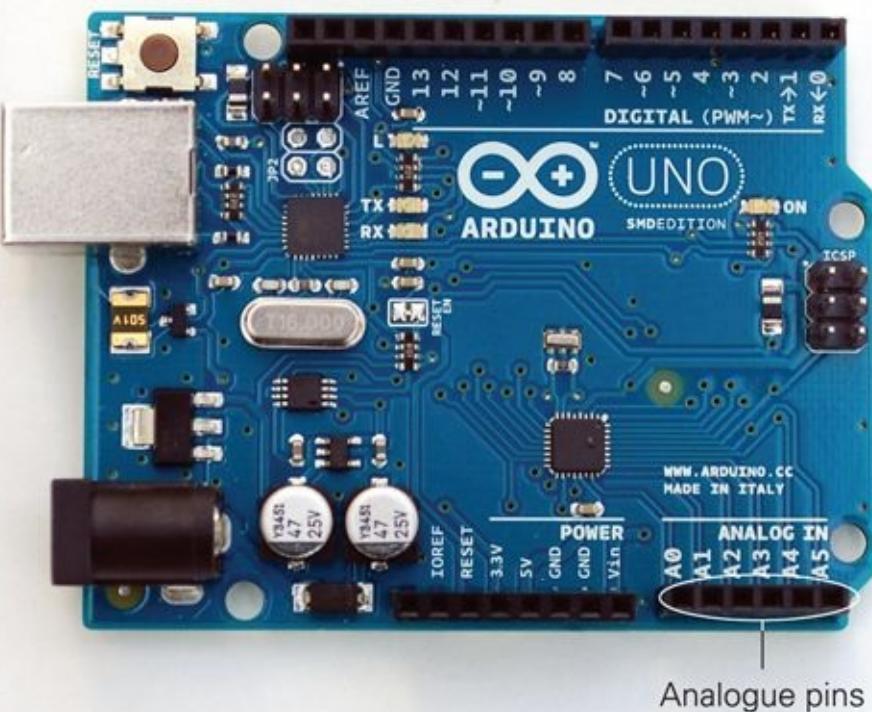


[FIGURE 2-5](#) Different potentiometers



Visit the companion site at [www.wiley/go/adventuresinarduino](http://www.wiley/go/adventuresinarduino) to watch a video showing different types of potentiometers.

The circuit to connect the potentiometer to the Arduino involves three connections. You can think of the potentiometer as having two kinds of pins: a pair of outside pins and an inside pin. The inside pin is what is connected to an Analogue Pin on the Arduino. The Analogue Pins are the section of pins you haven't yet used (see [Figure 2-6](#)). There are six pins in total and they each start with the letter A (A0, A1, A2, A3, A4 and A5).



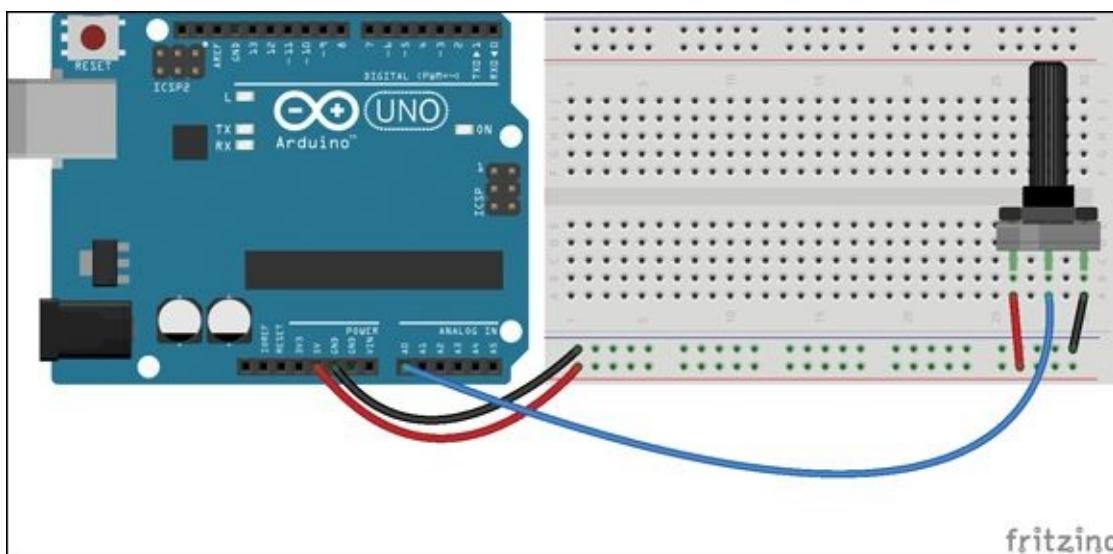
**FIGURE 2-6** Analogue pins on the Arduino Uno

It's time to build your first circuit with a potentiometer! Go through the following steps to build the circuit in [Figure 2-7](#):

1. Use a jumper wire to connect 5V on the Arduino Uno to one of the long rows running along the bottom of the breadboard. If the breadboard is labelled with a red line or +, connect 5V to that row; otherwise, choose either row.
2. Use another jumper wire to connect GND on the Arduino Uno to the other long row on the breadboard.
3. Insert the legs of the potentiometer into any of the short rows in the middle of the breadboard.
4. Use a jumper wire to connect one of the outside legs of the potentiometer to the long

row connected to GND on the Arduino Uno.

5. Use another jumper wire to connect the other outside legs of the potentiometer to the long row connected to 5V on the Arduino Uno.
6. Connect the middle leg of the potentiometer to pin A0 on the Arduino Uno.



**FIGURE 2-7** Circuit for connecting a potentiometer

You can read in a value coming from the potentiometer that corresponds to the position of the shaft or knob on the potentiometer and print it to the Serial Monitor. To do this, go to File⇒ Examples⇒01.Basics and open the sketch AnalogReadSerial. Then upload the sketch to your Arduino Uno (remembering to set the board and port as you did in [Adventure 1](#) and earlier in this adventure).

When you've finished, click the Serial Monitor button to open the Serial Monitor. Rotate the potentiometer all the way to the left and then all the way to the right. You should see numbers displayed in the Serial Monitor, ranging from **0** at one end to **1023** at the other. These are the minimum and maximum numbers that the Arduino can read in from an analog input. When the number is **0**, the pin is reading in ground (0V). When it's **1023**, it means that the pin is reading in 5V. Any number in between means that it is reading in a voltage that's somewhere between ground and 5V. 5V is the maximum voltage that the Arduino Uno outputs and 0V is the minimum, so this circuit measures whether the potentiometer is all the way to the left or right by measuring whether the voltage the potentiometer is outputting is the maximum, minimum or something in between.

# DIGGING INTO THE CODE



So what is happening in the code? There's one line of code in the AnalogReadSerial sketch that you haven't seen before:

```
int sensorValue = analogRead(A0);
```

This line reads in the value (or voltage) being output by the potentiometer circuit to Pin A0 using the function `analogRead()`. This function gives you number between `0` and `1023`. You need to save this value into a variable so that you can do something with this number later. A new variable, `sensorValue`, is created, and the number that `analogRead()` reads in is saved in that variable. That variable is then printed to the Serial Monitor.



You might have already noticed that Arduino uses American spellings for words like “analog.” It’s important to remember this as an Arduino Uno doesn’t know what `analogueRead()` means, only `analogRead()`.

## CHALLENGE



Switch the outside pins on the potentiometer so that the one that was connected to 5V is now connected to ground, and the one that was connected to ground is now connected to 5V. You don't need to change any of the Arduino code.

What changes when you have the circuit set up this way as opposed to how it's wired in [Figure 2-7](#)?

# Making Decisions in Code

To build interactive projects, you need to be able to take input from the real world and then have the Arduino Uno do what you want according to that input. That means you need to use code to make decisions based on incoming information from sensors. For example, if you were building a burglar alarm, you would want to sound the alarm only if the alarmed door was open, so you need to know how to explain that in code.

Computers work by answering yes or no questions. Those yes or no questions need to be phrased like this:

- Is 3 greater than 5?
- Is 10 equal to 10?
- Is 4 less than or equal to 8?

Written in code (so that the computer understands it) these questions would look like this:

```
3 > 5
10 == 10
4 <= 8
```

The computer or Arduino Uno can then do different things based on whether the answer to the question is yes or no (or true or false). It does this by using **if** statements. If the answer to the question in the **( )** is yes, then the code between the **{** and **}** is executed:

```
if(a<b) {
    // then execute the code in here
}
```

If the answer is no, then the code in the **{ }**  is skipped.

For example, the following code:

```
if(3<5) {
    Serial.println("The statement is true.");
}
Serial.println("This is after the if statement.");
```

would print the following:

```
The statement is true.
This is after the if statement.
```

But the following code:

```
if(3<1) {
    Serial.println("The statement is true.");
}
Serial.println("This is after the if statement.");
```

would print:

```
This is after the if statement.
```

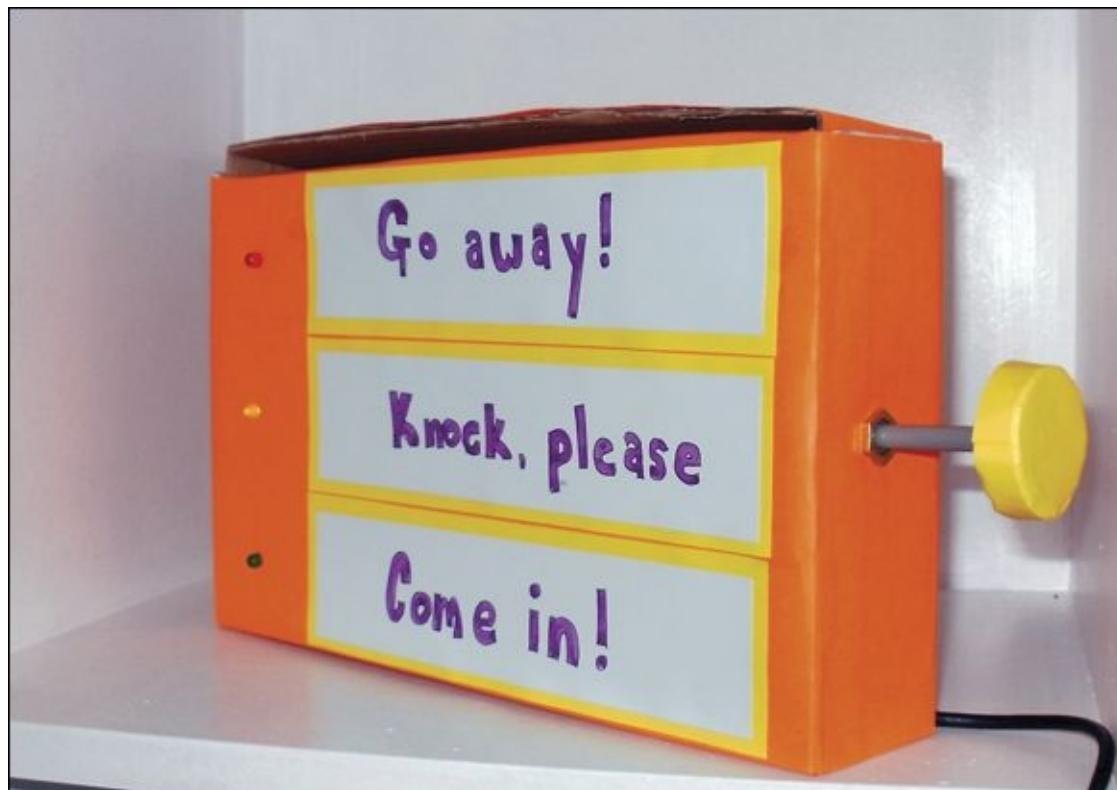
It's always easier to understand a new concept by building something yourself and seeing how it works. In the next section, you're going to use **if** statements in a project to create a

status message sign.

# Building a Status Message Sign

Have you ever seen a recording studio, either in real life or on a TV program or movie? There is usually a sign on the outside of the room that says “RECORDING” that lights up to let people know not to enter because there’s a recording session in progress.

You are going to build your own sign that lets other people know whether they can enter, knock first or stay out. (If you don’t want to use the signs suggested here, feel free to make up three different messages that are entirely your own.) Each message has an LED next to it. The lighted LED indicates which message is the active one. You set which message you want to be active with a control knob on the side of the sign, as shown in [Figure 2-8](#).

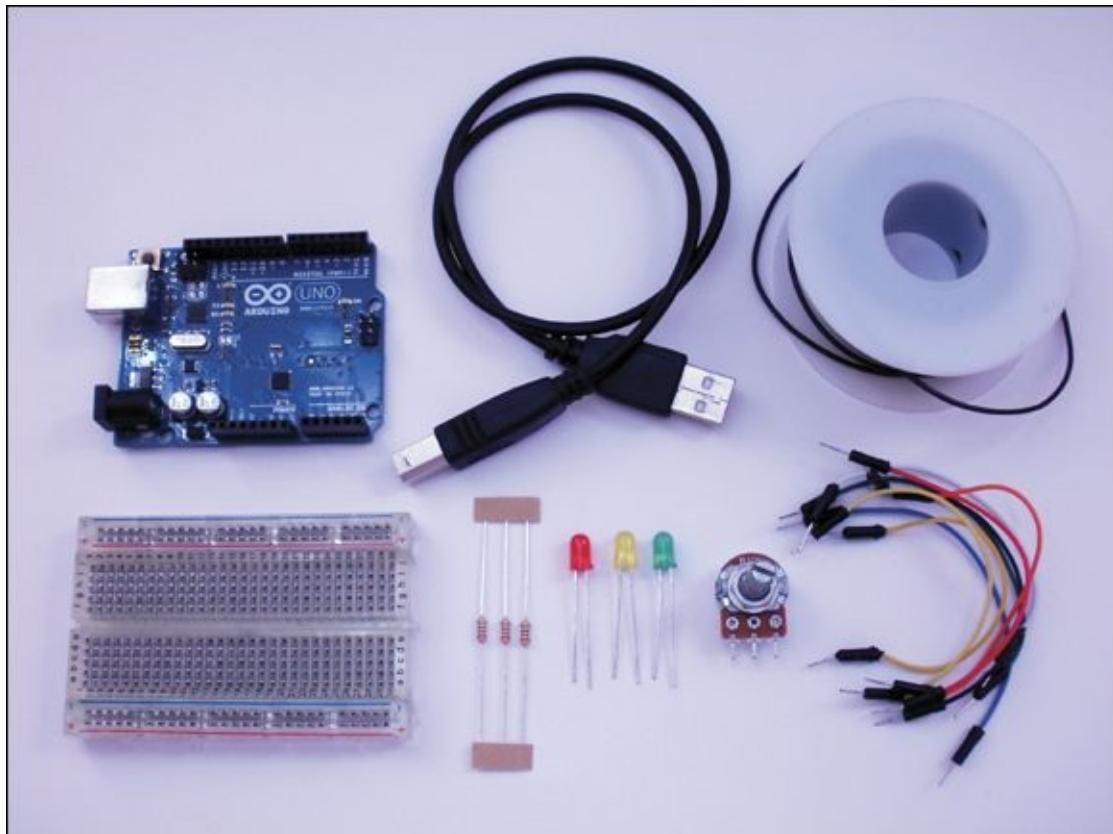


**FIGURE 2-8** A status message sign

## What You Need

For this adventure you build your first project with its own housing. You need the following tools and materials to first build and test the circuit on a breadboard and then also the tools and materials to build the complete project. [Figure 2-9](#) shows the electronic components that you need.

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 3 LEDs (1 green, 1 yellow, 1, red)
- 3  $220\Omega$  resistors
- 1  $10k\Omega$  potentiometer
- 8 jumper wires
- Some wire
- Some electrical tape
- Some solder
- A shoebox or other small box
- Paper or paint to decorate the box
- Scissors or a utility knife
- A soldering iron
- Wire cutters
- Wire strippers
- USB power supply (optional)

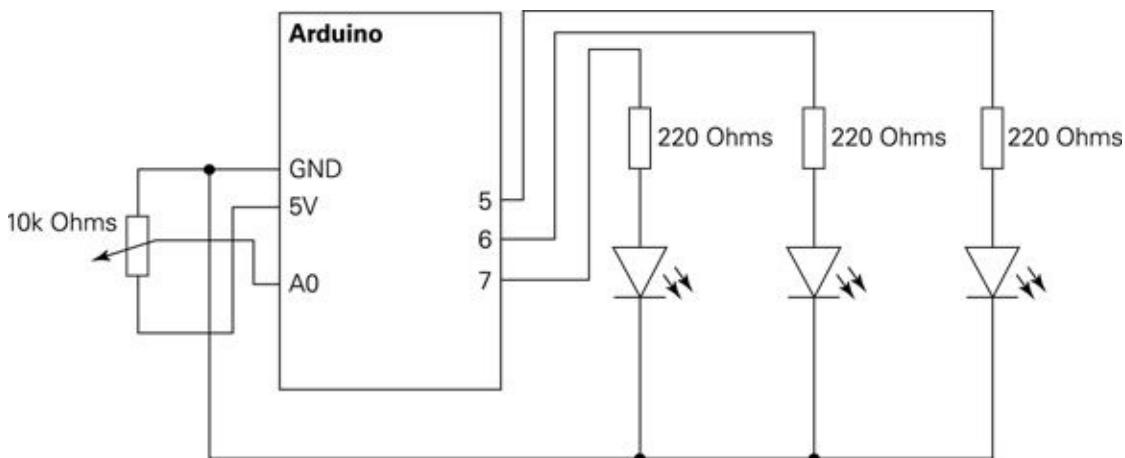


**FIGURE 2-9** The electronic components you need to make a status message sign

## Understanding the Circuit

As soon as you start soldering and gluing materials together, if you make a small mistake it can be difficult to undo. There's a way around this: before you create your finished circuit, you should always make a prototype of it on a breadboard first, to make sure the circuit works properly. That way, if you make any mistakes in your design you can easily correct them before you have the components permanently in place.

[Figure 2-10](#) shows the circuit that you're going to build for your sign. You will be building a circuit with three LEDs and one potentiometer. The LEDs will be connected to Pins 5, 6 and 7, and the potentiometer will be connected to ground, Pin A0 and 5V.



**FIGURE 2-10** Circuit schematic for the sign

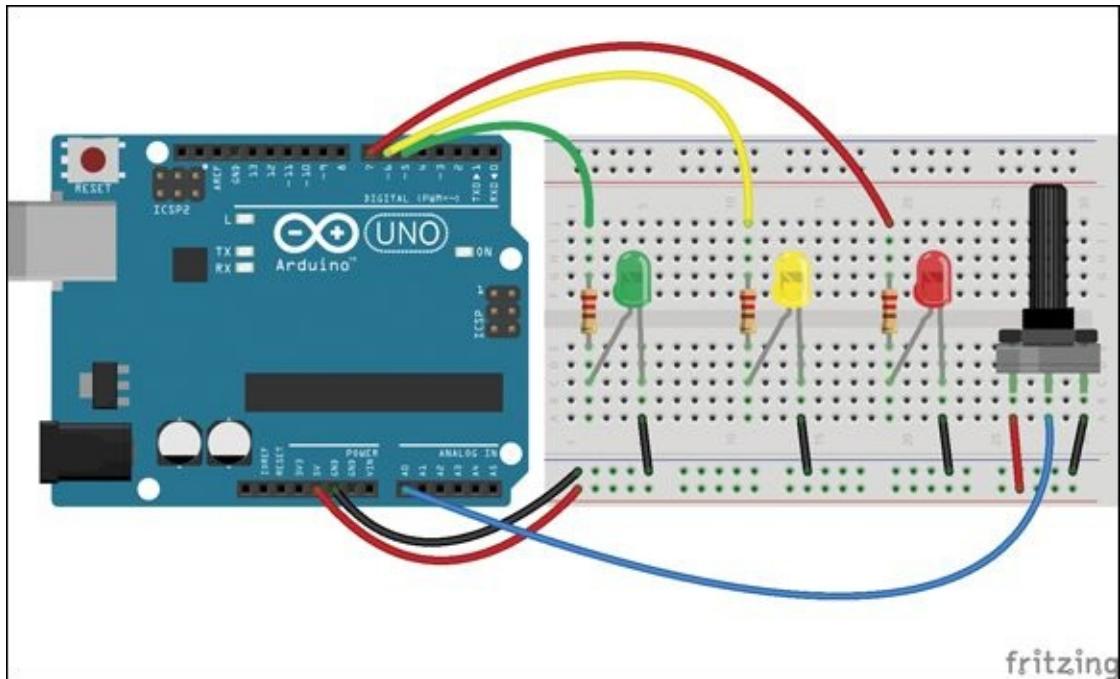
You are now going to test the circuit and the Arduino code on the breadboard, then you will rebuild the circuit without a breadboard.

# Prototyping on a Breadboard

To build your prototype circuit, use the following steps:

1. Use a jumper wire to connect 5V on the Arduino Uno to one of the long rows running along the bottom of the breadboard. If the breadboard is labelled with a red line or +, connect 5V to that row; otherwise, choose either row.
2. Use another jumper wire to connect GND on the Arduino Uno to the other long row on the breadboard.
3. Insert the legs of the potentiometer into any of the short rows in the middle of the breadboard.
4. Use a jumper wire to connect one of the outside legs of the potentiometer to the long row connected to GND on the Arduino Uno.
5. Use another jumper wire to connect the other outside legs of the potentiometer to the long row connected to 5V on the Arduino Uno.
6. Connect the middle leg of the potentiometer to pin A0 on the Arduino Uno.
7. Put one leg of one of the resistors in a short row on the top half of the breadboard towards the left side of the board. Put the other leg of the resistor in the short row across the gap in the middle of the breadboard directly below where you've inserted the first resistor leg. The rows of the breadboard aren't connected across the gap, so each resistor leg is in its own row—they aren't touching the same piece of metal inside the breadboard.
8. Repeat with the second and third resistors. Place one resistor in the centre of the breadboard and the other towards the right side of the breadboard. Each resistor should reach across the gap in the middle of the board and have one leg in a short row above the gap and the other in a short row below the gap.
9. Now add the LEDs. The long leg of each LED connects to the resistor, and the short leg connects to ground. Insert the long leg of each LED into the same short row as each resistor. It should be placed just below the resistor. Place the green LED on the left side of the breadboard, the yellow in the middle and the red on the right side.
10. Insert the short leg of each LED into the long rows running the entire length of the breadboard at the very bottom that is connected to GND on the Arduino Uno.
11. Using three more jumper wires, connect one wire from Pin 5 on the Arduino Uno (not A5, but the 5 in the section labelled Digital) to the top of the resistor on the left side of the breadboard connected to the green LED. Use a second jumper wire to connect Pin 6 to the middle resistor connected to the yellow LED and a third jumper wire to connect from Pin 7 to the last resistor connected to the red LED.

When finished, your prototype circuit should look like the one in [Figure 2-11](#). Notice anything? The full circuit for the sign is a combination of the two circuits you were worked with earlier in this chapter. The potentiometer is read into Pin A0, and the three LEDs are controlled by the output on Pins 5, 6 and 7.



**FIGURE 2-11** Prototype circuit on the breadboard for the sign

## Writing the Code

Next you need the code. Launch the Arduino IDE and type the following sketch in a new sketch window. Don't forget to save it!

Start your sketch by creating empty `setup()` and `loop()` functions.

```
void setup() {  
}  
  
void loop() {  
}
```

Next add a variable at the top of the sketch to keep track of your potentiometer.

```
// Pins  
int potPin = A0;
```

In the `setup()`, start serial communication so you can print messages to Serial Monitor. Type the following lines between the `{` and `}` of the `setup()`.

```
// start serial  
Serial.begin(9600);
```

The `loop()` function controls all the action. The value from the potentiometer is read and saved in a variable called `potValue`. A different message is then printed according to the number saved in the `potValue` variable. The message prints out what should happen with the LEDs. Type the following lines between the `{` and `}` of the `loop()`.

```
int potValue = analogRead(potPin);  
  
// print what the pot value is  
Serial.print("Potentiometer is: ");  
Serial.println(potValue);  
  
// if pot is less than 341  
if(potValue < 341) {  
    Serial.println("Turn on green, turn off yellow and red");  
}  
  
// if pot more than or equal to 341 and  
// less than 682  
if(potValue >= 341 && potValue < 682) {  
    Serial.println("Turn on yellow, turn off green and red");  
}  
  
// if pot more than or equal to 682  
if(potValue >= 682) {  
    Serial.println("Turn on red, turn off green and yellow.");  
}  
  
// A pause to slow down the messages  
delay(50);
```

Save the sketch and upload it to your Arduino Uno. Open the Serial Monitor and see what happens when you turn the potentiometer. You should see the value of the potentiometer

print along with what the LEDs should be doing—but you haven't programmed the LEDs yet. Time to do that now!

Add three more variables to keep track of the LED pins at the top of your sketch.

```
int greenLED = 5;  
int yellowLED = 6;  
int redLED = 7;
```

Inside `setup()`, add the code to set each `pinMode()`.

```
// set to output to LED pins  
pinMode(greenLED, OUTPUT);  
pinMode(yellowLED, OUTPUT);  
pinMode(redLED, OUTPUT);
```

In the `loop()`, add the `digitalWrite()` functions to turn on and off each LED (shown in bold in the following code). You can also remove the `delay()` at the end of the `loop()`.

Your full sketch should look like this:

```
// Pins  
int potPin = A0;  
int greenLED = 5;  
int yellowLED = 6;  
int redLED = 7;  
  
void setup() {  
    // set to output to LED pins  
    pinMode(greenLED, OUTPUT);  
    pinMode(yellowLED, OUTPUT);  
    pinMode(redLED, OUTPUT);  
  
    // start serial  
    Serial.begin(9600);  
}  
  
void loop() {  
    int potValue = analogRead(potPin);  
  
    // print what the pot value is  
    Serial.print("Potentiometer is: ");  
    Serial.println(potValue);  
  
    // if pot is less than 341  
    if(potValue < 341) {  
        Serial.println("Turn on green, turn off yellow and red");  
        // turn on green LED  
        digitalWrite(greenLED, HIGH);  
  
        //turn off yellow and red LEDs  
        digitalWrite(yellowLED, LOW);  
        digitalWrite(redLED, LOW);  
    }  
  
    // if pot more than or equal to 341 and  
    // less than 682  
    if(potValue >= 341 && potValue < 682) {
```

```
Serial.println("Turn on yellow, turn off green and red");
// turn on yellow LED
digitalWrite(yellowLED, HIGH);

// turn off green and red LEDs
digitalWrite(greenLED, LOW);
digitalWrite(redLED, LOW);
}

// if pot more than or equal to 682
if(potValue >= 682) {
    Serial.println("Turn on red, turn off green and yellow.");
    // turn on red LED
    digitalWrite(redLED, HIGH);

    // turn off green and yellow LEDs
    digitalWrite(greenLED, LOW);
    digitalWrite(yellowLED, LOW);
}
}
```

Upload the sketch with the circuit on the breadboard. If you don't want to type all the code, you can download the sketch from the companion site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

Ready? Time to try it out. You should be able to change which LED turns on by turning the potentiometer. Only one LED should turn on at a time. You can open the Serial Monitor in the Arduino IDE to make sure the correct values are coming from the potentiometer.

# DIGGING INTO THE CODE



There is one bit of code in the sketch for the status message sign that you haven't seen before: `&&`. Those two ampersands (`&&`) without a space in between means that both the piece of code before it and after need to be true.

For example:

```
4<6 && 10<20
```

is true because both `4<=6` and `10<20` are true. But:

```
3>9 && 5<7
```

is false because only `5<7` is true; `3>9` is false. The `&&` symbol is a way to combine restrictions in an `if` statement. In your sketch, it's used to turn on the yellow LED only if `potValue>=341` and also `potValue<682`.

## Creating your Sign

In any project, the thing that really brings it come to life is the structure in which you house the electronics. It doesn't just protect your electronics and hide the parts you don't want to see—it also gives you a chance to get creative. The code and circuit are a big part of the creative process of making an Arduino project, of course! But this is the part where you can really let your imagination run riot so you can show off your project to your friends and family by getting it off the breadboard and into a stylish new home!

You can choose whatever materials you would like to use to create your sign, but a shoebox works well. It can easily be cut with a utility knife or scissors and decorated with paper and glue or paint, and you'll be able to make it as personal as you like.



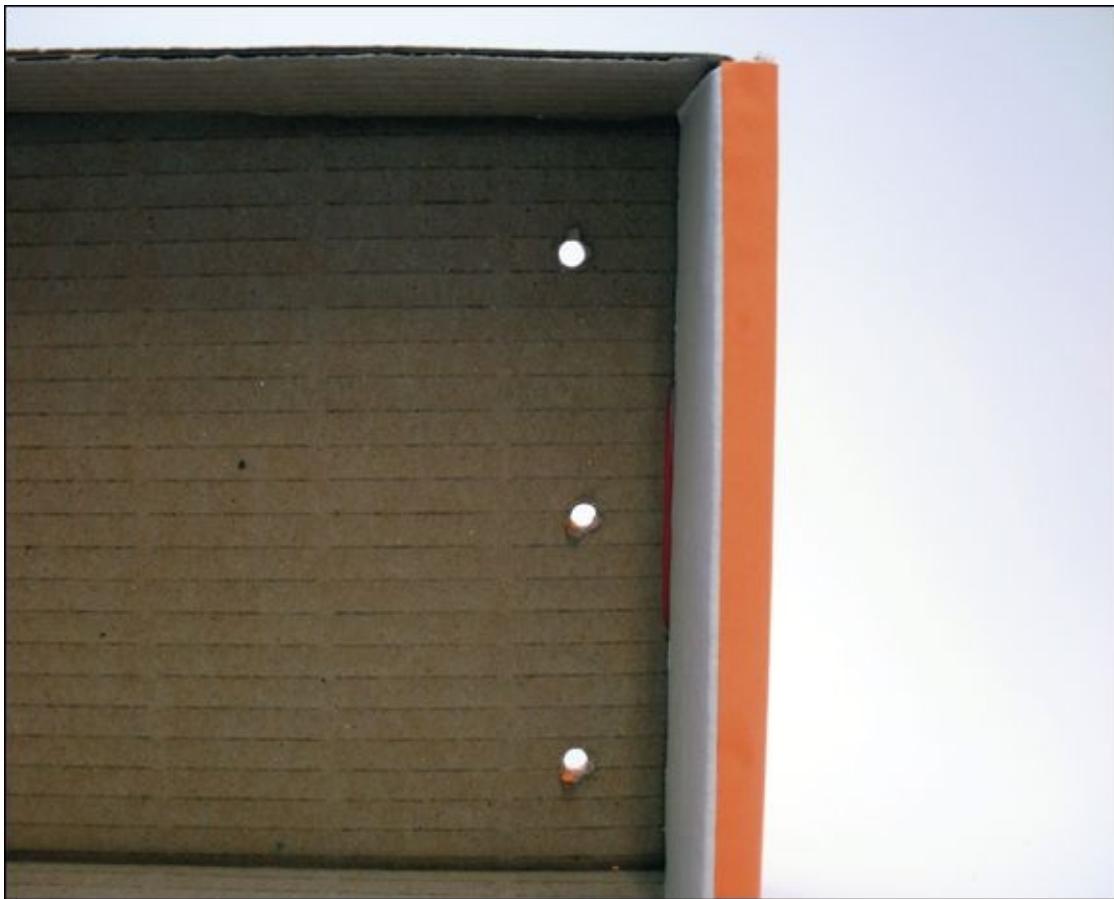
You can watch a video demonstrating how to build the sign and solder the circuit on the companion site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

## Cutting Holes for the Potentiometer and LEDs

Before you decorate the box, you need to cut some holes in it where you want your LEDs and the knob of the potentiometer to be located. Make five holes: three for the LEDs, one for the knob and one for the USB cable. Measure the lenses of the LEDs and the shaft of the control knob so you can make the holes just big enough for those components to fit snugly into them. (If you make the holes too big, the LEDs and knob will just fall out!) For the USB cable, the hole needs to be large enough for you to pass the end of the USB cable that plugs into the Arduino through it.



If you are using a shoebox, I recommend that you make the holes for the LEDs, knob ,and USB cable in the bottom of the box, as shown in [Figure 2-12](#). That way you can easily remove the lid to access the electronics and then quickly hide them all away.



**FIGURE 2-12** Cutting holes for the LEDs and knob

## Adding the Status Messages and Decorating the Sign

Now you're ready to transform your old shoebox into a slick sign box by painting it or covering it with paper. Decide on your messages—you can use the messages I've suggested in [Figure 2-11](#) or create your own. It doesn't matter whether you write or paint them onto the box yourself, print them from a computer and glue them next to the LEDs, cut them out of magazines—do whatever you like. There are no limits! Express your creativity—use paint, markers, crayons or whatever you have available. In my opinion, you can seldom go wrong with glitter. Or why not use natural materials like feathers or dried flowers?

## Soldering the Circuit

You know that your code and circuit work (and if you haven't tested them, go back and do that!), so you are ready to more permanently assemble your circuit. Circuits depend on electricity flowing through conductive materials like metal. That means you can't use things like glue to connect components—the electricity can't flow through glue. Instead you use solder. It's like conductive glue.

Solder is a metal that melts at a lower temperature than most metals, but that lower temperature is still quite hot! Much hotter than the oven in your kitchen ever gets, so it's important that you are safe when soldering. Take as much care as you would handling hot pots and pans when cooking.



Only solder when an adult is nearby to help!



There are lots of resources online to help you get started soldering if you haven't done it before. YouTube is full of videos, but the tutorials on Sparkfun (<https://learn.sparkfun.com/tutorials/how-to-solder---through-hole-soldering>) and Adafruit (<https://learn.adafruit.com/adafruit-guide-excellent-soldering>) are excellent places to start.



Soldering can get difficult when you feel like you've run out of hands to hold things. You can get a tool called a third hand or helping hand that can help hold things still for you. An alternative is to use a bit of poster putty to hold an item in place while you solder it.

When any paint or glue on your box is dry, you can start laying out your circuit. Before you start, you should decide where your Arduino will be located inside your box.

Here's how you make your LED circuit (see [Figure 2-13](#)):

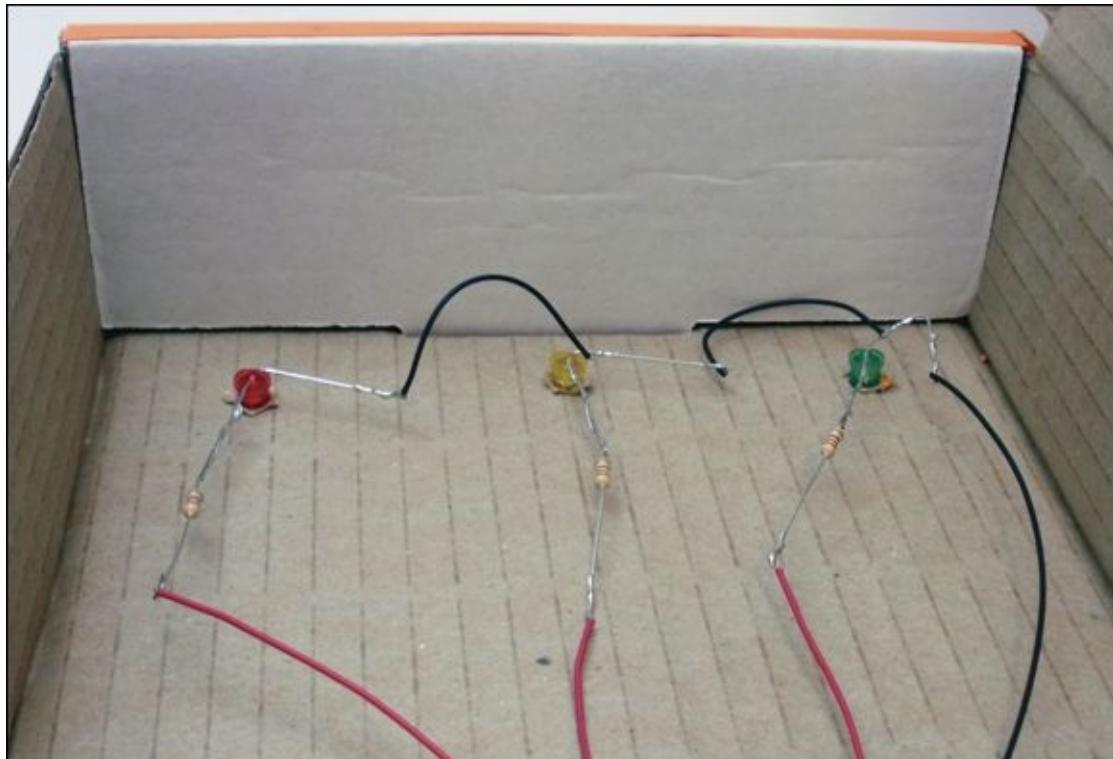
1. Connect a resistor to each of the LEDs. Twist the leg of a resistor with the long leg of an LED so that they don't easily come apart. Solder the connection. Do this will all three LEDs.
2. Place the three LEDs in their holes in the box. Bend the shorter leg (the leg that connects to ground) of the top LED down towards the LED below it. Repeat with the middle LED and bend the shorter leg towards the bottom LED.
3. Cut a piece of wire that reaches from the leg of the top bent LED to the middle bent LED leg and then a second piece of wire that reaches from the middle bent LED leg to the bottom bent LED leg. It's better for the wires to be too long than too short. Cut at least an inch more than you measured. Use wire strippers to strip approximately a  $\frac{1}{2}$ " from each end of the wires.
4. Remove the LEDs from the box. Twist one end of the one of the wires with the short bent leg of the top LED. Solder them together with a soldering iron.
5. Twist one end of the other wire with the short bent leg of the middle LED. Solder them together.
6. Now you will connect the wire connected to the short leg of your top LED to the short leg of the middle LED. Twist the end of the wire hanging from the top LED around the bent leg of the middle LED and solder them together.
7. Repeat with the wire connected to the middle LED to connect it to the bottom LED.
8. Put your newly connected LEDs into their holes in the box to make sure they still fit. If they don't, you can cut or desolder the wires and try again.
9. You now are going to cut the wires that will reach from the LEDs to the Arduino

Uno. You need to measure and cut three wires that reach from the resistors connected to the LEDs and one wire that reaches from the short leg of the bottom LED. Again, cut them about an inch longer than the measurement and strip a  $\frac{1}{2}$ " from each end.

**10.** Solder each of the wires to their resistor or LED.



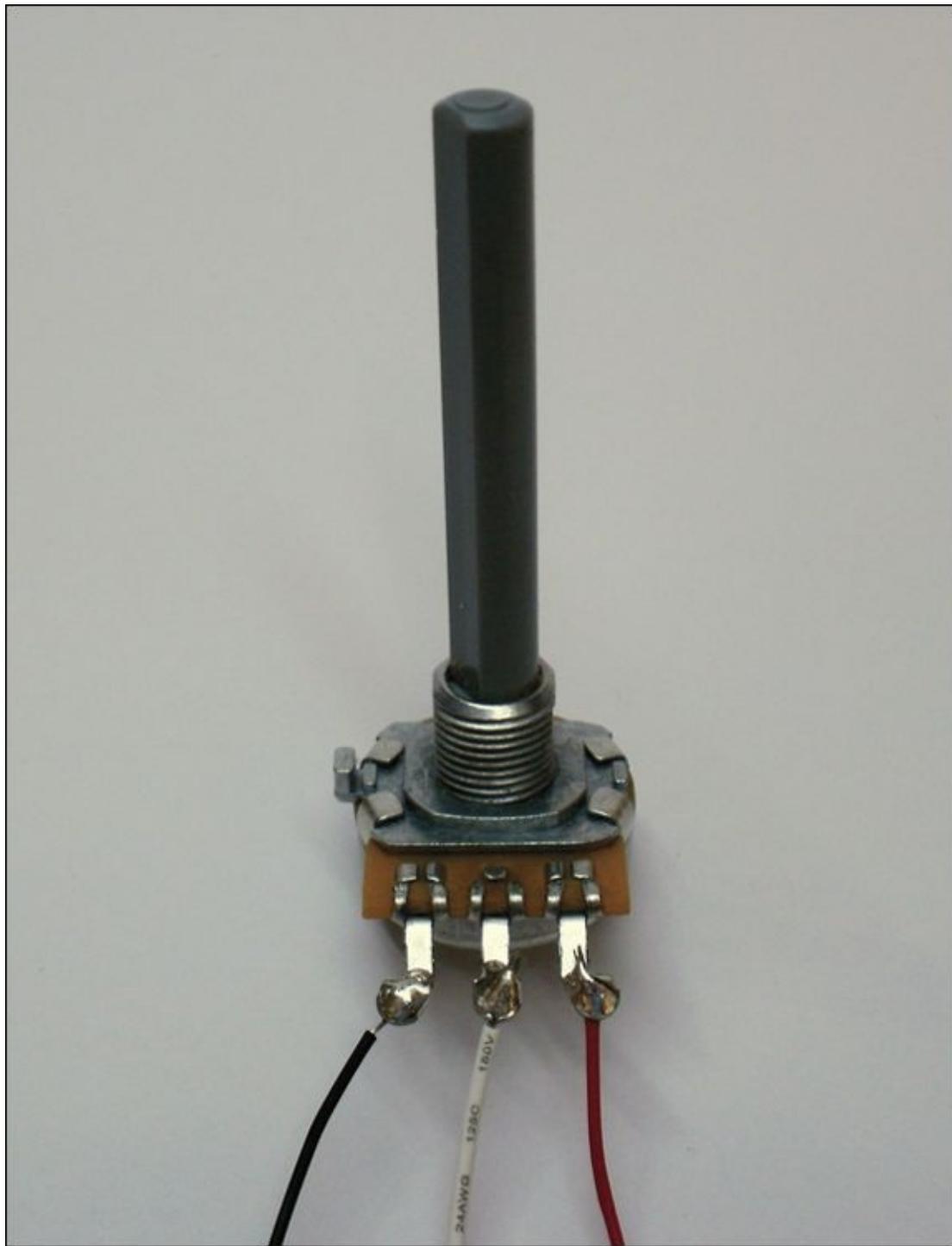
Remember to only solder with an adult. Be careful; the end of the soldering iron is very hot!



**FIGURE 2-13** The LED portion of the circuit

Now solder the wires to the potentiometer (shown in [Figure 2-14](#)):

1. Place the potentiometer in its hole. Cut three wires that reach from the potentiometer to the Arduino board.
2. Strip about half an inch of the plastic from one end of each of the wires and solder each wire to a leg of the potentiometer.
3. Strip about  $\frac{1}{4}$ " of the plastic from the free ends of each wire. You do this so that they can be inserted into the pins on the Arduino board.



**FIGURE 2-14** Soldered potentiometer

At this point, stop and inspect your work. Carefully check that none of the exposed metal from the wires or component legs can touch each other. If they can, they might create accidental electrical connections. If this is the case, wrap electrical tape around the metal to prevent that happening.

## Inserting the Electronics

When the glue and paint you've used to decorate the box is completely dry, you are ready to finish your sign and install your electronics.

Place the LEDs into their holes in the box. You can use a little glue or tape to hold them in place if you need to.

The potentiometer comes with a washer and nut that screw down on the base of the shaft. Remove these, push the potentiometer through its hole and screw down the washer and nut to hold it firmly on the box. You can make a control knob to attach to the end of the potentiometer if you'd like.

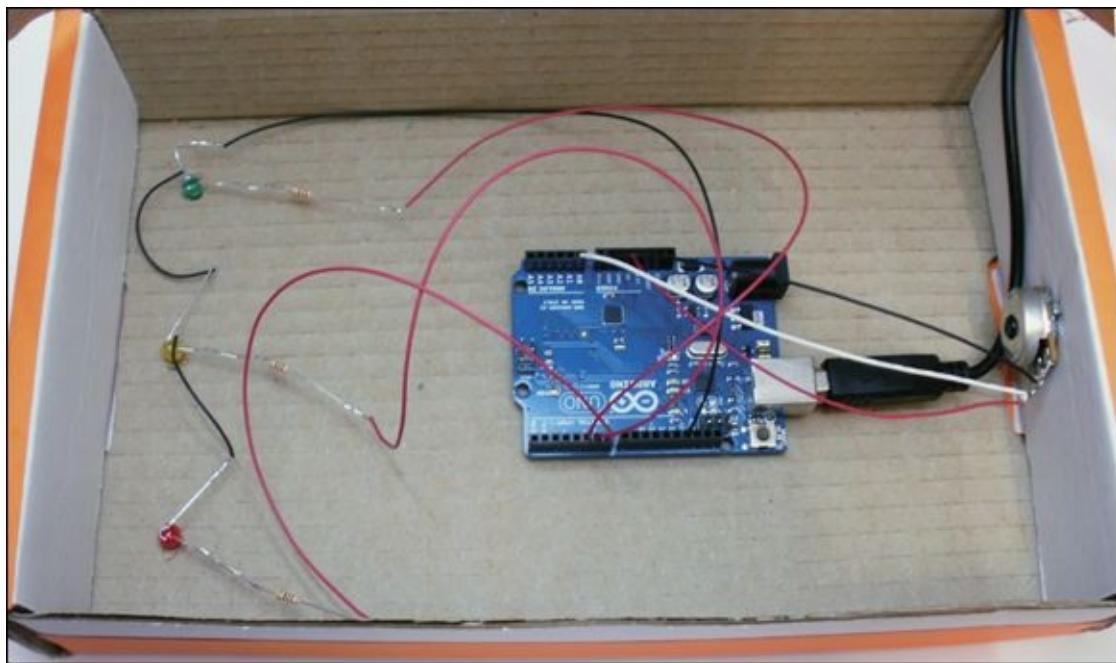
Insert your wires from your components into the Arduino Uno. The three wires connected to the resistors go to Pins 5, 6 and 7. The wire connected to the short leg of the bottom LED is inserted in a GND pin. One of wires connected to an outside leg of the potentiometer is inserted in 5V, and the other outside leg is connected to GND. The remaining wire connected to the middle pin is inserted in Pin A0.

You now need to decide how you want your project to be powered. You can, of course, leave your Arduino Uno connected to your computer, but that can be inconvenient. You can also still use your USB cable, but plug it into a wall adapter instead of your computer, as shown in [Figure 2-15](#). Wall adapters often come with new mobile phones, so you might have one lying around already. Any that lets you connect a USB cable is fine to use.



**FIGURE 2-15** Power supply that you can use with an Arduino board

Congratulations! You have created your own status message sign that you can set up and plug in to display a message of your choice. You've created your first Arduino project that breaks free from the computer and can run on its own. Yours may look very different from the one in [Figure 2-16](#), and that's great! You are well on your way to becoming an Arduino expert!



**FIGURE 2-16** Completed status message sign

# Further Adventures with Arduino

Now that you can change the output of the Arduino Uno according to the turn of a potentiometer, what else could you do? Here are some project ideas:

- Change the speed of a flashing LED by turning the potentiometer.
- Make the LEDs flash in a sequence and change the speed with the potentiometer.

**Arduino Command Quick Reference Table**

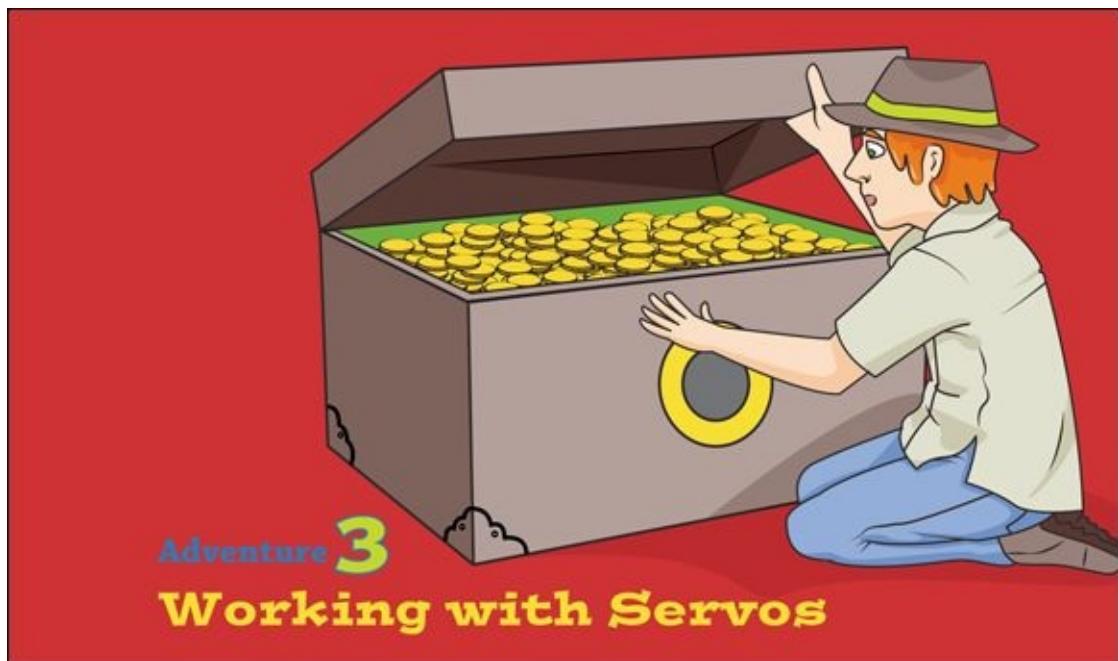
Command	Description
int	Data type that creates a new variable that is an integer (whole number). See also <a href="http://arduino.cc/en/Reference/Int">http://arduino.cc/en/Reference/Int</a> .
Serial.begin()	Starts the serial communication so messages can be sent and received. See also <a href="http://arduino.cc/en/Serial/Begin">http://arduino.cc/en/Serial/Begin</a> .
Serial.print()	Sends a message without a newline at the end. See also <a href="http://arduino.cc/en/Serial/Print">http://arduino.cc/en/Serial/Print</a> .
Serial.println()	Sends a message with a newline at the end. See also <a href="http://arduino.cc/en/Serial/Println">http://arduino.cc/en/Serial/Println</a> .
analogRead()	Reads in the voltage on the specified pin and assigns it a number from 0 (for ground) to 1023 (for 5V). See also <a href="http://arduino.cc/en/Reference/AnalogRead">http://arduino.cc/en/Reference/AnalogRead</a> .
if()	Used to determine whether a section of code will be executed. See also <a href="http://arduino.cc/en/Reference/If">http://arduino.cc/en/Reference/If</a> .



**Achievement Unlocked:** You are taking charge and making decisions!

## In the Next Adventure

You will start adding motion and controlling a motor in the next project.



### Adventure 3

## Working with Servos

ONE WAY OF making your projects more dynamic is by introducing movement. When you add movement to a project it can feel as if you've actually brought it to life. This adventure will show you how, by working with a servo motor and adding switches to your circuits.

In this adventure, you will start by finding out about the new components you are going to work with, then use those components to build a fantastic combination safe, which only opens when you turn all the potentiometers to their secret positions and push the button. If you know the right combination, the safe opens automatically. The safe is constructed from cardboard, so won't withstand a brute force attack, but it can be used to deter parents from getting inside!

# What You Need

You will be using a new **actuator**, which is a fancy word for an object that takes an electrical signal and then does something in the real world. You have already used one type of actuator in [Adventures 1](#) and [2](#): the LED. It takes electricity and turns it into light. In this adventure, you use a motor that takes electricity and turns it into motion.



An **actuator** translates an electrical signal into a real-world modifies action such as light, sound or movement.

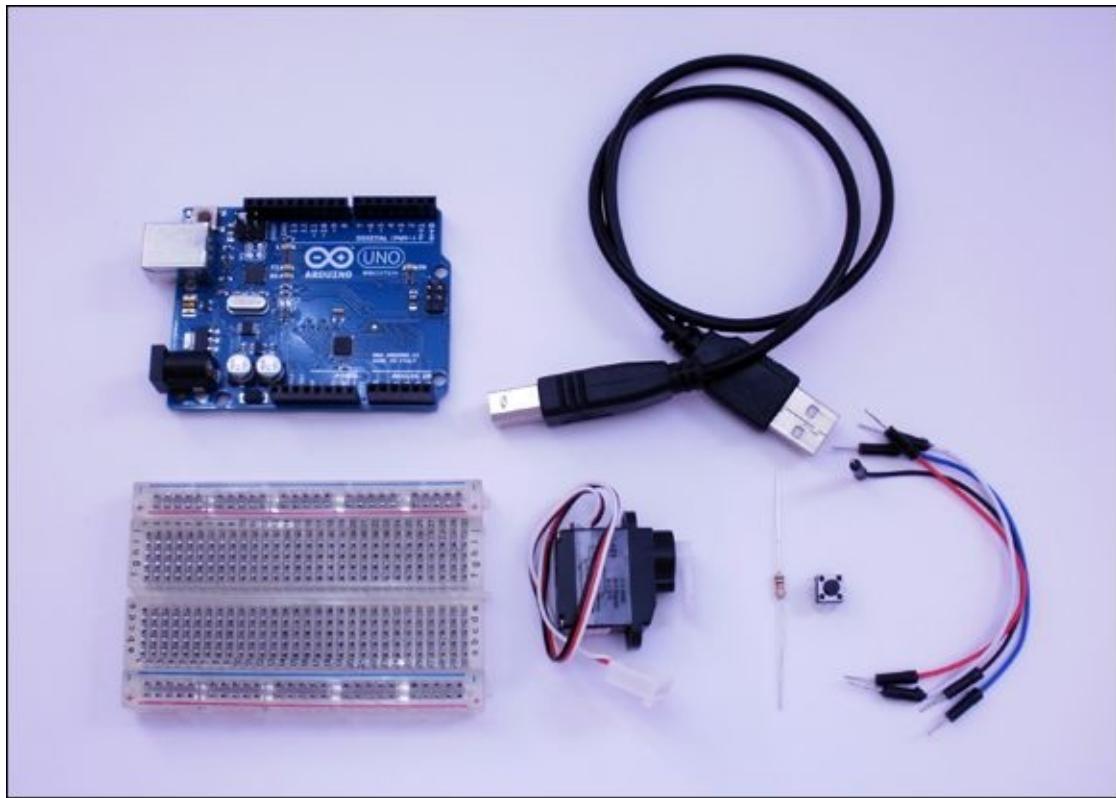
The opposite of an actuator is a **sensor**, and you will be using a new one of those as well. [Adventure 2](#) introduced the potentiometer, which senses rotation and translates it into an electrical signal. Here you use potentiometers again and also use a button to translate a press into an electrical signal.



A **sensor** detects something in the real world such as light, sound or movement, and translates it into an electrical signal.

You need the following items. The electronic components are shown in [Figure 3-1](#).

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 4 jumper wires
- A servo motor
- A tactile push button
- 1  $10\text{k}\Omega$  resistor



**FIGURE 3-1** The electronic components you need for the first part of Chapter 3

# Understanding Different Types of Motors

A *motor* is general term for something that takes electricity and turns it into mechanical movement, but different types of motor let you control that movement in different ways. When you think of a motor, the thing you think of is probably what is called a *DC motor*. The DC in DC motor stands for **direct current**. Direct current is the type of electricity that you use in your Arduino circuits.

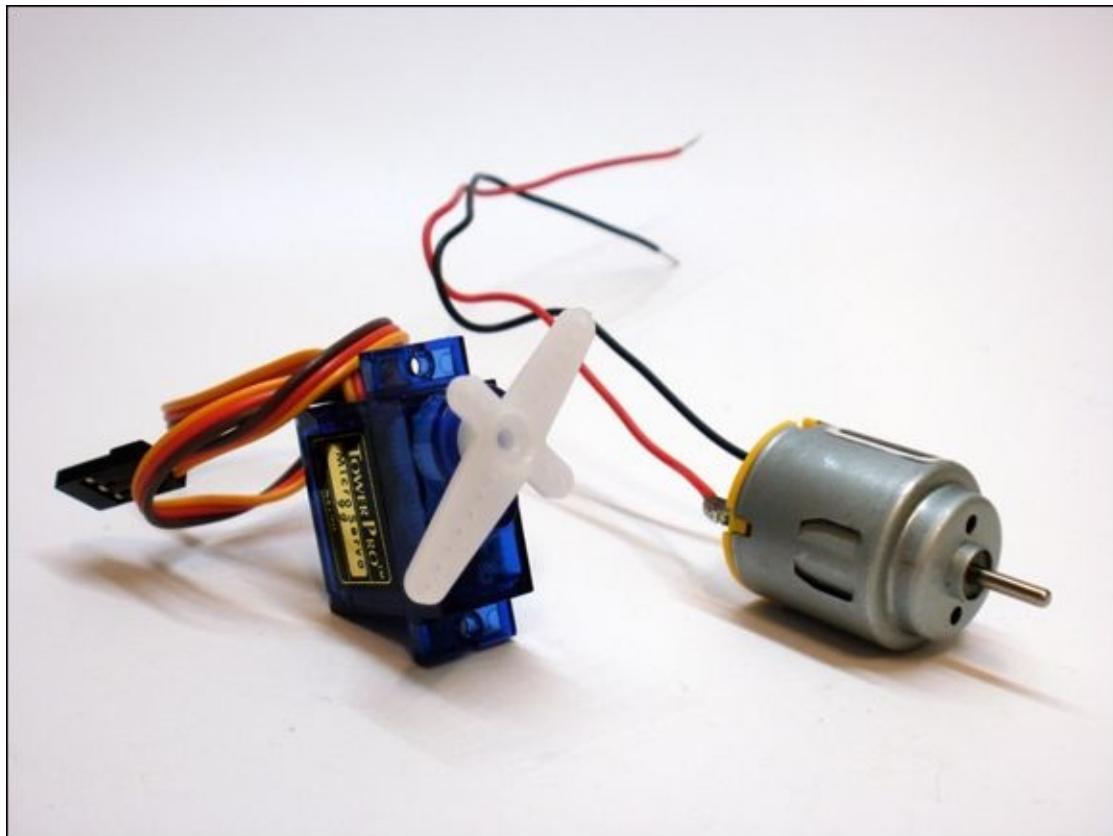


**Direct current (DC)** is the type of electricity used in Arduino circuits. It's the same kind that is generated by a battery and is the opposite of alternating current (AC), which is what comes out of main plugs in the wall.

Toy DC motors are common in things like remote control cars. When a DC motor is connected to DC current, it spins a shaft; you can control the speed of the motor and the direction it spins, but not much else. For more control, you need something that will do more—either a servo motor or a stepper motor.

There are different types of servo motor, but the most common is known as a *standard hobby servo motor*. With a DC motor, the shaft spins, but you don't necessarily know where the shaft is pointing when the motor stops. A servo motor knows which way the shaft is pointing. Although you can tell a servo motor where to point, it has some limitations; it can only point in some directions and can't rotate a full circle. Whereas a DC motor can rotate continuously, a servo motor can usually only rotate 180 degrees.

A *stepper motor* combines the strengths of the DC and servo motors in that it can rotate continuously and you can tell it a precise location to rotate to. But that comes at a price! Stepper motors tend to cost more than other types of motor. There's a solution to this: you can choose the cheaper option of a DC or servo motor for your project (see [Figure 3-2](#)) and get round the limitations by engineering a solution yourself.



**FIGURE 3-2** A servo motor and toy DC motor



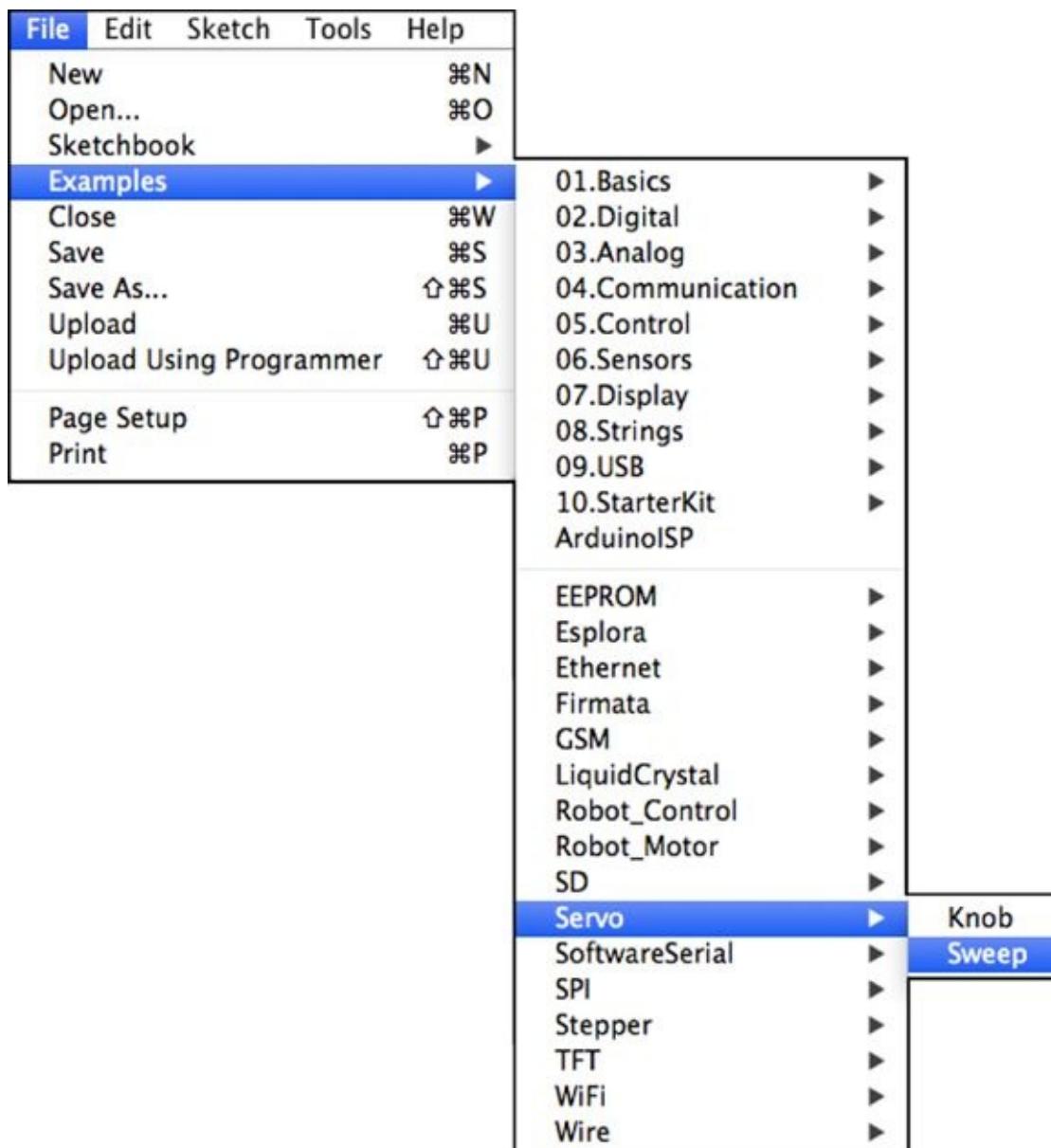
A **servo** is a motor that can be controlled to rotate to a specific position. It usually can't rotate more than 180 degrees.



If you'd like to read more about how to use motors, check out *Making Things Move* by Dustyn Roberts (McGraw-Hill, 2010).

# Controlling a Servo with Arduino

The Arduino integrated development environment (IDE) comes with everything you need to control a servo. It even has example sketches to get you going. In order to control your servo, you need to open a sketch called Sweep. You open Sweep by launching the Arduino IDE and clicking on File ⇒ Examples ⇒ Servo ⇒ Sweep ([Figure 3-3](#)).



[FIGURE 3-3](#) Opening the Sweep example sketch

This sketch shows how to tell a servo to move. Read through the code in the Sweep sketch. The first line of code after the comments hasn't appeared in the code you used in [Adventures 1](#) and [2](#):

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

The `#include` is telling Arduino's compiler that the Sweep sketch will be using some functions that aren't always included in an Arduino sketch, and that the compiler can find those functions in a library called `Servo`. The code tells the compiler that it should read the library file called `Servo.h`. The `< >` around the filename means that the file is located in

the standard location on the computer where all Arduino libraries are stored.

Now look at the next line of code:

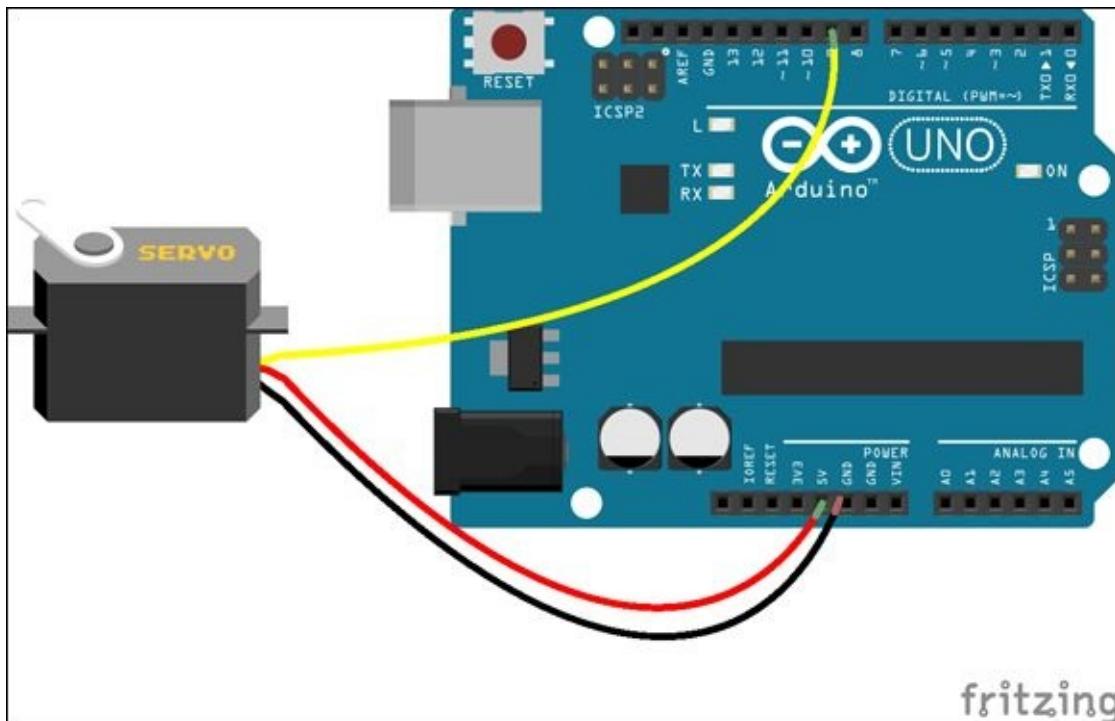
```
Servo myservo;
```

This creates a new variable called `myservo` but this variable isn't an integer like the other variables you've used (such as `greenLED` in [Adventure 2](#)). Instead, it is the type `Servo` (instead of `int`). Because the variable is a `Servo`, it holds all the information needed to communicate with a servo.

There is just one more line of code to finish setting up the servo in the Sweep sketch. The Arduino Uno needs to know which pin the servo will be connected to. This only needs to be done once, so that should happen in the `setup()` function. A few lines down in the sketch you should see the following line:

```
myservo.attach(9);
```

Now you need to build a circuit to hook up your servo motor to your Arduino board, which will end up looking like [Figure 3-4](#). You'll be glad to hear that this needs just three connections: 5V, ground and the controlling pin. Unfortunately the bad news is that not all servo motors make those connections in the same order. Some servos (such as those shown in [Figures 3-1](#) and [3-2](#)) come with a label that nicely illustrates the connections. If yours doesn't, find out if the place where you bought the servo has any information. If that doesn't provide any help, you can just try wiring the circuit in different combinations until it works!



**FIGURE 3-4** Circuit to connect a servo to the Arduino board

Build the circuit for the servo:

1. Use a jumper wire to connect the ground pin (may be labelled GND or 0V) on the servo to any of the GND pins on the Arduino Uno.

2. Use a second jumper wire to connect the 5V pin on the servo to the 5V pin on the Arduino Uno.
3. Use a third jumper wire to connect the remaining pin on the servo to Pin 9 on the Arduino Uno.

After the circuit is built, upload the Sweep sketch (check out [Adventure 1](#) if you haven't done this before), and watch your servo come to life! It should start rotating back and forth. You will hear and see it working.

# Repeating the Same Thing Over and Over

The Arduino board controls the servo motor through electrical pulses that tell it where to rotate. You don't have to worry about how it does that, as the details are nicely handled by the servo library. All you have to do is tell the servo where to go to. You can choose a position for the servo to point anywhere between 0 and 180 degrees.

In the Sweep sketch, the servo rotates its arm back and forth. You could tell the servo to do this by copying and pasting `myservo.write` over and over again, like this:

```
myservo.write(0);
myservo.write(1);
myservo.write(2);
myservo.write(3);
```

That isn't a very efficient way to do things, though. Computers are really good at doing repetitive tasks, so there's a better way to make the servo rotate back and forth. If you've read the code in the Sweep sketch to see how it's done there, you might have noticed a programming tool called a `for` loop.



A `for` loop is a programming device that repeats a block of code for a predetermined number of times.

The Sweep sketch has two `for` loops. Here's the first one:

```
for(pos = 0; pos < 180; pos += 1)
{
    myservo.write(pos);
    delay(15);
}
```

To set up a `for` loop, you need to provide three pieces of information:

1. First of all, you need to state what the starting condition is. In this the Sweep sketch it's `pos = 0`.
2. Next, you need to say what needs to happen for the `for` loop to continue. Here, `pos` has to stay below `180` (expressed in code as `pos < 180`). When `pos` is equal to or larger than `180`, the loop stops and the code in between the `{` and `}` is no longer executed.
3. Finally, you need to say what changes each time the loop is executed. In the Sweep sketch, `1` is added to `pos` each time the code in between `{` and `}` is executed. This is written as `pos+=1`, for short, but you can write it in a number of ways; you could write it as `pos=pos+1` or `pos++`.



The computer doesn't notice indents or spaces between pieces of code. Sometimes code has spaces added to make it easier to read. The following two lines of code look the same to the Arduino.

```
for(pos=0; pos<180;pos+=1)
for ( pos = 0; pos < 180 ; pos +=1 )
```

It's easier to see the three parts of the `for` loop when there are spaces included.

Phew! It's probably time for a recap. In this example, in the `for` loop, `pos` starts at `0`. Because `0` is less than `180`, the code in the `{ }` is executed. The servo is set to `0` and then pauses for 15 milliseconds (by using the `delay()` function). `1` is added to `pos`, so it now equals `1`. Because `1` is less than `180`, the servo is set to `1` and then pauses for 15 milliseconds. This keeps happening until `pos` is `179`. The servo is set to `179` and `1` is added to `pos` making it `180`. `pos` is no longer less than `180`, so the code in the `{ }` is skipped and the Arduino goes onto the next line of code after the `for` loop.

# CHALLENGE



What is happening in the second **for** loop in the Sweep sketch? This is what it looks like:

```
for(pos = 180; pos>=1; pos-=1)
{
myservo.write(pos);
delay(15);
}
```

Change the **for** loop so that the servo only rotates from 0 to 90.

# Digital Input with a Push Button

You might think the simple **switch** would be quite a straightforward electrical component, but in fact switches are deceptively complicated. They come in many shapes and sizes. You have many of them in your house to turn on and off your lights. All they do is complete or interrupt a circuit. Sometimes they change where the current flows in a circuit, but the type of switch that turns your lights on and off is made from two pieces of metal that either touch or don't touch, depending on the position of the switch.



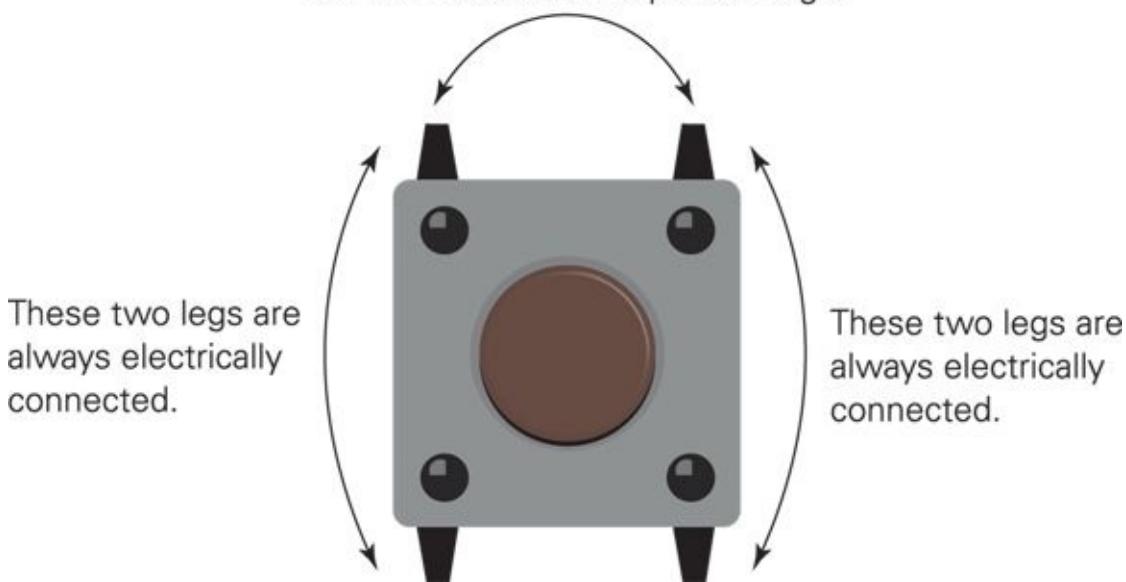
A **switch** is a component that either disrupts or redirects the flow of current in a circuit.

There is another type of switch, called a **tactile push button**. It also has two pieces of metal inside of it, but in this case they only touch when the button is actually being pressed. A tactile push button has four legs, but it's better to think of them as two pairs of legs because the two legs in each pair are always electrically connected—even when the switch is not being pressed. When the button is pressed, all four legs are electrically connected. See [Figure 3-5](#) for an illustration of how a tactile push button works.



A **tactile push button** is a type of switch. A push-to-break push button interrupts the flow of current in a circuit when it is pressed. A push-to-make push button does the opposite by interrupting current only when it is *not* pressed.

When the button is pressed current can flow between the two pairs of legs.



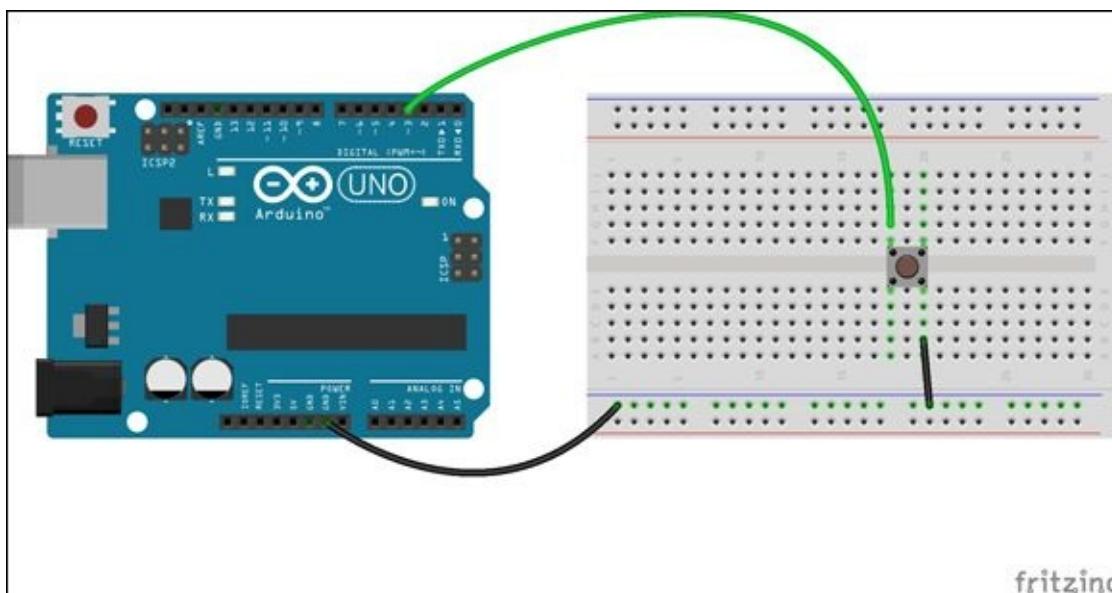
[FIGURE 3-5](#) How a tactile push button works

Now you're going to build the circuit, including a push button, as shown in [Figure 3-6](#).

1. Insert the push button into four rows in the centre of your breadboard. The push

button fits over the gap in the middle, so two legs are inserted in two rows on the top half of the board and the other two legs are in two rows on the bottom half of the board.

2. Use a jumper wire to connect the row where the bottom-right leg of the push button is inserted to one of the long rows along the bottom of the breadboard. If your breadboard is labelled with a black or blue line or a -, connect it to that row. If your breadboard isn't labelled, connect it to either row.
3. Use another jumper wire to connect the long row connected to the push button to one of the GND pins on the Arduino Uno.
4. Use a jumper wire to connect Pin 2 on the Arduino Uno to the row connected to the top-left leg of the push button.



**FIGURE 3-6** Circuit with a tactile push button

Now you've built your circuit, open the example sketch at File ⇒ Examples ⇒ 01.Basics ⇒ DigitalReadSerial. Upload the sketch and open the Serial Monitor by clicking the button in the Arduino IDE or going to Tools ⇒ Serial Monitor.

Ready? Time to press and release the button. What happens in the Serial Monitor? When your finger is pressing the button, you should see a 0 printed; when the button is not being pressed, you should see a mixture of 0s and 1s. The sequence of 0s and 1s is random, so you might see mostly 0s or mostly 1s rather than an even mixture of the two. This is what's called a **floating input**. When the button isn't being pressed, the pin isn't connected to a voltage source such as ground or 5V—it's floating. The Arduino Uno is reading in random values from that pin.



A **floating input** is a pin that is not connected to anything. The pin reads in random values if it is not connected to a voltage source, such as ground, 5V or a sensor.

It's not a good thing to have floating values. The main reason is that when the Arduino

“reads in” a digital signal from a pin, it reads in a 0 when the pin is connected to ground and reads in a 1 when it’s connected to 5V. If the pin isn’t connected to either ground or 5V and is randomly reading in 0 and 1, then it’s impossible for your code to make good decisions based on the input from that pin. If you want to start a motor moving only when a button is pressed, connecting that pin to ground, then you can’t have the pin reading in 0 when the button isn’t pressed.

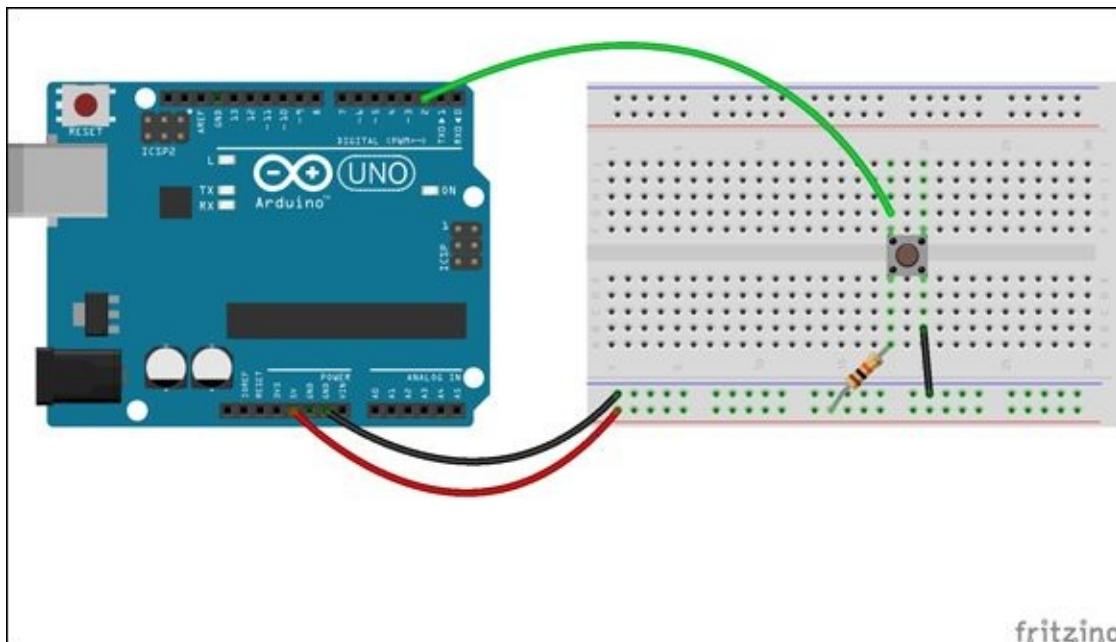
The way around this problem is to use a kind of resistor called a **pull-up resistor**. A pull-up resistor gives a default value of 5V to a pin by always connecting that pin to 5V. The pin is also connected to the push button, and the push button is connected to ground. The resistor usually has quite a high value, such as  $10\text{k}\Omega$ . There is no resistance between ground and the pin when the button is pressed, so the pin connects to ground instead of 5V through the pull-up resistor. Electricity always chooses the path with the least resistance, and, in this case, that is the path between ground and the pin.



A **pull-up resistor** is a resistor that is connected to the high voltage in a circuit, which sets the default state of the pin on that circuit to **HIGH**. The resistor is usually  $10\text{k}\Omega$ .

Like most things with electronics, the effect of a pull-up resistor is much easier to understand when you actually build a circuit and see what happens for yourself. That’s what you’re going to do now. Change the circuit on your breadboard to the one shown in [Figure 3-7](#).

1. Start with the circuit you just built with the push button in the centre of the breadboard.
2. Use a jumper wire to connect from the 5V pin on the Arduino Uno to the other long row along the bottom of the breadboard (the one that isn’t connected to ground).
3. Place one leg of the  $10\text{k}\Omega$  resistor in the same short row as the lower-left leg of the push button. Insert the other leg of the resistor into the long row now connected to 5V.



**FIGURE 3-7** Circuit with a pull-up resistor

You don't need to change anything in your Arduino code, and you can just leave the Serial Monitor open. Now, what happens when you press and release the button? It should now only show 0 when the button is pressed and 1 when it is released.

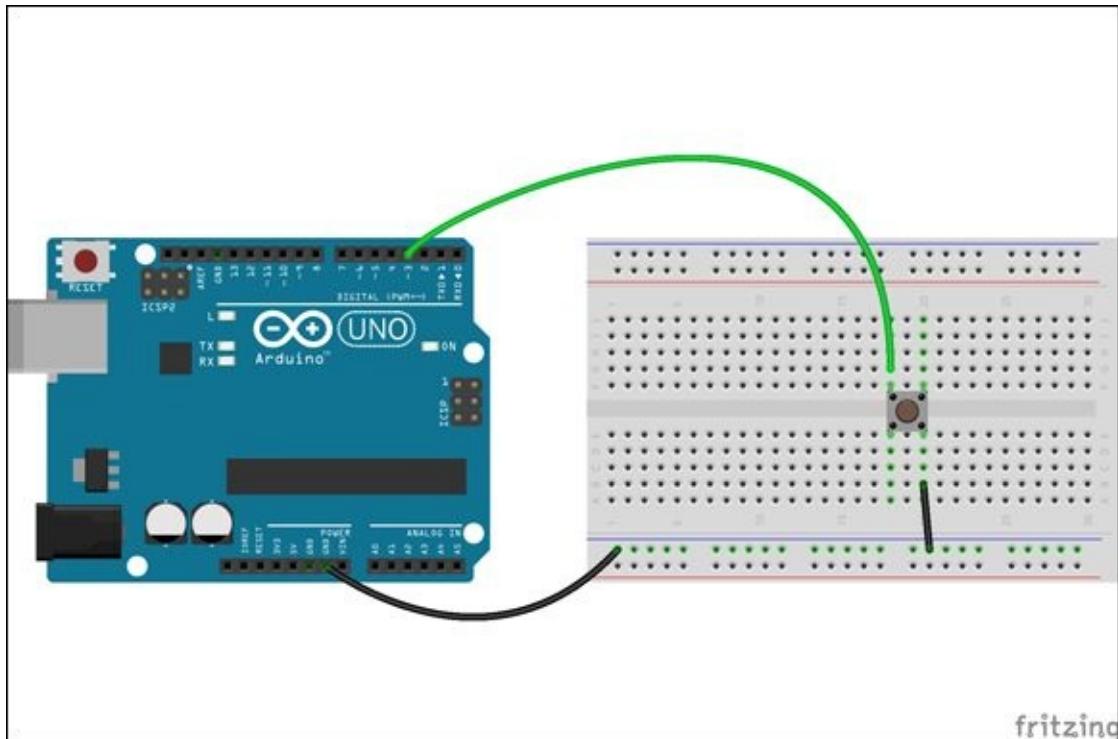
The Arduino board has pull-up resistors built into it already, so you can use these instead of building a pull-up resistor into your circuit on the breadboard. To do this, you first indicate that you want to use one of the built-in pull-up resistors when you set up the `pinMode()` in `setup()`, by typing in the following code:

```
pinMode(pushButton, INPUT_PULLUP);
```

Next, change the DigitalReadSerial sketch so that the second argument of `pinMode` is `INPUT_PULLUP` instead of `INPUT`. Your `setup()` should look like:

```
void setup() {
    // initialize serial communication at 9600 bits per second:
    Serial.begin(9600);
    // make the pushbutton's pin an input:
    pinMode(pushButton, INPUT_PULLUP);
}
```

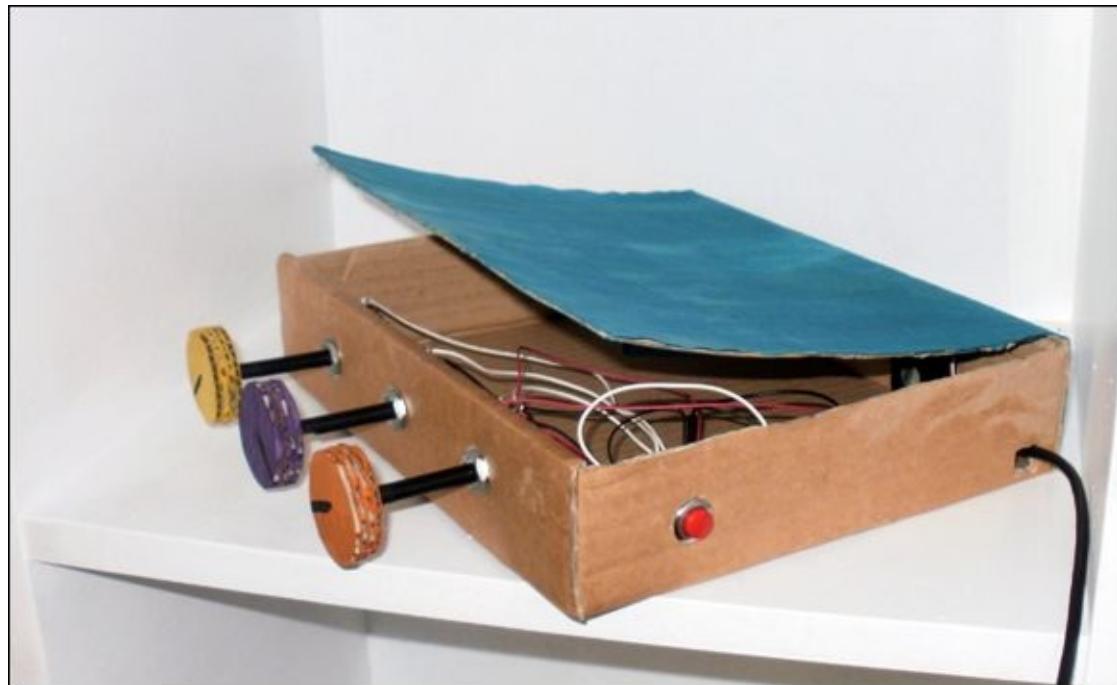
Finally, upload the sketch again and change your circuit on your breadboard to the one in [Figure 3-8](#) by removing the  $10\text{k}\Omega$  resistor and jumper wire connecting 5V and one of the long rows. Your button should act the same way as it did when you had the pull-up resistor on the breadboard.



**FIGURE 3-8** Circuit with a push button and internal pull-up resistor on the Arduino board

# Building a Combination Safe

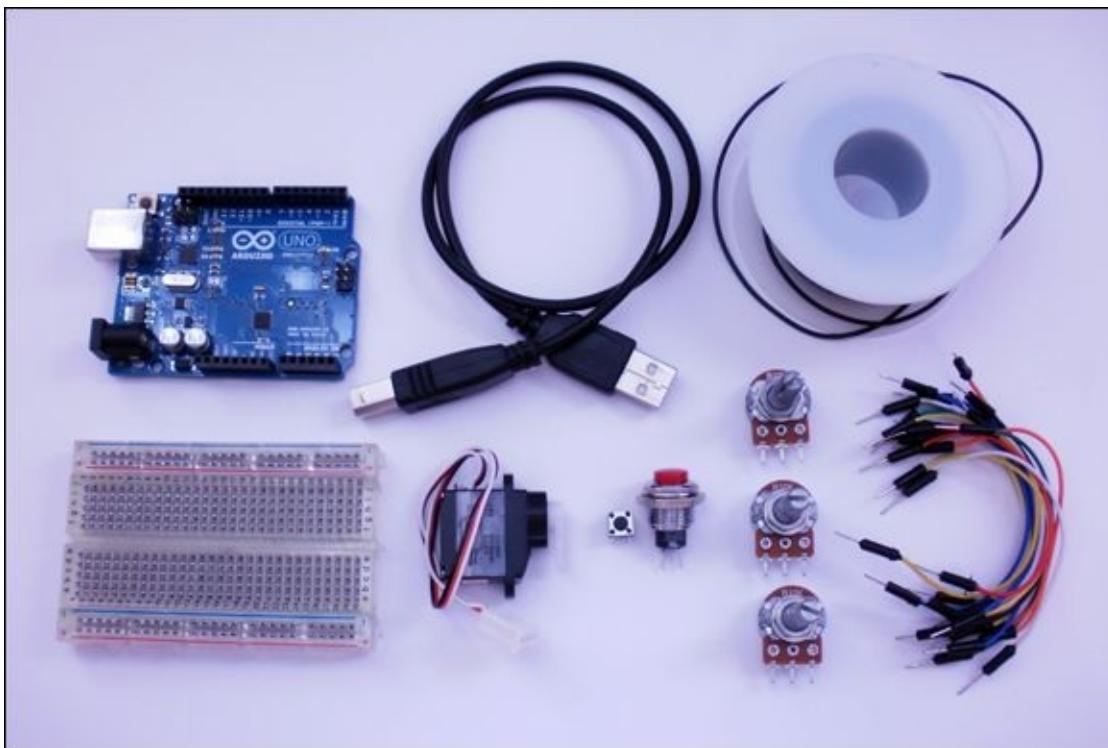
Well done! You have built up quite an arsenal of sensors and actuators. Now you can start putting them together to make something very cool and very useful: a combination safe that opens and closes automatically (see [Figure 3-9](#)). To open the safe, you dial in a combination and push a button. The box will only open if the combination is correct, and it will stay open until you release the button. It can be a great place to keep a secret candy stash or keep your favourite pens and pencils from being “borrowed” without your permission.



[FIGURE 3-9](#) Combination safe

## What You Need

You need the items in the following list to build your safe. It includes the components you need to prototype your circuit on a breadboard and the components you use in your safe. [Figure 3-10](#) shows the electronic components you need.



[FIGURE 3-10](#) The electronic components you need to build your combination safe

You use a different button in your safe than on your breadboard. You use a **panel mount push button** instead of a tactile push button. You still use a tactile push button to test your circuit on a breadboard, but the panel mount button is bigger and easier to mount on a cardboard box. You find out how to connect wires to the panel mount button later in this chapter.



A **panel mount push button** is a push button that is designed to be mounted inside a case. It comes with a nut and washer to secure it to a panel.

Have the following supplies on hand before you start the project:

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 16 jumper wires
- A tactile push button (push-to-make)
- A panel mount push button (push-to-make)

- A servo motor
- 3 10k $\Omega$  potentiometers
- Some solder
- A soldering iron
- Some wire
- A paperclip or bamboo skewer
- A small box with a lid to be your safe
- A hot glue gun
- Scissors or a utility knife

Your box can be any size, but a box approximately the size of a shoebox works well. It works best if the lid is already attached to the base of the box, but the lid isn't attached, I explain how you can attach it yourself.

You can also use anything you would like to decorate your box, such as paint or paper.

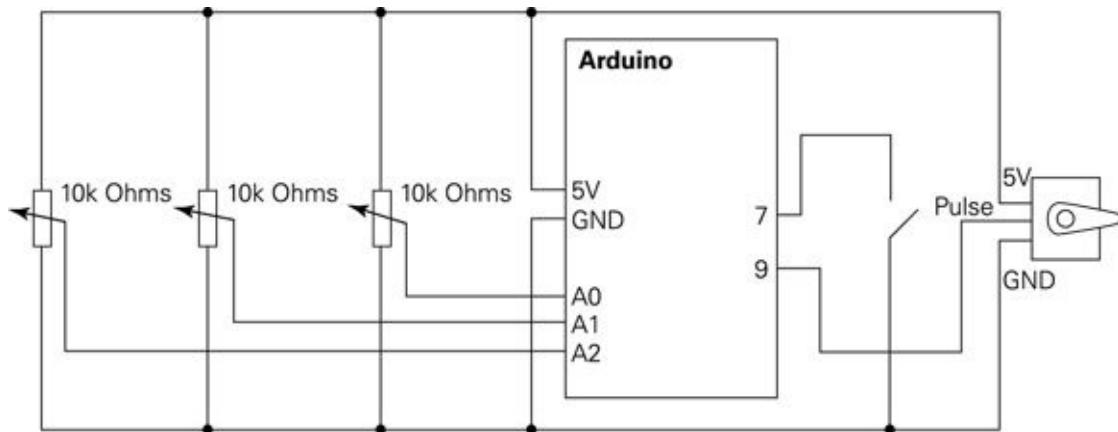


Visit the companion website to see a video showing how the building the safe  
([www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino) ).

## Understanding the Circuit

The circuit for the safe has three components a servo, three potentiometers and a push button. The three potentiometers are read into three analog pins and the push button is read into a digital pin. The servo motor is controlled from another digital pin.

[Figure 3-11](#) shows the schematic for the safe. Looks complicated, doesn't it? Don't worry; you will build it step by step. Before you solder the circuit for your safe, you're going to make a prototype of it on your breadboard.



[FIGURE 3-11](#) Circuit schematic for the combination safe

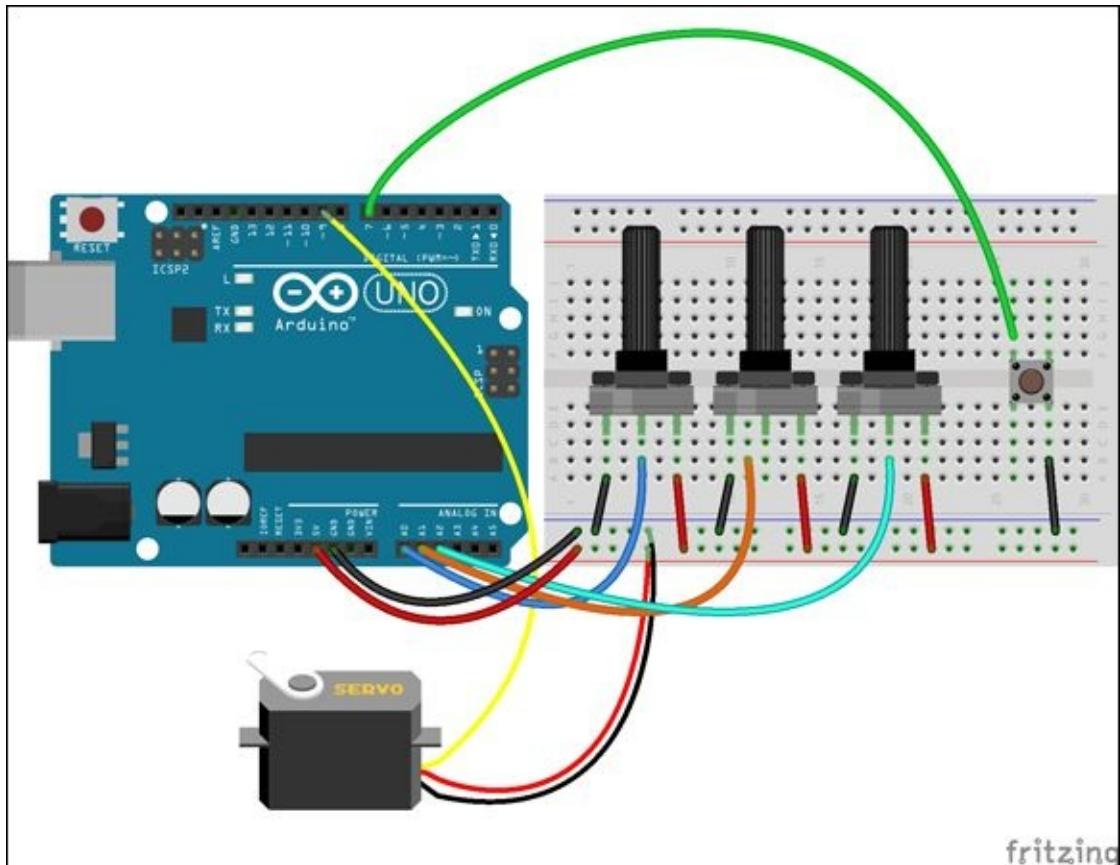


In circuit schematics, lines showing connections often cross over each other. In order to make it less confusing, two lines are electrically connected to each other only when there is a circle over their intersection. Otherwise, they are two independent wires that don't electrically make contact.

## Prototyping on a Breadboard

You should always prototype a circuit on a breadboard before building your final project. It's much easier to fix any errors before you have started cutting wire and soldering connections together! Build the circuit on a breadboard as shown in [Figure 3-12](#).

1. Start by placing the tactile push button on the right of the breadboard. It should fit over the gap in the middle of the breadboard, and each of the four legs should be pushed into the nearest rows.
2. Place three potentiometers evenly across the rest of the breadboard. Each leg of the potentiometers should be in its own row on the breadboard.
3. Use a jumper wire to connect the 5V pin to one of the long rows along the bottom of the breadboard and a second jumper wire to connect a GND pin to the other long row. If your breadboard is labelled (not all are), then connect 5V to the row with a red line or +, and connect GND to the row with a blue or black line or -.
4. Connect the left leg of each potentiometer to the long row connected to GND using three jumper wires.
5. Connect the right leg of each potentiometer to the long row connected to 5V using three jumper wires.
6. Connect the lower-right leg of the push button to GND.
7. Use three jumper wires to connect the middle pin of each potentiometer to Pins A0, A1 and A2.
8. Use a jumper wire to connect the top-left leg of the push button to Pin 7.
9. Make the three connections for the servo. Connect the ground pin of the servo to the long row on the breadboard connected to GND, connect the 5V pin of the servo to the long row connected to 5V and connect the remaining pin on the servo to Pin 9 on the Arduino Uno.



**FIGURE 3-12** Circuit for the combination safe

## Writing the Code

Just like the circuit, the sketch for the safe looks complicated at first. But after you build it up, step-by-step, you will see that sketch is just the combination of smaller sections of code. The code detects whether the button is being pressed. If it is, the Arduino Uno checks whether the three potentiometers are turned to the correct values to open the box. If they are, then the box opens; if they aren't, then nothing happens.

Start by launching the Arduino IDE. Start a new sketch with an empty `setup()` and `loop()`:

```
void setup() {  
}  
void loop() {  
}
```

At the very top of your sketch before `setup()`, add the following lines:

```
#include <Servo.h>  
  
// Pins  
int potPin1 = A0;  
int potPin2 = A1;  
int potPin3 = A2;  
int buttonPin = 7;  
int servoPin = 9;  
  
// other variables  
int open1 = 0;  
int open2 = 1023;  
int open3 = 0;  
int range = 10;  
int boxOpen = 0;  
Servo servo;
```

The first line imports the library to control the servo and the rest of the lines are variables. The first group (under `// Pins`) are the variables to keep track of which pins are connected to the sensors and actuator.

The next five variables are for controlling the box. The variables `open1`, `open2` and `open3` are the values that the potentiometers need to be turned to in order to open the box. Because it can be difficult to turn the potentiometer to a precise number (especially when you aren't using Serial Monitor to see the exact values from the potentiometers), the `range` variable is used to determine how close you have to be to the right number. For example, any value from `open2-range` through to `open2+range` registers the second potentiometer as being in the correct position. The larger the number stored in `range`, the easier it is to open the box.

The `boxOpen` variable is used to keep track of whether the box is opened or closed. The box starts closed, so the variable is set to `0`. When it is opened, it is set to `1` and then changed back to `0` when the box is closed. The `loop()` function holds the code that

controls `boxOpen`.

The last variable is a familiar one: `servo`. It is the variable that communicates with the servo.

The next step is to add the code to the `setup()`:

```
// set button pin to be an input with
// with pull-up resistor
pinMode(buttonPin, INPUT_PULLUP);

// attach servo to pin
servo.attach(servoPin); // attaches the servo on pin 9
                        // to the servo object
servo.write(90); // start with the box closed
Serial.begin(9600); //start serial communication
```

The first line of `setup()` sets the `pinMode()` for the push button and turns on the internal pull-up resistor. The rest of the function attaches the servo to its pin, makes sure the servo has closed the box and then starts serial communication.

Finish your sketch by adding the following to the `loop()`:

```
// check if button is pressed
int buttonValue = digitalRead(buttonPin);
// if button is pressed and box is closed
if(buttonValue == 0 && boxOpen == 0) {
    // button is pressed
    int potValue1 = analogRead(potPin1);
    int potValue2 = analogRead(potPin2);
    int potValue3 = analogRead(potPin3);

    Serial.print("pot 1: ");
    Serial.print(potValue1);
    Serial.print(" pot 2: ");
    Serial.print(potValue2);
    Serial.print(" pot 3: ");
    Serial.println(potValue3);

    // if all values are within correct range
    if(potValue1 < (open1+range) &&
       potValue1 > (open1-range) &&
       potValue2 < (open2+range) &&
       potValue2 > (open2-range) &&
       potValue3 < (open3+range) &&
       potValue3 > (open3-range)
    ) {

        // open the box
        Serial.println("opening");
        for(int pos = 90; pos > 0; pos -= 1)
        {
            servo.write(pos);
            delay(15);
        }
        boxOpen = 1;
    }
}
```

```

}

// if button is pressed and box is open
if(buttonValue==1 && boxOpen==1) {
  Serial.println("closing ");

  // close the box
  for(int pos = 0; pos < 90; pos+=1)
  {
    servo.write(pos);
    delay(15);
  }
  boxOpen = 0;
}

```

You now have a complete sketch and you are ready to check whether your circuit is working correctly. Upload your sketch to the Arduino Uno and open the Serial Monitor. Turn the potentiometers until they match the values stored in `open1`, `open2` and `open3`, and then push and hold the button. The servo should rotate and stop. Release the button and the servo should return to its starting position.

The Digging into the Code section goes through the loop in more detail to explain how your safe functions. Here's the full sketch:

```

#include <Servo.h>

// Pins
int potPin1 = A0;
int potPin2 = A1;
int potPin3 = A2;
int buttonPin = 7;
int servoPin = 9;

// other variables
int open1 = 0;
int open2 = 1023;
int open3 = 0;
int range = 10;
int boxOpen = 0;
Servo servo;

void setup() {
  // set button pin to be an input with
  // with pull-up resistor
  pinMode(buttonPin, INPUT_PULLUP);

  // attach servo to pin
  servo.attach(servoPin); // attaches the servo on pin 9 to the servo
  object
  servo.write(90); // start with the box closed
  Serial.begin(9600); //start serial communication
}

void loop() {
  // check if button is pressed
  int buttonValue = digitalRead(buttonPin);

```

```
// if button is pressed and box is closed
if(buttonValue == 0 && boxOpen == 0) {
    // button is pressed
    int potValue1 = analogRead(potPin1);
    int potValue2 = analogRead(potPin2);
    int potValue3 = analogRead(potPin3);

    Serial.print("pot 1: ");
    Serial.print(potValue1);
    Serial.print(" pot 2: ");
    Serial.print(potValue2);
    Serial.print(" pot 3: ");
    Serial.println(potValue3);

    // if all values are within correct range
    if(potValue1 < (open1+range) &&
       potValue1 > (open1-range) &&
       potValue2 < (open2+range) &&
       potValue2 > (open2-range) &&
       potValue3 < (open3+range) &&
       potValue3 > (open3-range)
    ) {

        // open the box
        Serial.println("opening");
        for(int pos = 90; pos > 0; pos -= 1)
        {
            servo.write(pos);
            delay(15);
        }
        boxOpen = 1;
    }
}
// if button is pressed and box is open
if(buttonValue==1 && boxOpen==1) {
    Serial.println("closing ");

    // close the box
    for(int pos = 0; pos < 90; pos+=1)
    {
        servo.write(pos);
        delay(15);
    }
    boxOpen = 0;
}
}
```

# DIGGING INTO THE CODE



Let's look at the `loop()` of the code you've just input in a little more detail.

The value of the `buttonPin` is read in. If the value is 0 and the box is closed, then the values of each of the potentiometers are read:

```
// check if button is pressed
int buttonValue = digitalRead(buttonPin);

// if button is pressed and box is closed
if(buttonValue == 0 && boxOpen == 0) {
    // button is pressed
    int potValue1 = analogRead(potPin1);
    int potValue2 = analogRead(potPin2);
    int potValue3 = analogRead(potPin3);
```

The value of each potentiometer is printed to the Serial Monitor to help with any debugging:

```
Serial.print("pot 1: ");
Serial.print(potValue1);
Serial.print(" pot 2: ");
Serial.print(potValue2);
Serial.print(" pot 3: ");
Serial.println(potValue3);
```

If each potentiometer is within range of the correct value:

```
// if all values are within correct range
if(potValue1 < (open1+range) &&
   potValue1 > (open1-range) &&
   potValue2 < (open2+range) &&
   potValue2 > (open2-range) &&
   potValue3 < (open3+range) &&
   potValue3 > (open3-range)
) {
```

then the box is opened by using a `for` loop to rotate to the 0 position. You know if the box is closed if `boxOpen` is 0. After the box is open, `boxOpen` gets set to 1 so that you have confirmation that the box is open.

```
// open the box
Serial.println("opening");
for(int pos = 90; pos > 0; pos -= 1)
{
    servo.write(pos);
    delay(15);
}
boxOpen = 1;
}
```

If the value of the `buttonPin` is 1 and the box is open, the box is closed by using a `for` loop to rotate the servo to position 90. The `boxOpen` variable is then set to 0.

```
// if button is pressed and box is open
if(buttonValue==1 && boxOpen==1) {
    Serial.println("closing ");
```

```
// close the box
for(int pos = 0; pos < 90; pos+=1)
{
    servo.write(pos);
    delay(15);
}
boxOpen = 0;
}
```

If the button is pressed while the box is already open, or the button is released while the box is already closed, then nothing is done and the `loop()` is repeated.

## CHALLENGE

Set your secret combination to open the safe using the `open1`, `open2` and `open3` variables. Adjust how easy it is to dial in the numbers using `range`.

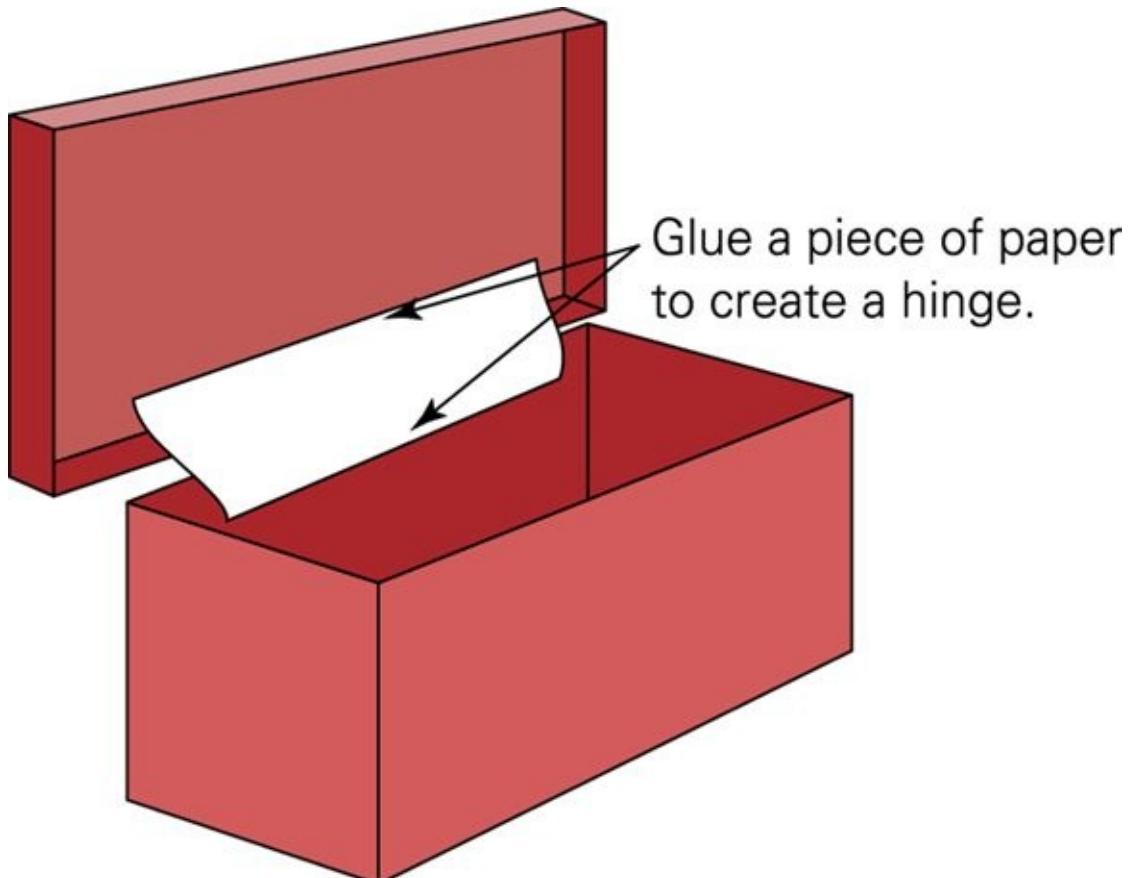
## Making the Safe

At last, you're ready to make your safe! This is very similar in construction to the status message sign in [Adventure 2](#). You're going to use a box (such as a shoebox) to house the electronics.

1. Attach the lid to the box. When you attach the lid, make sure you attach it along one side so that it hinges open and shut. That way, the servo motor can dramatically push the lid up to open the box without the lid falling off. One way of doing this is to make a paper hinge with a strip of paper and glue, as shown in [Figure 3-13](#).
2. Next, you need to decide where you want to put your potentiometers and button. This is entirely up to you, although you probably want them to be on the front of the box for easy access. Cut holes so that the shafts of the potentiometers and button fit snugly. Cut a hole that can pass the USB cable into the box to power the Arduino Uno.
3. Servo motors come with a selection of different arms. These pop onto the end of the rotating shaft of the servo motor. You want to use the one that is a single arm extending from the shaft. (Don't use the cross arm.)

The arm isn't very long, so you can extend it by attaching another object to it. You can use anything you like, but a bamboo skewer or paperclip works well. Glue the object to the servo arm and make sure it's firmly attached (see [Figure 3-14](#)).

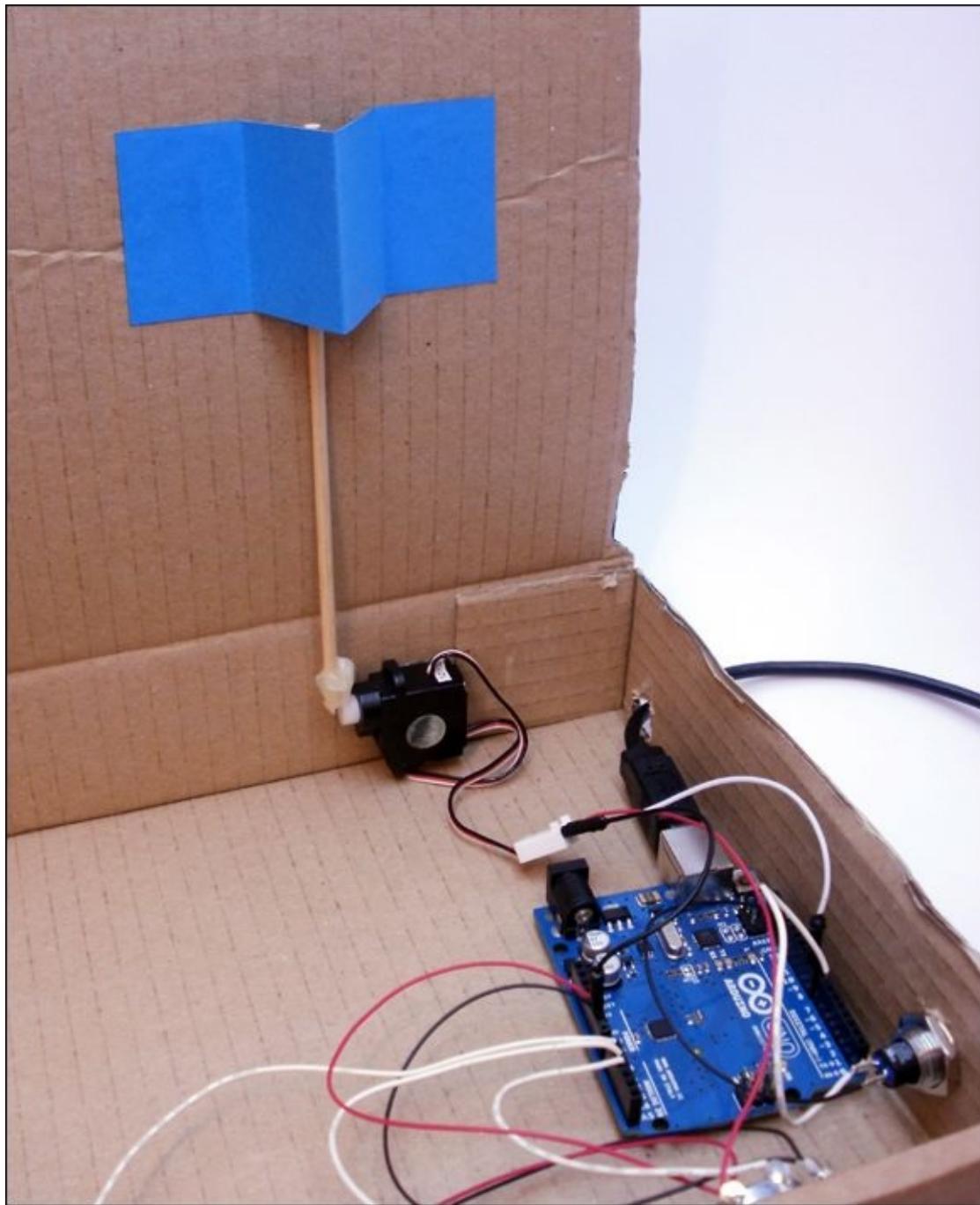
4. Upload the sketch for the safe onto the Arduino Uno and set up the prototype circuit on the breadboard if you haven't done so already.
5. Remove any arm attachments from the servo. If the code is running on the Arduino board and you aren't pressing the button, the servo should be rotated to the 90 position. Now attach the servo arm so that it is at 90 degrees—position it so that it won't push the lid of the box up. When you dial the correct combination and press the button, the servo arm should rotate to point straight up.
6. The extended arm of the servo pushes up the closed lid, but how does it close it again after it's open? Create a paper loop that is attached to the underside of the lid as in [Figure 3-15](#). The extended arm of the servo goes in this loop and uses it to pull the lid closed.



**FIGURE 3-13** If the lid is not already attached to your box, add a paper hinge.



**FIGURE 3-14** Extend the servo's arm by attaching an object like a paperclip or bamboo skewer.



**FIGURE 3-15** Paper loop so the servo can close the safe

# Soldering the Wires

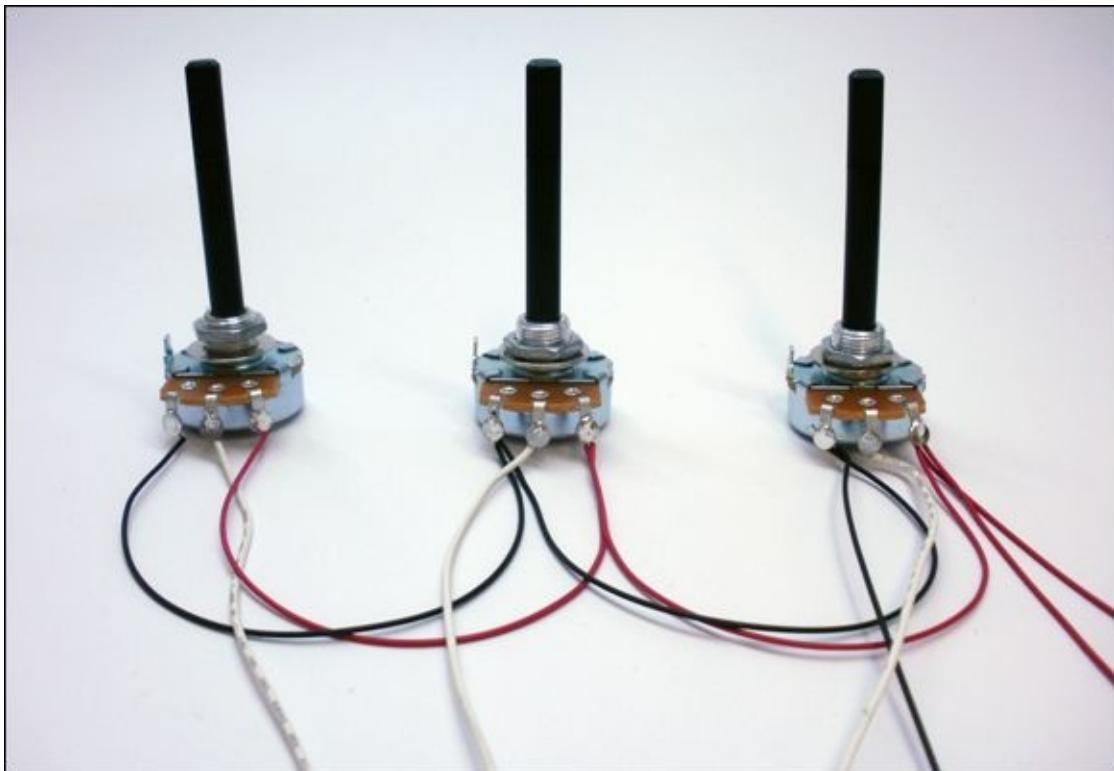
Use the following steps to solder the wires:

1. Place the potentiometers in their holes in the box. Measure and cut four pieces of wire that reach from the potentiometer farthest from the Arduino Uno to the next nearest potentiometer. Cut them about an inch longer than you need. Strip about  $\frac{1}{2}$ " from the end of each wire. Repeat and cut a wire that reaches from the middle potentiometer to the one closest to the Arduino Uno (see [Figure 3-16](#)). These wires connect the outside legs of the potentiometers to each other.
2. Cut two pieces of wire that reach from the outside legs of the potentiometer closest to the Arduino Uno to the 5V and GND pins on the board. Cut them about an inch longer than you need and use wire strippers to strip about  $\frac{1}{2}$ " from each end of the wires.
3. Cut a piece of wire that reaches from the closest potentiometer to the connector on the servo. Cut it about an inch longer than you need and use wire strippers to strip about  $\frac{1}{2}$ " from each end of the wires.



Remember to solder only with adult supervision. Visit the companion site for videos about how to solder ([www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino)).

4. Solder the outside legs of the two potentiometers farthest from the Arduino Uno to each other using the wires as shown in [Figure 3-16](#).
5. Solder the wires for the potentiometer closest to the Arduino Uno. One of the outside legs will have two wires soldered to it—one from the middle potentiometer and a wire that connects to the Arduino Uno. The other outside legs of the potentiometer will have three wires soldered to it—the remaining wire from the middle potentiometer, a wire that connects to the Arduino Uno and a wire that connects to the servo.
6. Measure and cut three pieces of wire that reach from each of the potentiometers to Pins A0, A1 and A2 on the Arduino Uno. Cut them each about an inch longer than you need and strip about  $\frac{1}{2}$ " from the end of each wire.
7. Solder one end of each wire the middle leg of each potentiometer.
8. Place the panel mount push button in its hole. Measure and cut two pieces of wire that reach from the push button to Pin 7 and a GND pin on the Arduino Uno. Cut the wire about an inch longer than you need and strip about  $\frac{1}{2}$ " from the end of each wire.
9. Solder one wire to one leg of the push button and the other wire to the other leg.



**FIGURE 3-16** Soldered components

To summarise:

- You will have three wires coming from the middle leg of each potentiometer that will eventually connect to the Arduino Uno.
- The potentiometer farthest from the Arduino Uno will have one wire connected to each outside leg that connects them to outside legs of the next potentiometer.
- The middle potentiometer will have two wires connected to each outside leg: a wire connecting that leg to the first potentiometer and a wire connecting it to the last potentiometer.
- The last potentiometer will have two wires connected to one outside leg and three wires connected to the other outside leg. The leg with two wires will be connected to the middle potentiometer and the remaining wire will eventually connect to the Arduino Uno. The leg with three wires is connected to the middle potentiometer, to a wire that will connect to the servo and to a wire that will connect to the Arduino Uno.
- The push button has one wire connected to each leg. These will eventually connect to the Arduino Uno.



Every box will be a little different. The sizes will be different, and lids will be looser or tighter. You may have to be creative to solve engineering problems so that your box opens and closes. For example, you might need to attach a paper loop on the underside of the lid so that the servo arm catches it and pulls the lid down (refer to [Figure 3-15](#)).

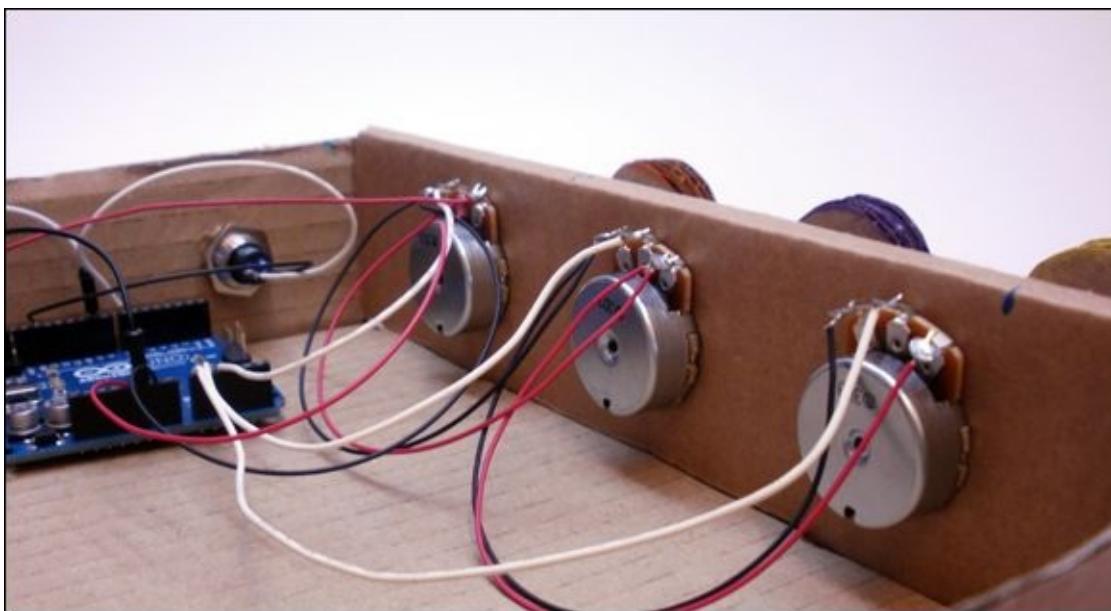
Why not try making your own paper box? Find an origami book in your library or look online for a box and lid pattern.

## Inserting the Electronics

The potentiometers and push button come with nuts and washers that thread onto the base of the shafts. Remove the nuts and washers and then stick the stems of the potentiometers and push button through the holes you've made in your box. Screw the washers and nuts back on to secure the components to the cardboard. Add some glue if they still wiggle around more than you'd like.

Inside the box, build your circuit by connecting the wires to their respective pins on the Arduino Uno. One of the wires soldered to an outside pin of a potentiometer with three wires soldered to it should be inserted in 5V; the other one from the leg with two wires soldered to it should be inserted in a GND pin. Connect the three wires from the three potentiometers to Pins A0, A1 and A2. Connect one wire from the push button to a GND pin and the other wire to Pin 7. Connect the last wire from the potentiometer leg with three wires soldered to it to the 5V connection on the servo. Use two jumper wires to connect the servo to GND and in 9.

Go ahead and test it out! You now have a box with a secret code that automatically opens. [Figure 3-17](#) shows the completed wiring.



**FIGURE 3-17** Completed combination safe

# Further Adventures with Arduino

Now that you have some servo experience under your belt, check out these projects:

- <http://playground.arduino.cc/ComponentLib/Servo>
- <http://arduino.cc/en/pmwiki.php?n=Tutorial/Knob>

Working with servos and Arduino is just the beginning of what you can do in the field of robotics. Check out these amazing robots—many built using Arduino!

- <http://artbots.org/2011/participants/>
- <http://makezine.com/projects/building-a-simple-arduino-robot/>
- <http://www.makershed.com/collections/robotics>

## Arduino Command Quick Reference Table

Command	Description
#include	Command to import a library. See also <a href="http://arduino.cc/en/Reference/Include">http://arduino.cc/en/Reference/Include</a> .
Servo.h	Library to control a servo. See also <a href="http://arduino.cc/en/reference/servo">http://arduino.cc/en/reference/servo</a> .
Servo	Object for controlling a servo. See also <a href="http://arduino.cc/en/reference/servo">http://arduino.cc/en/reference/servo</a> .
Servo.attach()	Attach a Servo variable to the specified pin. See also <a href="http://arduino.cc/en/Reference/ServoAttach">http://arduino.cc/en/Reference/ServoAttach</a> .
Servo.write()	Write a value to the servo to tell it what position to move to. See also <a href="http://arduino.cc/en/Reference/Servowrite">http://arduino.cc/en/Reference/Servowrite</a> .
for	Loops over a section of code a certain number of times. See also <a href="http://arduino.cc/en/Reference/For">http://arduino.cc/en/Reference/For</a> .



**Achievement Unlocked:** You are successfully combining circuits and code!

## In the Next Adventure...

A few LEDs are good, but lots of LEDs are even better! In the next adventure you find out how to control large batches of LEDs with small circuit chips called shift registers.



**YOU ARE WELL** on your way to becoming an Arduino expert. You've tackled all sorts of things, from motors to potentiometers. You've even handled three potentiometers at the same time. But what about working with more than three of the same thing? One LED is good, three LEDs are better—but how about 24 LEDs?

As an experienced Arduino engineer, you might take a look at your Arduino board and question my counting abilities. There aren't 24 output pins for LEDs on your board? You're right! But you can harness the power of special chips called shift registers to extend the number of outputs, and that's what you're going to do in this adventure.

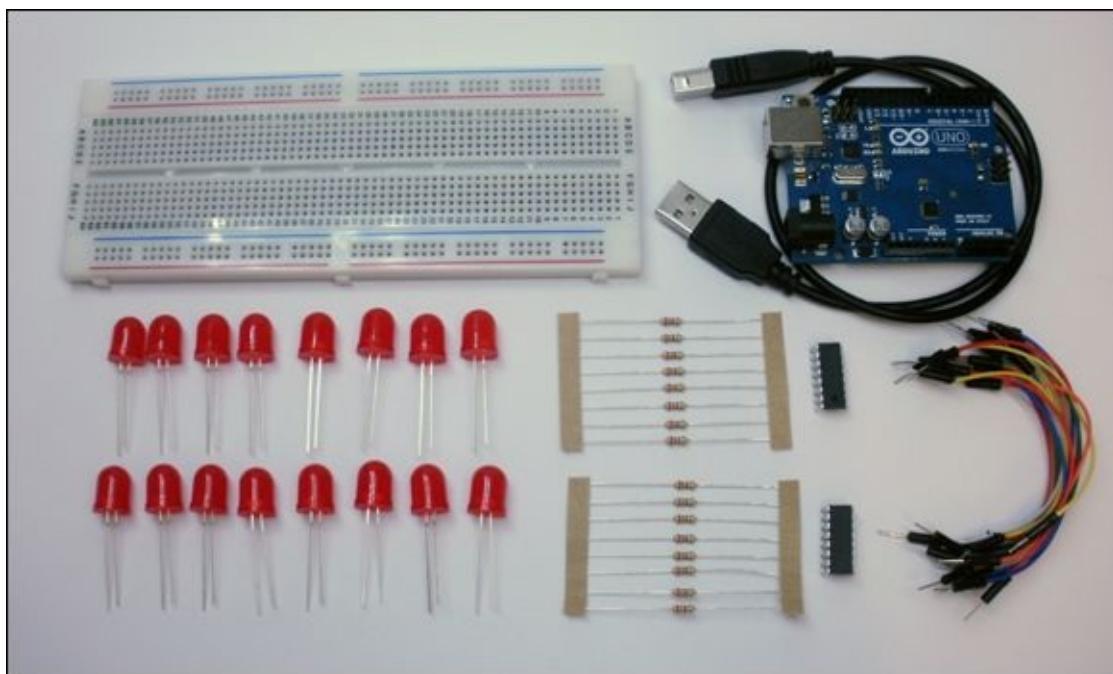
Code can start getting a little messy when you're working with so many outputs, so I'll show you some ways to keep your code tidy and easier to understand.

When you put it all together at the end of the adventure, you will make a carnival-style light-up sign that spells out your name (or any other word you choose).

# What You Need

For the start of this adventure, you need a breadboard, LEDs and resistors. You're going to be exploring different ways to light up a collection of LEDs in code. You then find out what a shift register is and how to use it. You need the following items; the electronic components are shown in [Figure 4-1](#):

- A computer
- An Arduino Uno
- A USB cable
- 1 large breadboard or 2 small ones
- 38 jumper wires
- 16 LEDs
- 16  $220\Omega$  resistors
- 2 74HC595 shift register integrated circuits (ICs)



[FIGURE 4-1](#) What you need for the first part of this adventure



Integrated circuit (IC) names can be quite long and seem complicated, but they are just holding a lot of little pieces of information. For this adventure, you're using the 74HC595 shift register, and that's the set of numbers and letters that you need to look for when you buy the part. If you get a chip that has two letters in the part number before 74HC595, it's okay. These are a code for the company that makes the chip. Chips like shift registers are made by lots of different companies, so you don't need to worry if the chip you are thinking about buying has these two extra letters. As long as the chips you buy has the next set of numbers and letters (74HC595), what you have is good. There may be an additional last letter tells you what shape or package the chip is. For breadboard circuits, you want it to be N for a DIP (a package with two rows of legs that fits into a breadboard). The section "Getting More Outputs with Shift Registers" explains more about what that means.

# Organising Your Code

Code is simply written instructions that a computer can understand. Often, you have to repeat those instructions. In such cases, you can save time by copying and pasting the same piece of code multiple times to get the computer to repeat the same set of instructions. But what if you make a small typo? Maybe you miss a semicolon? The tiniest mistake can lead to your program not working properly. It can be hard to figure out why it works the first two times and then fails the third time. Long sections of repeated code can also make it more difficult to follow what is happening in your sketch. Your code becomes less readable.

Programmers like to joke that they are lazy and don't want to do more work than necessary! So computer scientists who write programming languages spend a lot of time designing the way instructions are written out, to help minimise the risk of making simple mistakes when doing things like copying and pasting code. The following sections introduce you to some of the techniques you can use to simplify your code.

# Using Functions

One easy way to repeat code is to put the lines of code you want to be repeated into something called a function. It's like giving a name to a set of instructions. You then only need to write out the name of the group of instructions each time you want them to happen, instead writing all of the instructions individually.

If you've worked through the earlier adventures, you have already been using functions written by someone else. For example, `digitalWrite()` is a function that controls a Digital Pin on the Arduino Uno. The function handles all the details of turning on and off the pin; you just have to call the function. Now you get to start writing your own functions.



You're going to add functions to the `Blink` sketch in this adventure. You might recall that I've talked about functions before—in particular, the `setup()` and `loop()` functions. These are functions just like the ones you're going to use in the `Blink` sketch, but you don't get to give them your own names; they have to be called `setup()` and `loop()`. When the Arduino first starts up, it looks for a function called `setup()` and executes the lines of code in it. It then looks for a function called `loop()` and repeatedly does whatever lines of code are in that function.

It's always easier to understand a new concept when you get to try it out yourself, so take a look at the `Blink` sketch you first worked with in Chapter 1. Open up the sketch by going to File ⇒ Examples ⇒ 1.Basics ⇒ Blink.

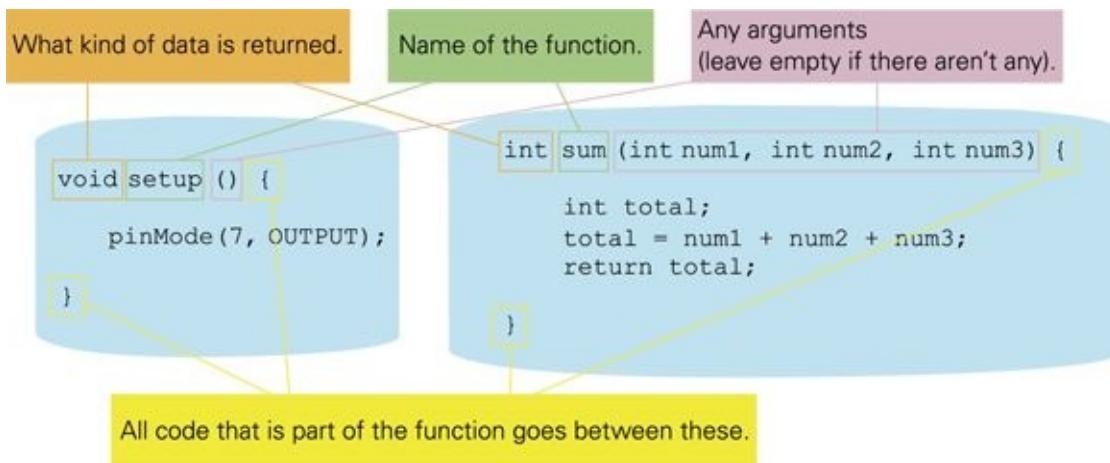
The first thing you need to do is save a copy of the `Blink` sketch. You're saving a copy rather than using the original example because you're going to make some changes to the sketch, so you don't want to overwrite the example. Save the sketch by selecting File ⇒ Save As. Name the file `BlinkingFunctions.ino`.

Take a look at your new `BlinkingFunctions` sketch (your copy of the `Blink` example sketch). In the sketch, most of the action happens in the `loop()` function:

```
// the loop routine runs over and over again forever:  
void loop() {  
    digitalWrite(led, HIGH);      // turn the LED on  
    delay(1000);                // wait for a second  
    digitalWrite(led, LOW);       // turn the LED off  
    delay(1000);                // wait for a second  
}
```

You can create your own function that does the same thing as the four lines of code in the preceding `loop()` function. But before you do that, hold fire! There are a couple of important things I want to highlight about writing a function first.

A function always has three pieces of information, and you need to type all three of these before the first curly bracket of each pair (see [Figure 4-2](#)).



**FIGURE 4-2** The anatomy of a function

The first piece of information is the type of data that will be output or returned from the function. In most of the code you create in this book, this is just `void`, meaning that there isn't anything returned. If there is data that is output or returned from the function, the data type is listed instead of `void`. For example, if your function computes an answer as an integer, the return data type is `int` instead of `void`.



You might have noticed that the term `void` keeps appearing before `setup()` and `loop()`. The term just means that the function doesn't return anything when it's finished. For example, you might write a function that calculates the sum of three numbers, intending the answer to the calculation returned to the position where you called the function so you can save the answer in a variable. This variable might be an `int`. For functions that simply turn on and off lights, no additional information is needed when the function finishes, so the return type is `void`.

The second bit of information is the name of the function. The rules for naming a function are similar to naming a variable as described in [Adventure 2](#):

- You can't have spaces in the name, but you can use numbers and letters.
- You can't start the name with a number.
- You make the first letter lowercase.

You might find the last piece of information a little hard to find; it's the round brackets (also known as parentheses). When there isn't anything between the first `(` and the second `)`, it means there aren't any input arguments. You can pass information to a function using input arguments; and you see how to do that later in this section.

Return to the sketch and go to the very last line of code—the one after the closing bracket of the `loop()` function. Add the following code, making sure it isn't inside any other function. (In other words, make sure that the code you're adding is not inside the parentheses or brackets of any other function.)

```
// turn on the LED for 1 second
// then off for 1 second
void blinkOnce() {
    digitalWrite(led, HIGH);      // turn the LED on
```

```
delay(1000);           // wait for a second
digitalwrite(led, LOW); // turn the LED off
delay(1000);           // wait for a second
}
```

You have just created a new function called `blinkOnce()`. It doesn't take any input arguments (because the `(` and `)` are empty after the function name), and it doesn't return anything (because it lists `void` before the function name). Inside the function, it blinks the LED on for 1 second and then off for 1 second.



It's a good habit to add a comment at the top of your function that explains what the function does. You can use `//` at the beginning of each line or `/*` and `*/` at the beginning and end of a paragraph. It might seem unnecessary when it's a simple function, but if you always do it, then you'll always remember to add comments for more complicated functions. When you share code that's well commented, others will be able to understand it, too.

Next, change the original `loop()` function so it only calls your new function:

```
// the loop routine runs over and over again forever:
void loop() {
  blinkOnce();
}
```

Upload the sketch to the Arduino board by connecting your Arduino Uno and clicking the Upload button. You should see the LED blink on and off just like the original `Blink` sketch did in [Adventure 1](#).

Now you're going to see what functions can really do. You're going to add a little more code that adds a variable to the function to control the speed of the blink.

Change your `blinkOnce()` function to the following (the changes are in bold):

```
// turn on the LED for time passed in argument
// then off for time passed in argument
void blinkOnce(int time) {
  digitalWrite(led, HIGH); // turn the LED on
  delay(time);           // wait
  digitalWrite(led, LOW); // turn the LED off
  delay(time);           // wait
}
```

By adding a variable between the `(` and `)`, your program is saying that the function needs some additional information in order to run. This is called an argument, and with it you can pass information directly from one function to another.

This means you need to include an argument when you call `blinkOnce()` in the `loop()` function. Change the `loop()` function again so it matches the following code:

```
// the loop routine runs over and over again forever:
void loop() {
  blinkOnce(1000);
}
```

When you add an argument of **1000**, the LED should blink on and off just as it did before —on for 1 second and then off for 1 second.

# CHALLENGE

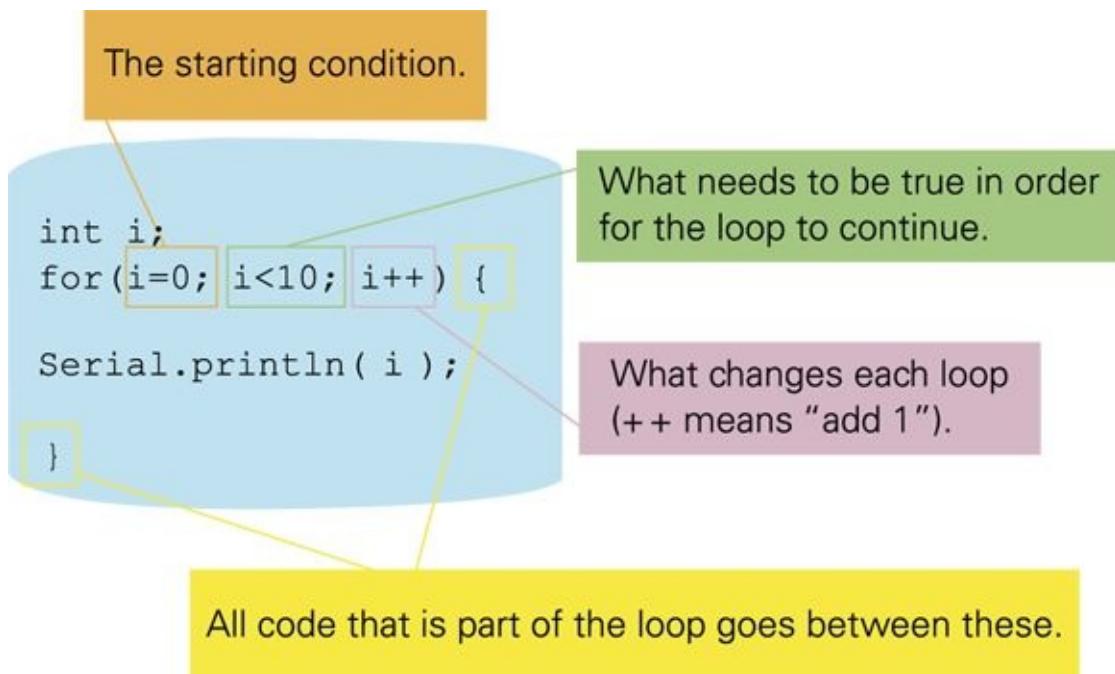


Make the LED blink on for 1 second and then off for 1 second, then on for 5 seconds and off for 5 seconds.  
You'll need to call the () function twice in the `loop()` function with different arguments.

# Using for Loops

Another useful way to organise your code is to repeat something a certain number of times. For example, instead of going to all the trouble of writing a function five times in a row, you can use another piece of code to do it for you. A **for** loop is one way of doing that.

A **for** loop needs three key pieces of information, as shown in [Figure 4-3](#).



[FIGURE 4-3](#) The anatomy of a **for** loop

The first piece of information is the starting condition. A **for** loop begins with a starting value for a variable. This is usually just a temporary variable that is only used in the **for** loop. It can be called anything that you would like, but programmers tend to call this variable **i**.

The second piece of information is what is needed in order for the loop to stop. This is phrased as a **true** or **false** question and is often checking if the variable has become too big. It might be something like **i<10**.

The last piece of information is what happens to the variable after each loop. The variable needs to get from its starting value to something that causes the loop to end; otherwise it would just go on forever and your program would never get past the **for** loop. This piece of information is usually **i++**, which simply means add **1** to **i** and save the new number in **i** again.

Make a new sketch (either by clicking the New button in the Arduino IDE or going to File => New) and type the following code:

```
void setup() {
    Serial.begin(9600);
}
```

```
void loop() {  
    int i;  
    for(i=0; i<10; i++){ // for loop that counts from 0 to 9  
        Serial.println(i); // print the current value of i  
        delay(1000); // wait for 1 second  
    }  
    delay(3000); // wait for 3 seconds  
}
```

Upload the code to your Arduino board and then open the Serial Monitor in the Arduino IDE by clicking on the Serial Monitor button or going to Tools ⇒ Serial Monitor. You should see the **for** loop counting from 0 to 9 over and over again.



Try changing **i++** to **i+=2**. What do you think is happening?

# Getting More Outputs with Shift Registers

In the earlier adventures in the book, you built circuits with some essential electrical components. Things like resistors are the most basic components but you can combine them with other basic components to form more complicated circuits. However, you don't necessarily have to spend a lot of time (and use up a lot of space) building a complicated circuit. You can sometimes buy a chip that has already been put together for you, containing more complicated circuits. These chips are called **integrated circuits**, or ICs for short.



**Integrated circuits** (ICs) are circuits contained within a single chip. The same circuit can be put into different shaped chips, called packages. When working with a breadboard, you want what is known a DIP or DIL package. That's the shape that has legs that fit into a breadboard.



Chips come in different packages. That just means different sizes and shapes. When working with breadboards, you will want to use components that are **dual in-line packages** (shortened to DIP or DIL). They have legs that fit into a breadboard. The other type of component package is a **surface-mount device** (SMD). SMD packages are very small and are designed to be easily placed on circuit boards in factories. They are much more difficult to use in circuits built at home with breadboards. Most of the components on your Arduino Uno (all those tiny black rectangles and even the LEDs) are SMD packages.



A **dual in-line package** (DIP or DIL) is one possible shape of an IC chip. It has two rows of legs that can fit into a breadboard.



A **surface-mount device (SMD)** is one possible shape of an IC chip or other component such as a resistor. It is made for soldering onto a flat surface without any legs being inserted into holes on a circuit board.

You can use multiple chips in the same circuit to do the same thing over and over. You can think of ICs as being the functions of electronic components.

In this adventure you are going to use an IC called a **shift register**. The shift register you'll use takes three inputs that control what happens on eight outputs. So with just three pins from the Arduino board, you will be controlling eight different LEDs. Even better, you can attach a shift register to another shift register in a chain. So you can keep adding eight more LEDs while still only using three pins on your Arduino board!



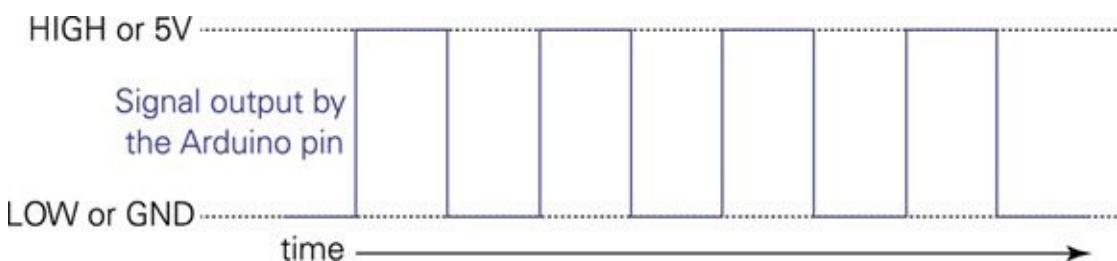
A **shift register** is a device that can control multiple outputs with relatively few inputs. It is commonly used to control a large number of LEDs.

# How a Shift Register Works

The three inputs that a shift register takes are the **CLOCK**, the **DATA** and **LATCH**.

## Clock

The **CLOCK** is the drum beat of the circuit. Messages are being sent from the Arduino to the IC. You can think of the Arduino as singing a song with the IC. In order for the IC to be able to follow along, the Arduino and IC need to sing at the same tempo. The **CLOCK** is a series of **HIGH** and **LOW** values (see [Figure 4-4](#)) that pulse to let the IC know when new information is being transmitted, like the drumbeat that lets the IC follow along with the Arduino.



**FIGURE 4-4** The **CLOCK** signal

## Data

The **DATA** is what you want each of the outputs (LEDs) to be set to, which will be either **HIGH** or **LOW**. A shift register can control eight LEDs. The Arduino Uno sends the shift register the value of the LEDs one by one like so:

1. The first LED value is sent to the shift register from the Arduino Uno. The shift register sets the first output pin to be that value.
2. The second LED value is sent to the shift register. The shift register set the second output pin to the value that was saved in the first output pin and then sets the first output pin to the most recent value sent by the Arduino Uno.
3. The Arduino Uno sends a third LED value to the shift register. The third output pin now is set to what the second output pin was previously set to; the second output pin is set to what the first output pin was set to; and the first output pin is set to the new value.
4. The Arduino Uno keeps sending new values to the shift register. Each time a new value comes in, the shift register shifts all the previously saved values for each output pin down to the next pin. The newest value sets the first output pin.

Each time you send the shift register a new value for an output, the previous value gets shifted to the next output. That's where the name shift register comes from!

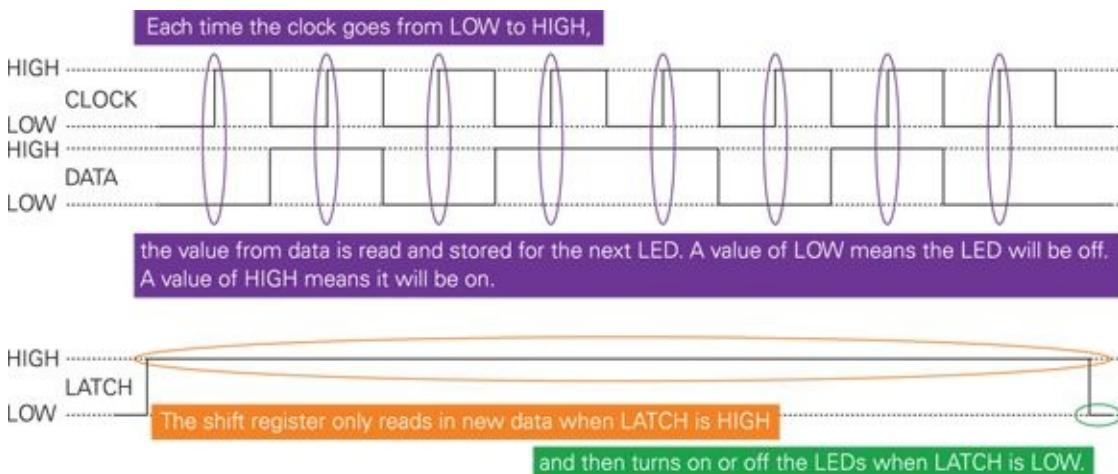
When you have finished sending output values, you need to tell the shift register that you have finished so it can turn on or off the output pins of the chip—which is what **LATCH**

does.

## Latch

The final input is the signal that tells the shift register to either listen for more information or go ahead and output the information it has. When the **LATCH** pin is **LOW**, the IC is listening; when it changes to **HIGH**, the IC starts doing and the output values are sent out. When LEDs are connected to the shift register, they will turn on or off according to the new values stored in the shift register when the **LATCH** changes to **HIGH**.

Figure 4-5 illustrates how the three inputs work together to control the shift register.



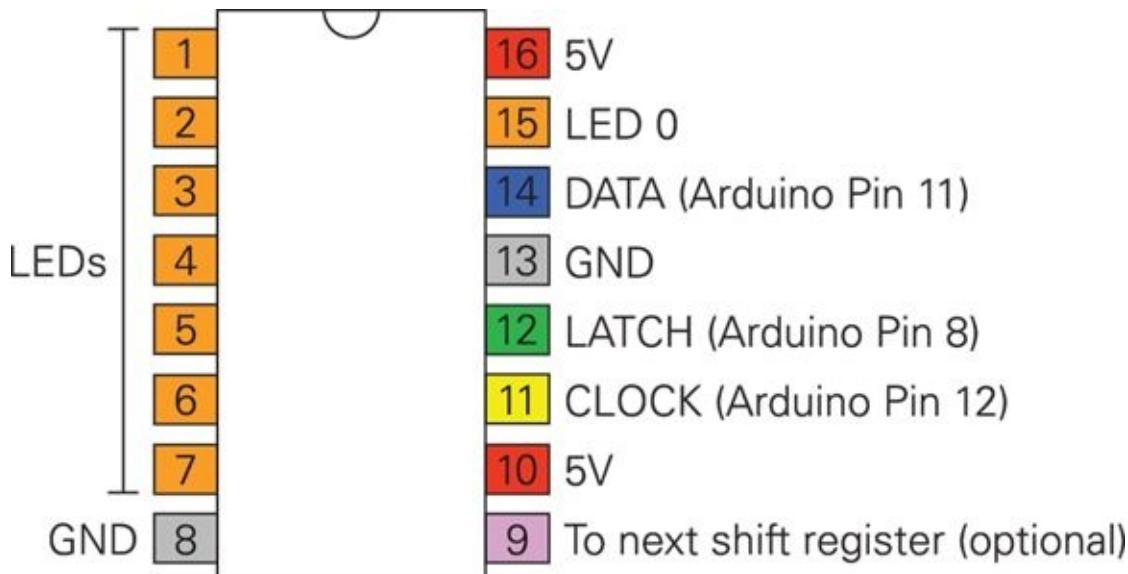
**FIGURE 4-5** How a shift register works

# Making the Connections for a Shift Register

The first thing to do is build your circuit. Start by putting your shift register chip on your breadboard. The chip fits over the gap in the middle of the board. You may need to bend the legs a little to get the chip to fit nicely into the holes. Notice that there's a little dot printed on a corner or a half circle cut out from one end of the chip: this is the top of the chip, and it's very important that the chip is in the same orientation as the diagram shown in [Figure 4-6](#).



The legs of the IC can be delicate, so take care when bending them. Also take care when removing the chip from the breadboard as it can be easy to accidentally bend the legs.



[FIGURE 4-6](#) Pin-out diagram for the shift register

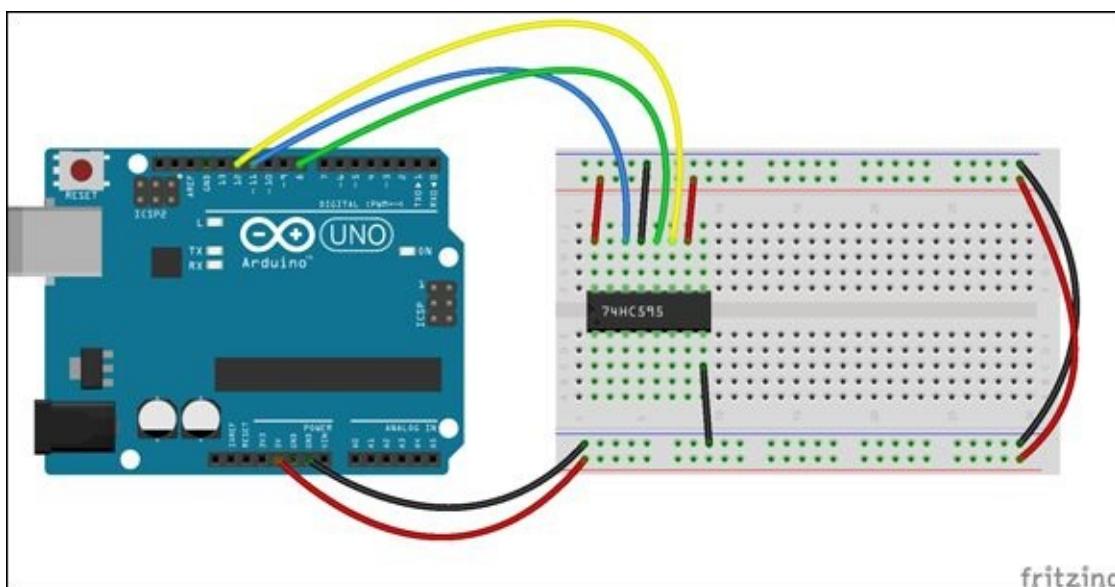
You are ready to start building your shift register circuit. [Figure 4-6](#) is a pin-out diagram that shows the shift register labelled with the pin numbers for the chip and shows what connects to each pin on the chip. Note that the colours correspond to the wire colours in [Figure 4-7](#). Now you need to make the first of the connections by following these steps (don't connect your Arduino Uno to your computer yet):

1. Use a jumper wire to connect one of the long rows along the bottom of your breadboard to a GND pin on the Arduino Uno. If your breadboard is a labelled with a blue or black line or a -, connect it to that row.
2. Use a jumper wire to connect the other long row along the bottom of your breadboard to the 5V pin on the Arduino Uno.
3. Use two jumper wires to connect each of the long rows along the bottom of the breadboard to the long rows along the top. If your breadboard is labelled, connect the red or + to the other row with red or a + then connect the remaining two long rows to each other.
4. Use two jumper wires to connect the short rows connected to Pin 8 and Pin 13 on the

shift register to the long row on the breadboard connected to ground.

5. Use two jumper wires to connect the short rows connected to Pin 10 and 16 on the chip to the long row connected to 5V.
6. Use a jumper wire to connect the short row connected to Pin 14 on the shift register to Pin 11 on the Arduino board.
7. Use a jumper wire to connect the short row connected to Pin 11 on the shift register to Pin 12 on the Arduino board.
8. Use a jumper wire to connect the short row connected to Pin 12 on the shift register to Pin 8 on the Arduino board.

When you've finished, your circuit should look like [Figure 4-7](#). Notice the shift register chip in the middle, facing in the correct direction.



**FIGURE 4-7** First connections for the shift register

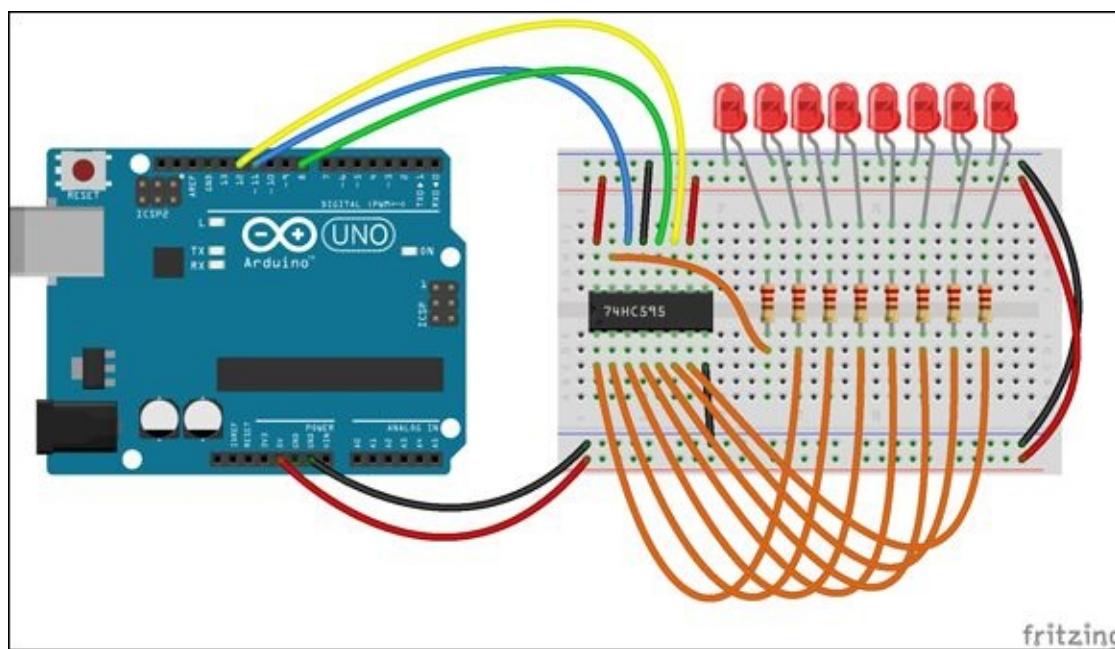
## Adding LEDs

Now it's time to add the LEDs. If you have a second breadboard, you might find it easier to have the shift register on one board and the LEDs on another, but you can also fit everything on a single breadboard.

Each of the LEDs needs a current-limiting resistor, just like when a LED is hooked up directly to a pin on the Arduino board. Each output of the shift register is connected to a current-limiting resistor and then to a LED, which is then connected to ground.

Using [Figure 4-8](#) as a guide, follow these steps to add your LEDs and resistors:

1. Place the short legs of 8 LEDs in the long row along the top of the breadboard that is connected to ground.
2. Place each of the long legs of the LEDs into their own short rows on the breadboard—wherever they easily fit is fine as long (as nothing else is already connected to the row).
3. Place one leg of a resistor into the same short rows as the long legs of the LEDs.
4. Bend the resistors over the gap in the middle of the breadboard, and insert the free legs of the resistors into the short row directly on the other side of the gap.
5. Use eight jumper wires to connect each resistor to an output pin on the shift register—pins 15 and 1 through 7. Use [Figures 4-6](#) and [4-8](#) as guides.



**FIGURE 4-8** The full circuit for the shift register

Double-check that your connections are correct—go through each step again. When are you sure everything is in the right place, you are ready to power the circuit by connecting the Arduino Uno to your computer.



Your chip should never get hot! If it ever gets hot, something is plugged in wrong, so you must remove

the power immediately. The chip might be damaged and may need to be replaced. If it ever makes a popping sound and even smokes a little, it definitely needs to be replaced. Be sure to check over your circuit again and find the mistake before putting in a new chip, or you will just damage the next chip in the same way!

## Writing the Code

Create a new sketch by clicking on the New button in the Arduino IDE or going to File => New and enter the following code:

```
void setup() {  
}  
  
void loop() {  
}
```

At the very top of your sketch, before the `setup()` begins, add the following variables:

```
//Pin connected to latch pin (ST_CP) of 74HC595  
int latchPin = 8;  
//Pin connected to clock pin (SH_CP) of 74HC595  
int clockPin = 12;  
//Pin connected to Data in (DS) of 74HC595  
int dataPin = 11;
```

They are the three pins for the **LATCH**, **CLOCK** and **DATA** connections to the shift register.

Next, type the following code between the `{` and `}` of `setup()`:

```
//set pins to output because they are addressed in the main loop  
pinMode(latchPin, OUTPUT);  
pinMode(dataPin, OUTPUT);  
pinMode(clockPin, OUTPUT);
```

Each of those lines of code set up the pins to be outputs that send out data to the shift register (as opposed to reading in data).

Lastly, type the following code between the `{` and `}` of `loop()`:

```
// loop through 0 to 256  
int i;  
for(i=0; i<256; i++) {  
    // turn off the output so the pins don't light up  
    // while you're shifting bits:  
    digitalWrite(latchPin, LOW);  
    shiftOut(dataPin, clockPin, LSBFIRST, i);  
  
    // turn on the output so the LEDs can light up:  
    digitalWrite(latchPin, HIGH);  
    delay(300);  
}
```

Your complete sketch should now look like this:

```
//Pin connected to latch pin (ST_CP) of 74HC595
```

```

int latchPin = 8;
//Pin connected to clock pin (SH_CP) of 74HC595
int clockPin = 12;
//Pin connected to Data in (DS) of 74HC595
int dataPin = 11;

void setup() {
    //set pins to output because they are addressed in the main loop
    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
}

void loop() {
    // loop through 0 to 256
    int i;
    for(i=0; i<256; i++) {
        // turn off the output so the pins don't light up
        // while you're shifting bits:
        digitalWrite(latchPin, LOW);
        shiftOut(dataPin, clockPin, LSBFIRST, i);

        // turn on the output so the LEDs can light up:
        digitalWrite(latchPin, HIGH);
        delay(300);
    }
}

```



You can download the sketches in this chapter from the companion site  
[www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino) ).

Upload the sketch to your Arduino Uno. Your LEDs should start turning on and off.

# DIGGING INTO THE CODE



So what is going on in the `loop()` function? The `loop()` function uses a `for` loop as described earlier in this adventure. Inside the `for` loop, you call a function named `shiftOut()`. This is a function that the Arduino knows, and it takes four arguments. In the first argument it sends out a number through the given pin (`dataPin`), using the pin given in the second argument to send the `CLOCK` signal (`clockPin`). The third argument uses a keyword to indicate whether the number being sent out starts with the first digit or the last digit (`LSBFIRST`). The last argument is the number being sent out. Here the number being sent out is stored in `i`, which is controlled by the `for` loop. It starts at `i=1` and stops after `i=255`.

```
shiftOut(dataPin, clockPin, LSBFIRST, i);
```

Before `shiftOut()` is called, the `latchPin` is set to `LOW`. This tells the shift register to stop doing and start listening. The new values for the LEDs are then sent in the `shiftOut()` function, and then the `latchPin` is set to `HIGH`. That tells the shift register to stop listening and start doing. It then turns on and off the LEDs according to the new values it just received.

You may have spotted something a little weird in the code. Why does the `for` loop start at `0` and count up to `256`? Doesn't that seem a little strange?

Computers like to start counting at `0`. As humans, we usually skip over `0` and start counting at `1`, but `0` is typically the starting point for computers. That's why the `for` loop starts at `0`.

And why are you using `255` as the maximum value? The shift register is controlling eight LEDs. Each LED can either be on or off, so that's two possible states for every LED. If a `0` represents an LED off and a `1` represents an LED on, you can describe the on and off states of all the LEDs with a single number. `11111111` would be all the LEDs on. `10000001` would be all the LEDs off except the first and last ones.

These numbers are special because they don't use all the possible digits between `0` and `9`, but instead only `0` and `1`. Numbers that use all the digits from `0` to `9` are called decimal numbers (what you think of as normal numbers), and numbers that count using only `0` and `1` are called **binary**. The number being sent out the `dataPin` is represented in binary.

The sketch that you just wrote is a binary counter; it shows you in lights how to count from `0` to `255` in binary (`0` to `11111111`).



A **binary** number uses only the digits `0` and `1`, as opposed to decimal, which uses the digits `0` through `9`. Binary is also referred to as base-`2`. Decimal is referred to as base-`10`.

[Figure 4-9](#) shows you how to convert binary numbers into decimal numbers. At this point, you don't need to worry too much about this if learning about different ways of representing numbers doesn't seem like much fun, but if you like secret codes and messages, it may be a topic that you will find very interesting.

To calculate the decimal number that represents a pattern of LEDs turned on and off:

Multiply either 0 or 1 by 2 to the power of the position of the light, depending if the light is on or off.

And then add up all those multiplications.

$$\begin{array}{r} 128 \times 1 \\ 64 \times 0 \\ 32 \times 1 \\ 16 \times 0 \\ 8 \times 1 \\ 4 \times 0 \\ 2 \times 1 \\ + 1 \times 0 \\ \hline 170 \end{array}$$

1 for the lights that are on,  
0 for the lights that are off.

Position Number:

1	0	1	0	1	0	1	0
7	6	5	4	3	2	1	0

$$2^7 = 128$$

$$2^6 = 64$$

$$2^5 = 32$$

$$2^4 = 16$$

$$2^3 = 8$$

$$2^2 = 4$$

$$2^1 = 2$$

$$2^0 = 1$$

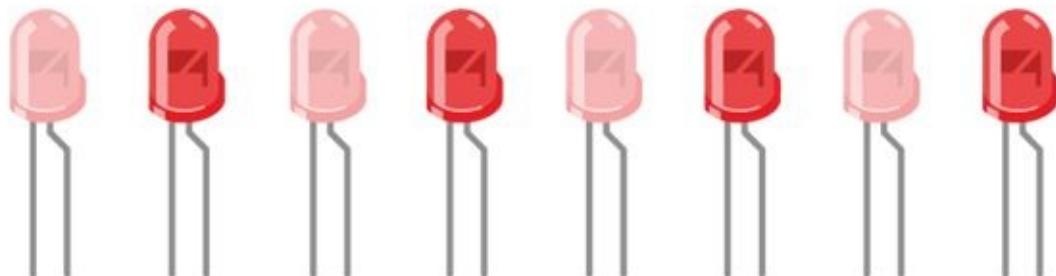
**FIGURE 4-9** How to convert from a binary number to a decimal number

# CHALLENGE



Calculate the decimal number from the binary pattern shown in [Figure 4-10](#).

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---



**FIGURE 4-10** How would this binary pattern be represented by a decimal number?

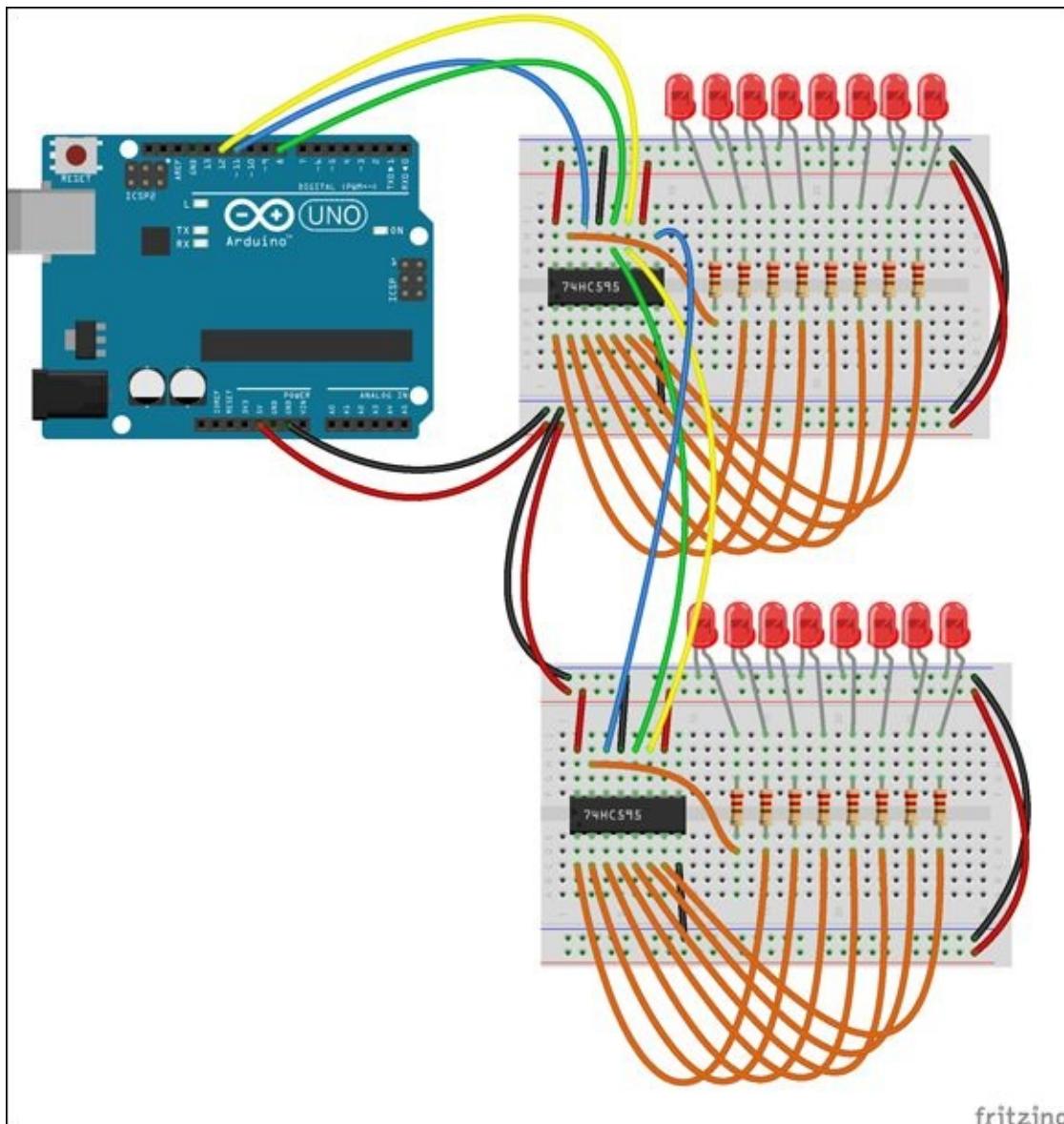


A single digit in a binary number is called a bit and a group of 8 bits is called a byte. All computing is based on bits and bytes. If you're interested in learning more, you can start by looking up more information on bits and bytes at <http://en.wikipedia.org/wiki/Bit> and <http://en.wikipedia.org/wiki/Byte>.

## Adding More Shift Registers

To add another shift register, you need to put a second IC on the breadboard (if you’re using a big breadboard) or on a second breadboard (if you’re using two small breadboards):

1. Follow the steps you followed earlier to connect the shift register to 5V and GND. If you are using a second breadboard, be sure to connect the long rows of that breadboard to the long rows of the first breadboard.
2. Instead of connecting Pins 12, 14 and 11 on the shift register to Pins 8, 11 and 12 on the Arduino Uno, connect Pins 11 and 12 of the second shift register to Pins 11 and 12 on the first shift register. This connects **LATCH** and **CLOCK** from the first shift register to the second one, as shown in [Figure 4-11](#).
3. The **DATA** for the second shift register doesn’t come from the Arduino board, but from Pin 9 of the first shift register. Use a jumper wire to connect Pin 14 of the second shift register to Pin 9 of the first shift register.
4. Follow the same steps in the “Adding LEDs” section for adding the LEDs the first shift register to connect 8 more LEDs to the second shift register.



**FIGURE 4-11** Adding a second shift register

When your circuit is built, you need to change your sketch so that it controls two shift registers instead of only one. Make the changes shown in bold to the `loop()` function of your sketch:

```
void loop() {
    // loop through 0 to 256
    int i;
    for(i=0; i<256; i++) {
        // turn off the output so the pins don't light up
        // while you're shifting bits:
        digitalWrite(latchPin, LOW);

        // send to second shift register
shiftOut(dataPin, clockPin, LSBFIRST, i);
        // send to first shift register
shiftOut(dataPin, clockPin, LSBFIRST, i);
```

```
// turn on the output so the LEDs can light up:  
digitalWrite(latchPin, HIGH);  
delay(300);  
}  
}
```

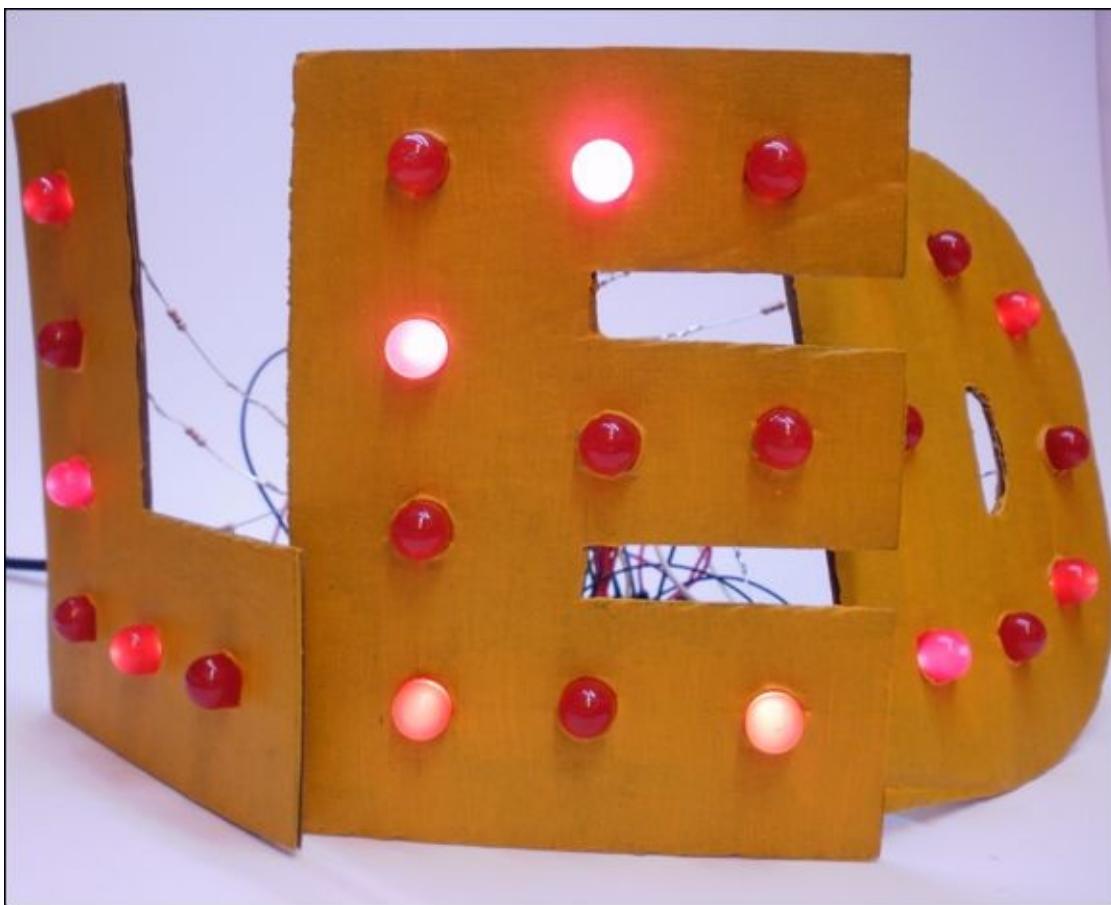
Upload the sketch to your Arduino Uno and watch all 16 of your LEDs turn on and off.

# Building Your Name in Lights

Different electrical components can perform the same function but look very different from each other. LEDs are an example of this; for example, you can use LEDs in different colours without having to change the circuit, although you still need a current-limiting resistor and a connection to a positive voltage and ground.

LEDs are measured by their width and you can also buy them in different sizes. You are probably using 5mm LEDs but you can get them in all sizes so you might like to try them in 3mm or 10mm. The 10mm LEDs work nicely in this project, but you can use whatever size and colour that you think looks good.

In this project you can light up as many as 24 LEDs. You can decide how you want to arrange those LEDs and use them to embellish a carnival-style letter sign. You're going to create your own design for a fantastic sign and put your name in lights (see [Figure 4-12](#)). You can choose what you'd like to spell. It can be your name (or just initials if your name is quite long) or any other word—like LED!



**FIGURE 4-12** Your name (or any other word) in lights!

You will choose what letters you would like to make and cut them out of cardboard. Then you can decide where you want to place the 24 LEDs and add them to your letters.



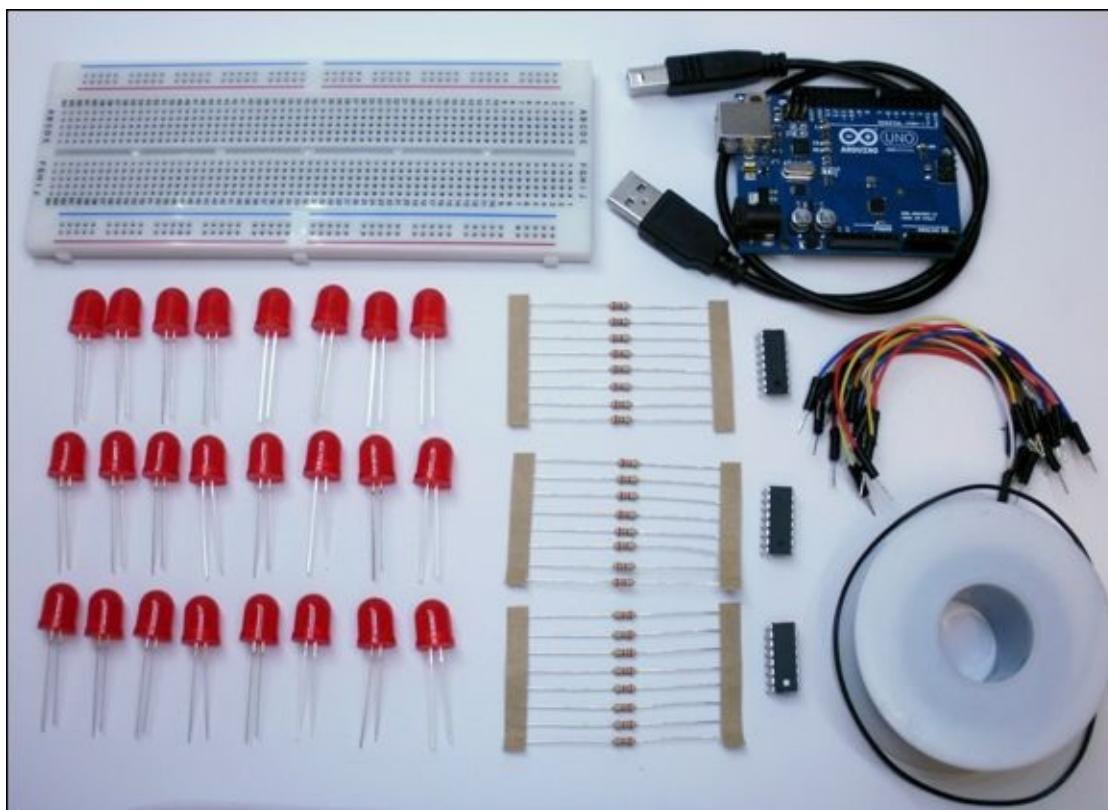
You can watch a video of how to build a carnival-style letter sign on the companion website at

[www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

## What You Need

Following is a list of what you need to build your sign. Remember that Appendix A lists places that you can buy the electronic components shown in [Figure 4-13](#).

- A computer
- An Arduino Uno
- A USB cable
- A breadboard (you may need several if you build a lot of letters)
- 57 jumper wires
- 24 LEDs
- 24  $220\Omega$  resistors
- 3 74HC595 shift register ICs (1 for every 8 LEDs)
- Some cardboard (cutting up old cardboard boxes works well)
- Some wire
- Some solder
- Paint or coloured paper for decoration
- A soldering iron
- Masking tape
- Scissors or a utility knife
- A pencil, screwdriver or hole punch

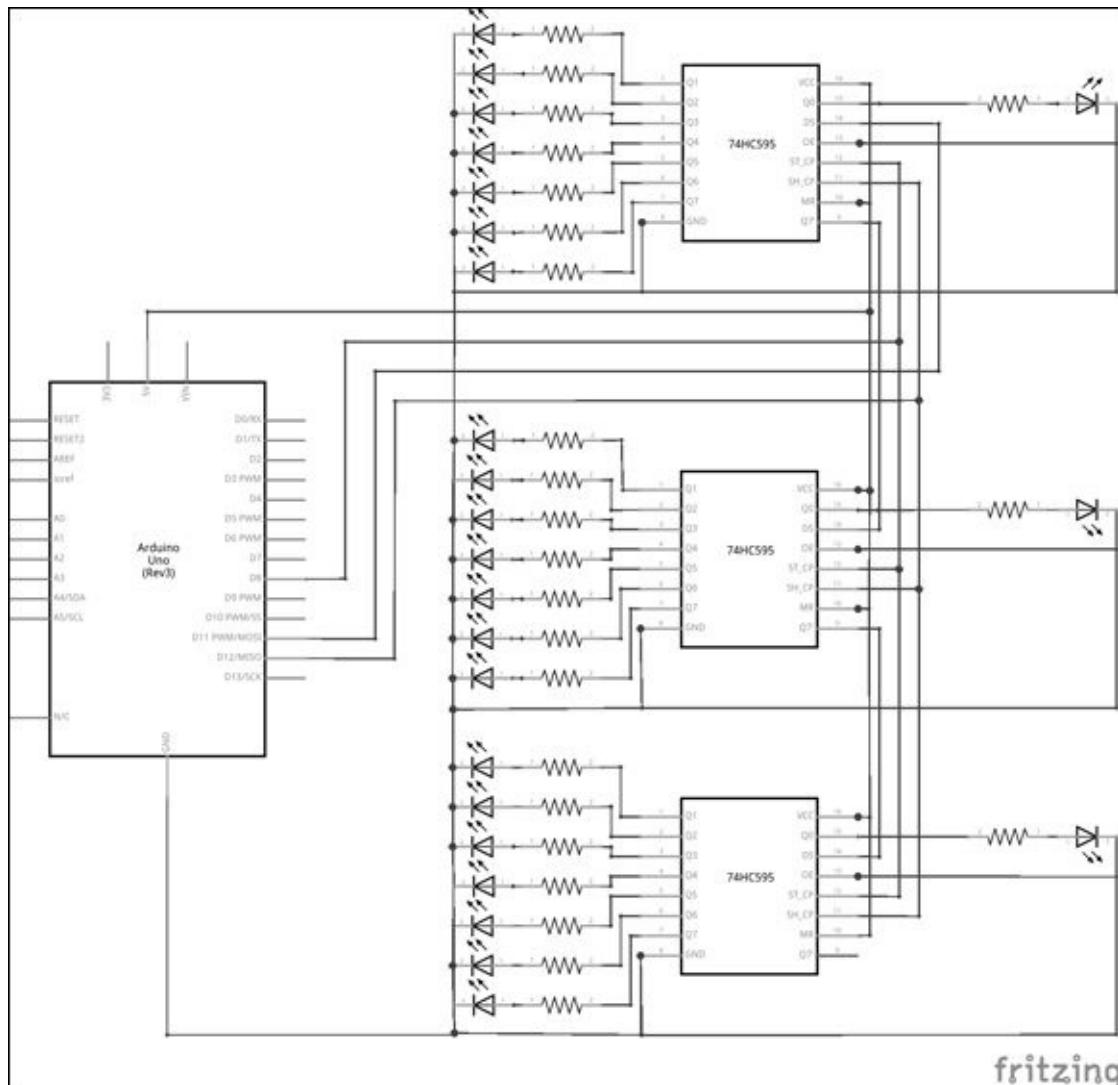


**FIGURE 4-13** The electronic components you need to build your name in lights

# Understanding the Circuit

The circuit for this project is very similar to the one you built earlier in this chapter using two shift registers with 16 LEDs. You can add up to three shift registers and 24 LEDs. You need one shift register for every 8 LEDs.

There are a lot of connections to make, so use [Figure 4-14](#) as a guide to what should be connected. Remember, only the lines that intersect with a circle are the wires that are connected to each other.



**FIGURE 4-14** Circuit schematic for three shift registers

## Prototyping on a Breadboard

Always check that your circuit is working before you start soldering! Prototype the circuit with three shift registers. You may need to use multiple breadboards. Follow the steps in the “Adding More Shift Registers” section to set up three shift registers. Pin 14 of one shift register will be connected to Pin 11 on the Arduino Uno. For the other two shift registers, Pin 14 on both of them connect Pin 9 of the next shift register. Use [Figure 4-11](#) as a guide and add a third shift register to the second shift register in the same way you connected the second shift register to the first shift register. Connect eight LEDs to each of the shift registers using the steps in “Adding the LEDs” section.

## Writing the Code

Create a new sketch by clicking on the New button in the Arduino IDE or going to File ⇒ New. Type the following code to begin writing your sketch:

```
void setup() {  
}  
  
void loop() {  
}
```

At the top of the sketch, before the `setup()`, add the following variables:

```
//Pin connected to latch pin (ST_CP) of 74HC595  
int latchPin = 8;  
//Pin connected to clock pin (SH_CP) of 74HC595  
int clockPin = 12;  
//Pin connected to Data in (DS) of 74HC595  
int dataPin = 11;  
  
// number of shift registers used  
int numRegisters = 3;  
  
// first pattern to be displayed  
int pattern1 = 85;  
// second pattern to be displayed  
int pattern2 = 170;
```

Inside `setup()`, set the pin modes and call a function, `setLEDs()`, that you will write later. Add the following code:

```
//set pins to output because they are addressed in the main loop  
pinMode(latchPin, OUTPUT);  
pinMode(dataPin, OUTPUT);  
pinMode(clockPin, OUTPUT);  
  
// start with all LEDs off  
setLEDs(0);
```

Inside the `loop()` type the following code to send a blinking pattern to the LEDs:

```
// turn on LEDs in the pattern 01010101  
setLEDs(pattern1);  
// wait 1 sec  
delay(1000);  
// turn on LEDs in the pattern 10101010  
setLEDs(pattern2);
```

```
// wait 1 sec  
delay(1000);
```

The only function missing is the `setLEDs()` function. This is a new function that you writing—it isn't included in the Arduino IDE. Type the following code after the `loop()` (after the `}`) and you can read more about what it is doing in the Digging into the Code sidebar:

```
// sends pattern to shift register for  
// which LEDs to turn on and off  
void setLEDs(int lightPattern) {  
    // turn off the output so the pins don't light up  
    // while you're shifting bits:  
    digitalWrite(latchPin, LOW);  
  
    int i;  
    for(i=0; i<numRegisters; i++) {  
        // sends out the pattern once for each shift register  
        shiftOut(dataPin, clockPin, LSBFIRST, lightPattern);  
    }  
  
    // turn on the output so the LEDs can light up:  
    digitalWrite(latchPin, HIGH);  
    delay(300);  
}
```

Following is the full sketch, but you can also download it from the companion site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

```
//Pin connected to latch pin (ST_CP) of 74HC595  
int latchPin = 8;  
//Pin connected to clock pin (SH_CP) of 74HC595  
int clockPin = 12;  
//Pin connected to Data in (DS) of 74HC595  
int dataPin = 11;  
  
// number of shift registers used  
int numRegisters = 3;  
  
// first pattern to be displayed  
int pattern1 = 85;  
// second pattern to be displayed  
int pattern2 = 170;  
  
void setup() {  
    //set pins to output because they are addressed in the main loop  
    pinMode(latchPin, OUTPUT);
```

```

pinMode(dataPin, OUTPUT);
pinMode(clockPin, OUTPUT);

// start with all LEDs off
setLEDs(0);
}

void loop() {
    // turn on LEDs in the pattern 01010101
    setLEDs(pattern1);
    // wait 1 sec
    delay(1000);
    // turn on LEDs in the pattern 10101010
    setLEDs(pattern2);
    // wait 1 sec
    delay(1000);
}

// sends pattern to shift register for
// which LEDs to turn on and off
void setLEDs(int lightPattern) {
    // turn off the output so the pins don't light up
    // while you're shifting bits:
    digitalWrite(latchPin, LOW);

    int i;
    for(i=0; i<numRegisters; i++) {
        // sends out the pattern once for each shift register
        shiftOut(dataPin, clockPin, LSBFIRST, lightPattern);
    }

    // turn on the output so the LEDs can light up:
    digitalWrite(latchPin, HIGH);
    delay(300);
}

```

Connect your Arduino Uno to your computer and upload the code. Your lights should start flashing.

# DIGGING INTO THE CODE



The variables at the top of the code and most of the `setup()` should look familiar to you; they are the same as what you used earlier in this adventure. There is one new variable: `numRegisters`. It is currently set to `3`, but if you would like to use fewer shift registers, you can change it to the number you are using.

There's also a new function: `setLEDs()`. This function is defined underneath `loop()`. It takes one argument: the pattern of LEDs to light up (`pattern1` or `pattern2`).

The pattern is described by a decimal number. For `pattern1`, it is the number 85 in decimal, which is 01010101 in binary, so it turns on every other LED of each set of eight LEDs. The other LED pattern is `pattern2`, which is 170 in decimal or 10101010 in binary. It is the opposite of `pattern1`—it turns on the LEDs that were off in `pattern1` and turns off the LEDs that were on. So when the LEDs alternate between `pattern1` and `pattern2`, it creates a flashing pattern.



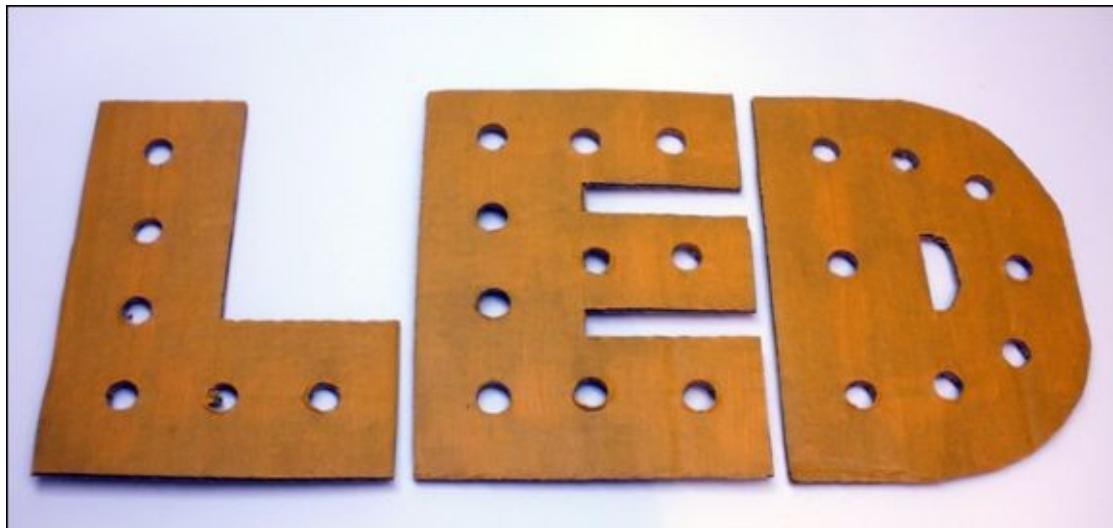
Want your sign to light up in a different pattern? It's not too difficult to create your own. Write out the pattern as a sequence of eight 0s and 1s and then calculate what that number would be in decimal. You can also use an online calculator to help convert between binary and decimal; try

[www.mathsisfun.com/binary-decimal-hexadecimal-converter.html](http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html) or  
[www.binaryhexconverter.com/binary-to-decimal-converter](http://www.binaryhexconverter.com/binary-to-decimal-converter).

## Making the Lights

Your first—and biggest—decision is to choose the letters you will create. Remember, you only have 24 LEDs to decorate your signs with. You need to decide what letters you would like to make and where the LEDs will be placed.

1. Trace your letters onto cardboard. Cardboard from old boxes works well. Use a pair of scissors or a utility knife to cut out the letters.
2. If you want to decorate your letters with paint or paper, go ahead and do that now.
3. When any paint or glue used to decorate your letters are dry, use a tool to poke a hole in the cardboard letters where you want to place each LED. A pencil or screwdriver can work well. Make sure the hole is just big enough to snugly hold the LED in place. At this point you're just determining the placement of the LEDs. You insert the lights into the holes after you have the wires soldered. You should now have letters similar to the ones in [Figure 4-15](#).



**FIGURE 4-15** Cardboard letters with holes for LEDs

# Soldering the Wires

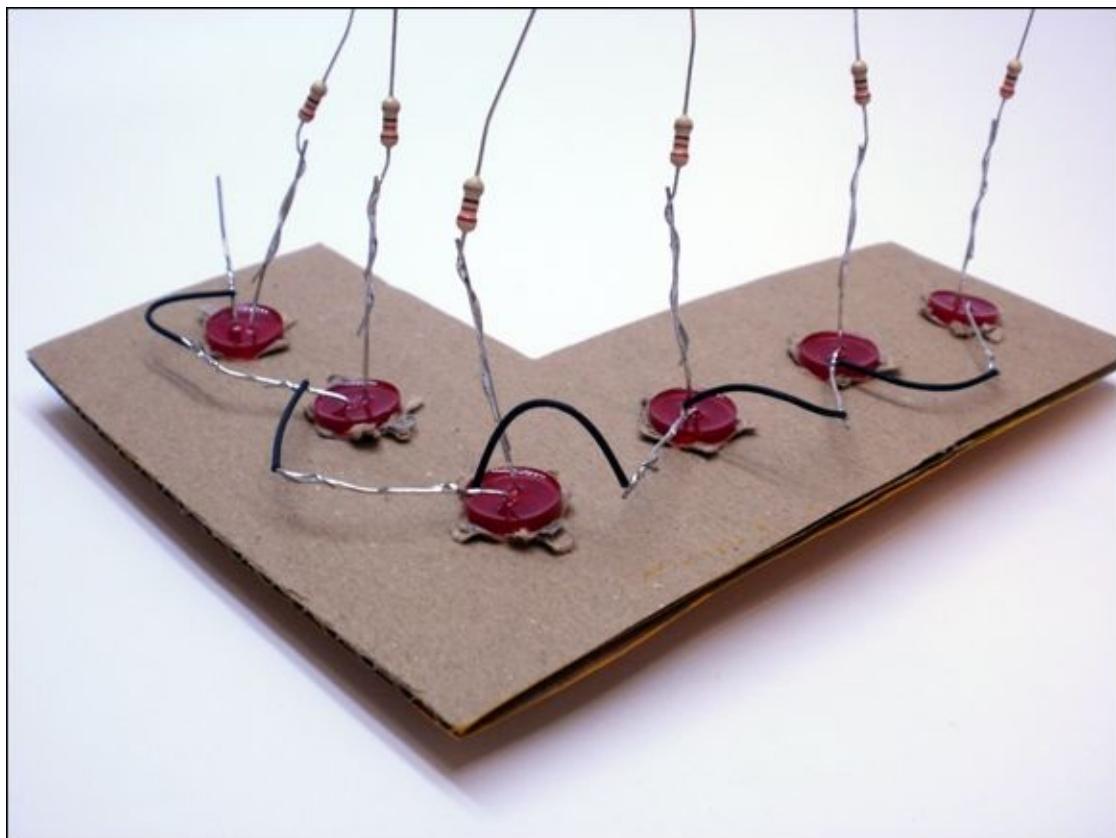
Use the following steps to solder your circuit:

1. Solder a resistor to the long leg of each LED. Twist the legs of the resistor and LED together so that they don't easily come apart and then solder them.



Always have an adult nearby when you are soldering. [Adventure 2](#) has some more tips to help you stay safe when you solder.

2. Place the LEDs in their holes in the cardboard letters. Bend the short legs of each of the LEDs towards the next LED (use [Figure 4-16](#) as a guide). If the short leg of an LED doesn't reach the LED next to it, cut a piece of wire that reaches from that LED to the next. Solder either the wire to each LED or the short leg of the first LED directly to the next. Repeat for all the LEDs. You should have one short leg of an LED left on each letter. The rest of the short legs of the LEDs should be connected to each other.
3. You use a breadboard to connect your shift registers to the LEDs and the Arduino Uno. Decide where the breadboard and Arduino Uno will be placed. They could be taped to the back of a letter or could rest on the table next to the letters.
4. Cut 24 pieces of wire that reach from each of the resistors soldered to the LEDs to the breadboard. You might want to hide the wires by taping them along the back of the letters, so be sure to cut them long enough for that if that's what you would like to do.
5. Strip about  $\frac{1}{2}$ " from each end of all the wires you just cut. Solder each wire to its LED.
6. Cut a wire that reaches from the remaining short LED leg on each letter to the breadboard. You need one for each letter.
7. Strip about  $\frac{1}{2}$ " from each end of the wires you just cut. Solder each wire to its short LED leg on each letter.



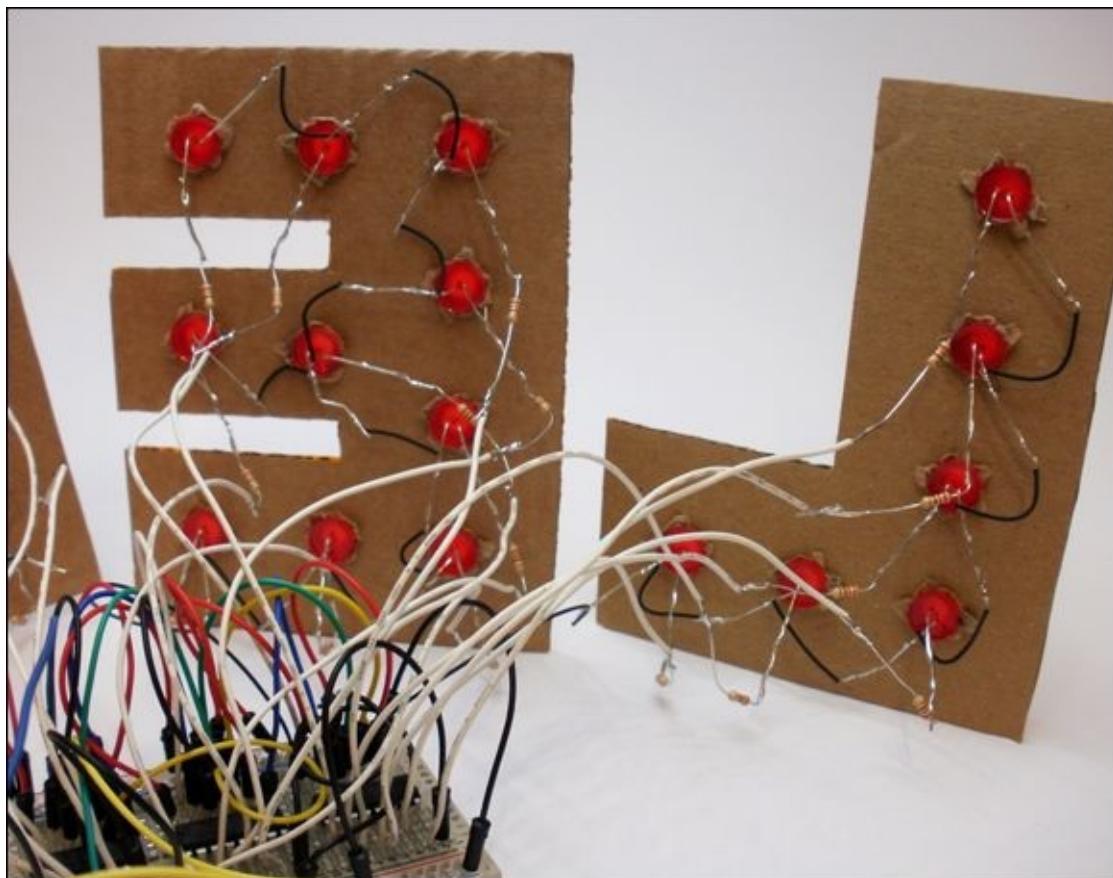
**FIGURE 4-16** Soldered LEDs and resistors

## Inserting the Electronics

When you've finished soldering all your LEDs, just pop them through the holes in your letters. You might need to add a little glue if they don't stay put. Then you are ready to build your shift register circuits onto one breadboard as shown in [Figure 4-17](#):

1. Follow the same steps you went through when building the prototype circuit. You connect the 5V, ground, **CLOCK**, **LATCH** and **DATA** lines as you have done previously.
2. The resistors are soldered to the LEDs, so they don't need to be placed on the breadboard. Instead connect the wires that you soldered to the resistors to the pins on the shift registers. See [Figure 4-16](#) for guidance.
3. Connect the wires from the short legs of each LED to ground on the breadboard.
4. Connect your Arduino Uno to your computer or to a power supply.

Power up your Arduino board and hey presto! Your name is in lights!



**FIGURE 4-17** Back of lights

# Further Adventures with Shift Registers

You now have a great display to try out different light animations. Try writing your own functions with light patterns of your own design.

You might want to check out some of these tutorials online:

- <http://arduino.cc/en/tutorial/ShiftOut>
- <https://learn.adafruit.com/adafruit-arduino-lesson-4-eight-leds/the-74hc595-shift-register>

In the project you just created, you are sending a decimal number to `shiftOut()`, which then turns the number into a binary number to tell each output pin whether it should be **HIGH** or **LOW**. You can read more about binary numbers at [http://en.wikipedia.org/wiki/Binary-coded\\_decimal](http://en.wikipedia.org/wiki/Binary-coded_decimal).

**Arduino Command Quick Reference Table**

Command	Description
<code>void</code>	Tells the computer that no data will be returned by a function when it finishes. See also <a href="http://arduino.cc/en/Reference/Void">http://arduino.cc/en/Reference/Void</a> .
<code>shiftOut()</code>	Sends a series of HIGH and LOW values in time with a CLOCK signal. See also <a href="http://arduino.cc/en/Reference/ShiftOut">http://arduino.cc/en/Reference/ShiftOut</a> .



**Achievement Unlocked:** Skillful engineer of shining signs!

## In the Next Adventure

In the next adventure, you will learn how to add a speaker and play music to transform your Arduino into an electronic synthesiser!



## Adventure 5

### Playing Sounds

THERE ARE LOTS of ways to make code control things in the real world. You have already controlled movement with motors and controlled light with LEDs. In this adventure you're going to create sound!

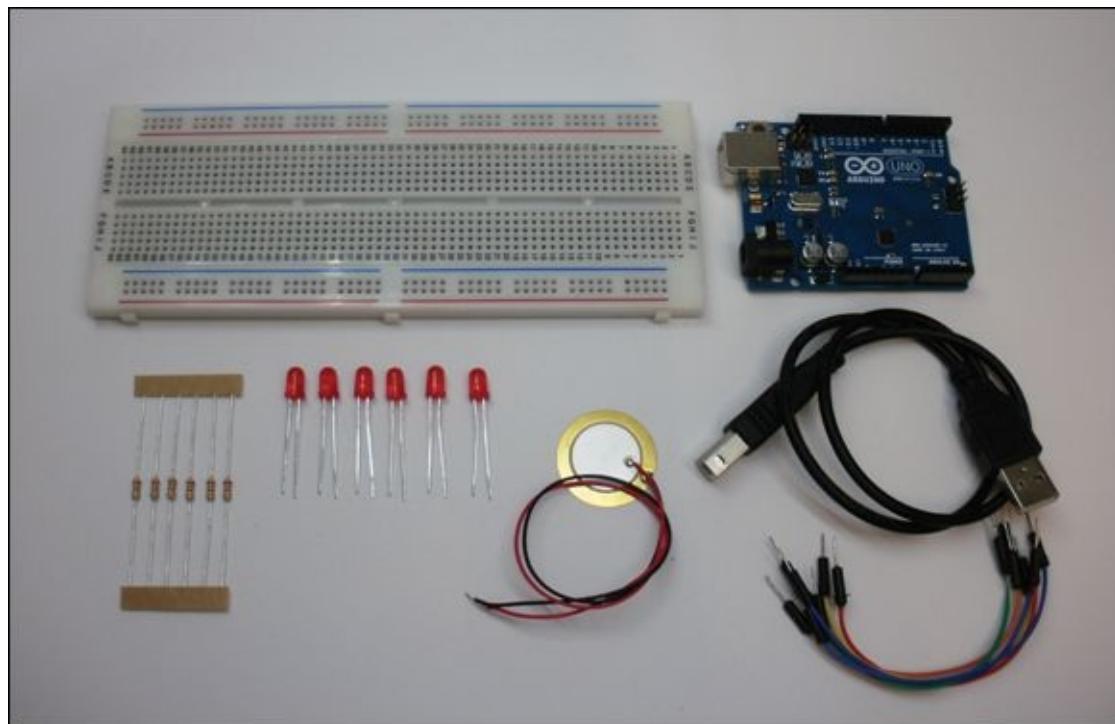
In [Adventures 3](#) and [4](#), you discover some new ways to make your code more efficient when you need to repeat the instructions more than once. The **for** loop is a great tool for repeating something a set number of times. When you combine a **for** loop with special lists in code—called arrays—you end up with a powerful coding tool.

After learning how to harness the power of arrays and figuring out how to get your Arduino to sing to you, you're going to put your new skills into practice by building an augmented wind chime that plays both acoustic and electronic sounds.

# What You Need

You need a few things for the first part of this chapter (the electronic components are shown in [Figure 5-1](#)):

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 7 jumper wires
- 6 LEDs
- 6  $220\Omega$  resistors
- A piezo



[Figure 5-1](#) The electronic components you need for the first part of this adventure

# Making a List

Variables are useful ways of keeping track of information like the numbers of the pins on your Arduino Uno. Creating a variable to store the number of the pin to which a particular LED is connected makes your code easier to read later on. When you create a variable like this:

```
int ledPin = 12;
```

you can write this later:

```
digitalWrite(ledPin, HIGH);
```

instead of having to write this:

```
digitalWrite(12, HIGH);
```

Using the variable makes your code easier to read. You might not remember what is connected to Pin 12, but the variable `ledPin` makes it easier to figure out what the code is doing.

But what if you want to keep track of more than one LED? Well, you could create a variable for each pin, like this:

```
int ledPin1 = 3;  
int ledPin2 = 4;  
int ledPin3 = 5;  
int ledPin4 = 6;  
int ledPin5 = 7;  
int ledPin6 = 8;
```

But that doesn't seem very efficient! Why should you have to type the same thing over and over again when, as we know, computers are good at doing the same thing many times. There must be a better way, right?

There is! Computers, including the Arduino, can collect information in lists called **arrays**. Instead of having to create a variable for each piece of information, you create a variable that is an array. Whenever you want to refer to one of the items in the array, you simply give the number of that item. [Figure 5-2](#) shows examples of two arrays.



An **array** is a list of the same type of thing in code. For example, an array can hold a list of **ints**.

```
int[] listOfInts = { 7, 9, 5, 1, 9, 2 };    int[] listOfFloats = { 1.2, 3.1, 9.8, 2.5,  
listofInts[1] is the int 9                      3.2, 2.7 };  
listofInts[5] is the int 2                      listOfFloats[1] is the float 3.1  
                                              listOfFloats[5] is the float 2.7
```

listOfInts						
Item Number	0	1	2	3	4	5
Item Value	7	9	5	1	9	2

listOfFloats						
Item Number	0	1	2	3	4	5
Item Value	1.2	3.1	9.8	2.5	3.2	2.7

[Figure 5-2](#) Two example arrays

The one little trick to remember with arrays is that the first item isn't number 1; it's actually number 0. So, for example, if you wanted to use the first item in the list called `ledPins`, you would type:

```
ledPins[0]
```

If you want to turn on an LED on the pin number stored in the third item in the array, you would type:

```
digitalWrite(ledPins[2], HIGH);
```

# Making Your Intentions Known

So, how do you create a new array? You don't just create a variable that holds a single number, like an **int** or **float**. The process involves more steps than that.



A **float** is a data type for numbers that aren't whole numbers but include a decimal place such as 1.3 or -54.089.

The first step is to **declare** the new variable that will hold the array. *Declare* is just a fancy word for something you have already been doing in your sketches. It means creating a new variable by giving it a name and a data type. The following line of code declares a new variable called **ledPin** that contains an **int**:

```
int ledPin;
```



**Declaring** a variable is where you create a new variable by giving it a name and a data type such as **int**. The variable does not hold a value until it is given its first value.

If you already know what value you want to store in the variable when you create it, you can **instantiate** it at the same time. That just means giving it a starting value:

```
int ledPin = 13;
```



**Instantiating** a variable is where you give it a value for the first time. Instantiation can happen at the same time you declare the variable, or you can do it later, but the declaration always needs to come first.

You don't have to declare and instantiate the variable at the same time.; you can declare a variable and instantiate it later on in your code:

```
int ledPin;  
// some more code happens here  
ledPin = 13;
```



Be careful if you decide not to instantiate a variable at the same time that you declare it. You can't use that variable until it has a value, or your Arduino code might not work as you would expect.

To declare and create a new array of six integers called **ledList**, use the following code:

```
int ledList[6];
```

The preceding code is only a little different from what you would type to create a new variable that holds an **int**. After the variable name **ledList**, there is **[6]**. The **[ ]** means that instead of a single **int**, the variable is going to hold an array of **ints**. The **6** is the

number of **ints** the array will hold.

Now that you have created the array by declaring it, you can instantiate it, and fill it with values later in the code:

```
ledList[0] = 3;  
ledList[1] = 4;  
ledList[2] = 5;  
ledList[3] = 6;  
ledList[4] = 7;  
ledList[5] = 8;
```

Declaring an array and not instantiating it at the same time is useful if you don't know what values need to go into the array when you create it. However, if you already know what all the values should be, you can instantiate the array at the same time you declare it, as shown here:

```
int ledList[] = {3, 4, 5, 6, 7, 8};
```

The values to be stored in the array are listed between **{** and **}**. The **[]** no longer needs a number in it, because the number of items in the array is established by the number of items in **{}**.

## Looping Through an Array

It's easy to do the same thing with each item in an array without copying and pasting the same code multiple times. You can use a **for** loop to do this. As you might remember from earlier adventures, a **for** loop has three parts that determine how many times it is run. A new variable, often named **i**, is created and used to count through the loop. In the following code, the loop will run six times:

```
int i;  
for(i=0; i<6; i++) {  
}  
}
```

**i++** indicates that the variable **i** increases each time the computer runs through the loop. Instead of accessing a single item in the array, such as **ledList[3]**, the variable **i** can be used to access the next item in the list each time through the loop. Here's an example:

```
int i;  
for(i=0; i<6; i++) {  
    Serial.println( ledList[i] ); // print the next item in the list  
}
```

You can also write a **for** loop so that it makes a change to each item in the array:

```
int i;  
for(i=0; i<6; i++) {  
    ledList[i] = i+2 // add 2 to each item in the array  
}
```

## DIGGING INTO THE CODE



The code snippet `i++` is shorthand for `i=i+1`, but that's not the only useful code snippet. If `i++` increases `i` by adding `1` to it and saving the new number in `i`, you might be able to guess what `i--` does. That's right; it subtracts `1` from `i` and saves the new number in `i`.

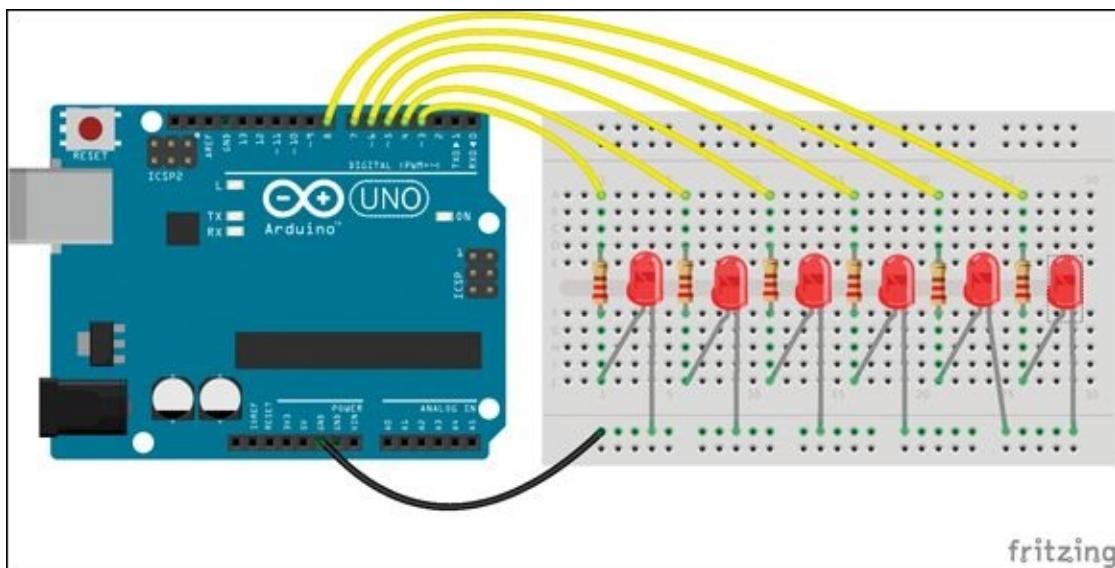
Another way to increase `i` by `1` is to write `i+=1`; similarly you can decrease `i` by `1` by writing `i-=1`. You can use these conventions to increase and decrease a variable by any number, so to increase `i` by `3` it would be `i+=3` or to decrease `i` by `7` it would be `i-=7`.

## Putting It Into Practice

That's enough talking about what happens in code. It's time to light up some LEDs and actually see what happens in code!

This circuit is one you have seen many times before in other adventures—an LED with a current limiting resistor. This time, you're going to set up six of them on digital Pins 3 to 8, as shown in [Figure 5-3](#):

1. Start by using a jumper wire to connect a GND on the Arduino Uno to a long row on the bottom of the breadboard. If your breadboard is labelled with a blue or black line or a –, connect GND to that row.
2. Place the six LEDs across the breadboard by inserting the short leg of each LED into the long row now connected to GND. Spread them out evenly across the breadboard.
3. Place the long leg of each LED into a short row that is easy for that LED to reach. It doesn't matter which rows you use; the only important thing is that each LED is in its own short row.
4. Connect one leg of a resistor to each short row that you just inserted the LED into. Bend the legs of the resistors so that they reach over the gap in the middle of the breadboard, and insert the other resistor leg into the short row across the gap.
5. Connect each resistor to an output pin on the Arduino Uno. Use six jumper wires to connect Pins 3 through 8 to the resistors.



[Figure 5-3](#) The circuit for an array of LEDs

The following code lets you put into practice setting up an array of values. Here, each of the items in the array is a pin number that controls an LED. So the whole array of LEDs can be blinked in the same way you would blink a single LED. The `pinMode()` is set for each of the pins, and each pin is set to `HIGH` and then `LOW`:

```
// pins for leds
int ledList[] = {
  3, 4, 5, 6, 7, 8};
// number of pins
```

```
int numPins = 6;

void setup() {
    int i;
    for(i=0; i<numPins; i++){
        // set pins to OUTPUT
        pinMode(ledList[i], OUTPUT);
    }
}

void loop() {
    // blink the LEDs one by one
    int i;
    for(i=0; i<numPins; i++) {
        // turn on the led
        digitalWrite(ledList[i], HIGH);
        delay(500);

        // turn off the led
        digitalWrite(ledList[i], LOW);
        delay(500);
    }
}
```



What happens if you change the `for` loop in the `loop()` function to `for(i=numPins-1; i>=0; i--)`?

# DIGGING INTO THE CODE



Your six LEDs are now blinking on and off, one by one, but how is the code making that happen? Let's start at the top of the sketch. There you see the variables being used. The first is an array `ledList` that holds all the pin numbers for the LEDs, and the second is an int, `numPins`, that stores the total number of LED pins:

```
// pins for leds
int ledList[] = {
  3, 4, 5, 6, 7, 8};
// number of pins
int numPins = 6;
```

Going on in the code, you come to your first `for` loop of the sketch in `setup()`. The `for` loop is used to set the pin mode of each of the LED pins to `OUTPUT`. The variable `i` is used to iterate through each item in the array:

```
int i;
for(i=0; i<numPins; i++){
  // set pins to OUTPUT
  pinMode(ledList[i], OUTPUT);
}
```

The second `for` loop of the sketch is in `loop()`. The variable `i` is used again to iterate through the array. This time, instead of setting the pin mode for each pin, the pin is turned on by being set to `HIGH`. The Arduino Uno waits for 500 ms when `delay(500)` is called, and then the pin is turned off by being set to `LOW`. The Arduino Uno is paused for 500 ms again before going on to the next pin in the array.

```
// blink the LEDs one by one
int i;
for(i=0; i<numPins; i++) {
  // turn on the led
  digitalWrite(ledList[i], HIGH);
  delay(500);

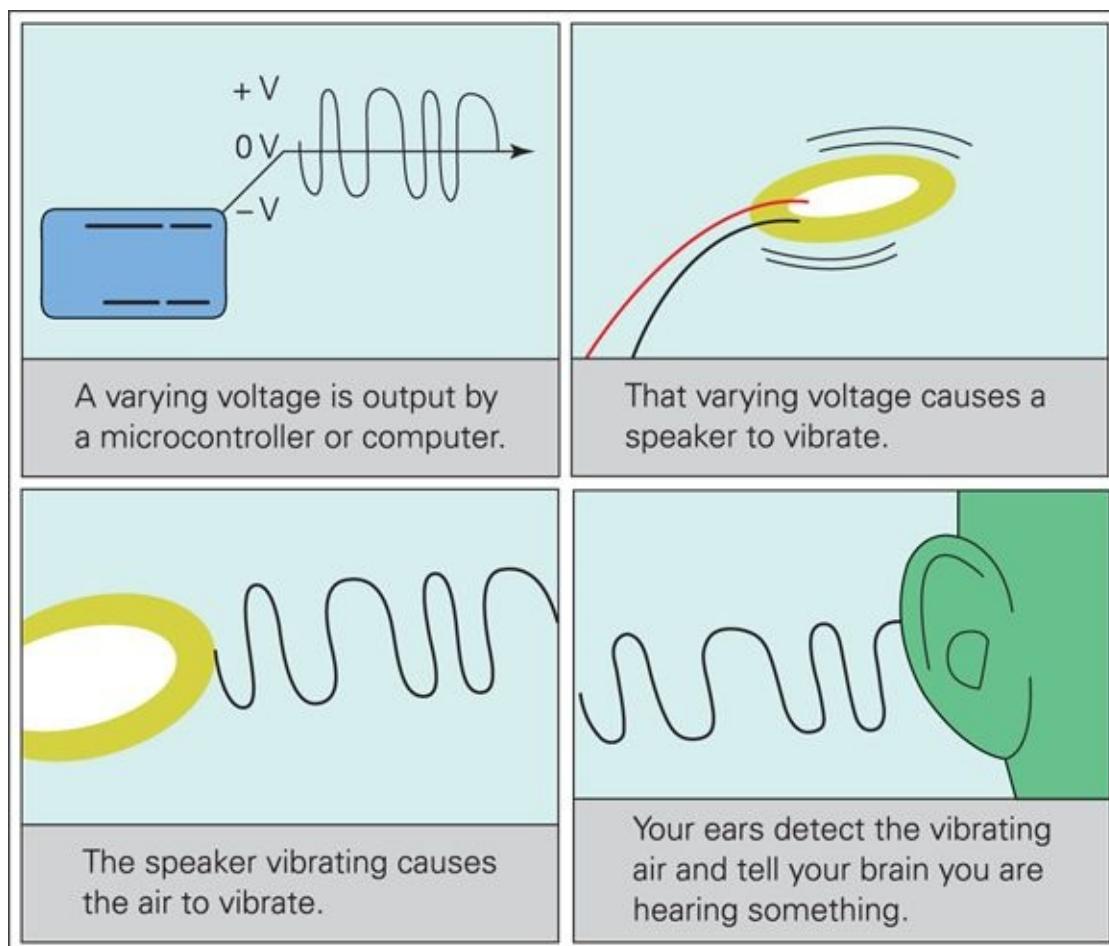
  // turn off the led
  digitalWrite(ledList[i], LOW);
  delay(500);
}
```

Because the `for` loop is inside `loop()`, it gets continuously run until the Arduino Uno is no longer powered.

# Making Noise

It's not unusual for computers to make sounds. You probably listen to music stored on a music player or phone (which are types of computers) all the time, but you might not have given much thought to how the computer physically creates the sound you hear.

Sound is just vibrations (usually vibrations in the air) that our ears can detect. The speed at which something vibrates determines whether it sounds low or high. The vibrations are measured in Hertz (Hz), which is equivalent to cycles per second. Humans can hear around 20 Hz to 20,000 Hz, though as we get older we tend to not hear high frequencies as well as we do when we are younger. [Figure 5-4](#) illustrates how sound is made.



[Figure 5-4](#) How sound is made

So, how does a computer get air to vibrate? A computer or Arduino Uno can output a changing voltage that alternates between positive and negative. A loudspeaker takes that changing voltage and turns it into vibrations. One type of loudspeaker uses a **piezo** element. This usually looks like a gold disc with two wires coming from it, although it is sometimes enclosed in plastic.

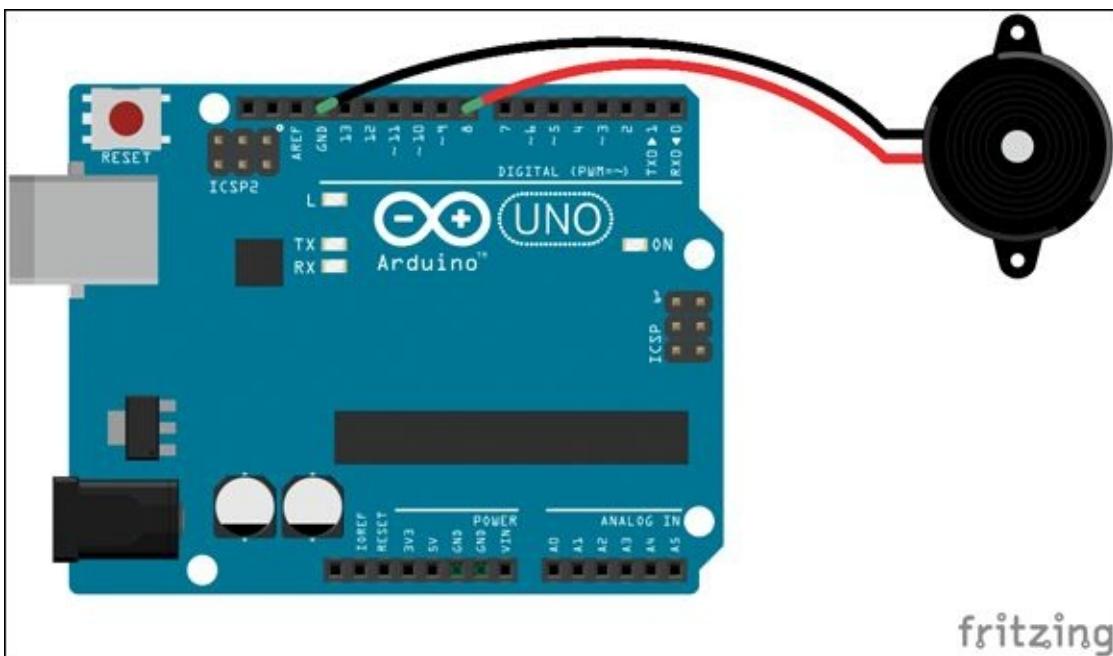


A **piezo** is a crystal that expands and shrinks when electricity is run through it. It also generates electricity when it is squeezed or bent.

Now that you know a little bit about how computers generate sound, you can make your Arduino produce some tones. You know that the Arduino outputs 0 and 5V because you have used `digitalWrite()` to set a pin to `LOW` (0V) and `HIGH` (5V). Making an Arduino board output a voltage between 0 and 5V is actually a tricky thing to do. Luckily the Arduino library has a built-in set of functions that does this and generates sound without you needing to know the details. Of course, if you would like to know more about how sound is generated, you can read more in the Arduino documentation at <http://arduino.cc/en/Reference/Tone>.

## Wiring the Circuit

A piezo has two wires attached to it. The black wire connects to **GND** on the Arduino board and the red wire connects to the pin that the audio plays from. Connect the black wire to GND and the red wire to Pin 8 now, as shown in [Figure 5-5](#).



[Figure 5-5](#) The circuit for a using a piezo as a speaker

## Writing the Code

The main function for creating sound is `tone()`. It can be used in two ways. The first is to give three arguments: the pin that the sound should play from; the frequency of the sound; and how long the sound should play. Here's an example:

```
// play on Pin 8 a tone of 750 Hz for 1000 ms (1 s)
tone(8, 750, 1000);
delay(1000);
```

Notice that right after the `tone()` function there is a `delay()` function. Even if the `tone()` function is told how long to play a sound, the Arduino board still needs to be told to wait for the sound to finish playing before `tone()` is called again. The `delay()` function specifies how long that wait should be.

The other way to use the `tone()` function is to give it only two arguments: the pin that the sound should play from; and the frequency of the sound. The sound starts playing when the function is called, and it doesn't stop until the function `noTone()` is called, as in the following code:

```
// play on Pin 8 a tone of 750 Hz
tone(8, 750);

// other code can happen here

// stop the tone
noTone(8);
```

The functions `tone()` and `noTone()` can be used in `setup()` if you want sound to play only once when the Arduino is first powered, or in the `loop()` if the sound should play repeatedly.

As you have already connected the piezo to the Arduino board, all that remains is for you to upload the following code:

```
int piezoPin = 8;

void setup() {
    // play 3 tones when the board first starts
    tone(piezoPin, 523, 200);
    // delay is slightly longer than tone
    // so that there is silence in between the sounds
    delay(210);

    tone(piezoPin, 784, 200);
    delay(210);

    tone(piezoPin, 1047, 250);
    delay(260);
}

void loop() {
    // play 5 more tones
    tone(piezoPin, 523, 200);
    delay(210);
```

```
tone(piezoPin, 587, 200);
delay(210);

tone(piezoPin, 659, 200);
delay(210);

tone(piezoPin, 698, 200);
delay(210);

tone(piezoPin, 784, 200);

// wait 5 seconds before starting the loop over
delay(5000);
}
```

When the Arduino first runs the code, you should hear three notes magically ringing out. The three notes should then play in a repeated pattern (that goes on and on and on...).

# DIGGING INTO THE CODE



So what exactly is happening in the sketch you just uploaded? The format of the sketch is similar to the others you have worked with: Variables are created at the top, anything that happens only once is done in `setup()`, and everything else that happens repeatedly is done in `loop()`.

This sketch only has one variable:

```
int piezoPin = 8;
```

The `setup()` in previous sketches has been used to set pin modes. That doesn't need to happen in this sketch because the only pin being used is outputting sound using `tone()`. Instead, you use `tone()` to demonstrate the difference between `setup()` and `loop()`. The `setup()` plays three tones before the `loop()` plays five tones. The tones in `setup()` only play once, when the Arduino Uno is first started, but the five tones keep playing and playing until you remove power from the board.

The `setup()` only calls two functions. The first is `tone()`, which tells the Arduino Uno what frequency to play on what pin and for how long. Then `delay()` is called to make the Arduino Uno wait for the `tone()` to finish playing before continuing onto the next line of code.

```
// play 3 tones when the board first starts
tone(piezoPin, 523, 200);

// delay is slightly longer than tone
// so that there is silence in between the sounds
delay(210);

tone(piezoPin, 784, 200);
delay(210);

tone(piezoPin, 1047, 250);
delay(260);
The loop() is just like setup() except it plays five tones instead of three:
// play 5 more tones
tone(piezoPin, 523, 200);
delay(210);

tone(piezoPin, 587, 200);
delay(210);

tone(piezoPin, 659, 200);
delay(210);

tone(piezoPin, 698, 200);
delay(210);

tone(piezoPin, 784, 200);

// wait 5 seconds before starting the loop over
delay(5000);
```



If you know how to read sheet music, open File => Examples => 02.Digital => toneMelody. You can see

that there are two tabs in the Arduino IDE: one labelled `toneMelody` and another labelled `pitches.h`.

The tab `pitches.h` is a list of values like this:

```
#define NOTE_B0 31
```

Just pay attention to `NOTE_B0` and `31`. The first is the musical note B and the number (31, in this example) is the frequency for that note. The number next to the note (like the `0` in `B0`) is the octave. You can use these numbers to help you write musical melodies.

# CHALLENGE



The following code uses an array to play the same sequence of sounds. Create a new sketch with the following code and upload it to your Arduino Uno. Try changing the array so it plays the pitches in reverse (high to low instead of low to high):

```
int piezoPin = 8;
int pitches[] = {
  523, 587, 659, 698, 784};
int numPitches = 5;

void setup() {
  // play 3 tones when the board first starts
  tone(piezoPin, 523, 200);
  // delay is slightly longer than tone
  // so that there is silence in between the sounds
  delay(210);

  tone(piezoPin, 784, 200);
  delay(210);

  tone(piezoPin, 1047, 250);
  delay(260);
}

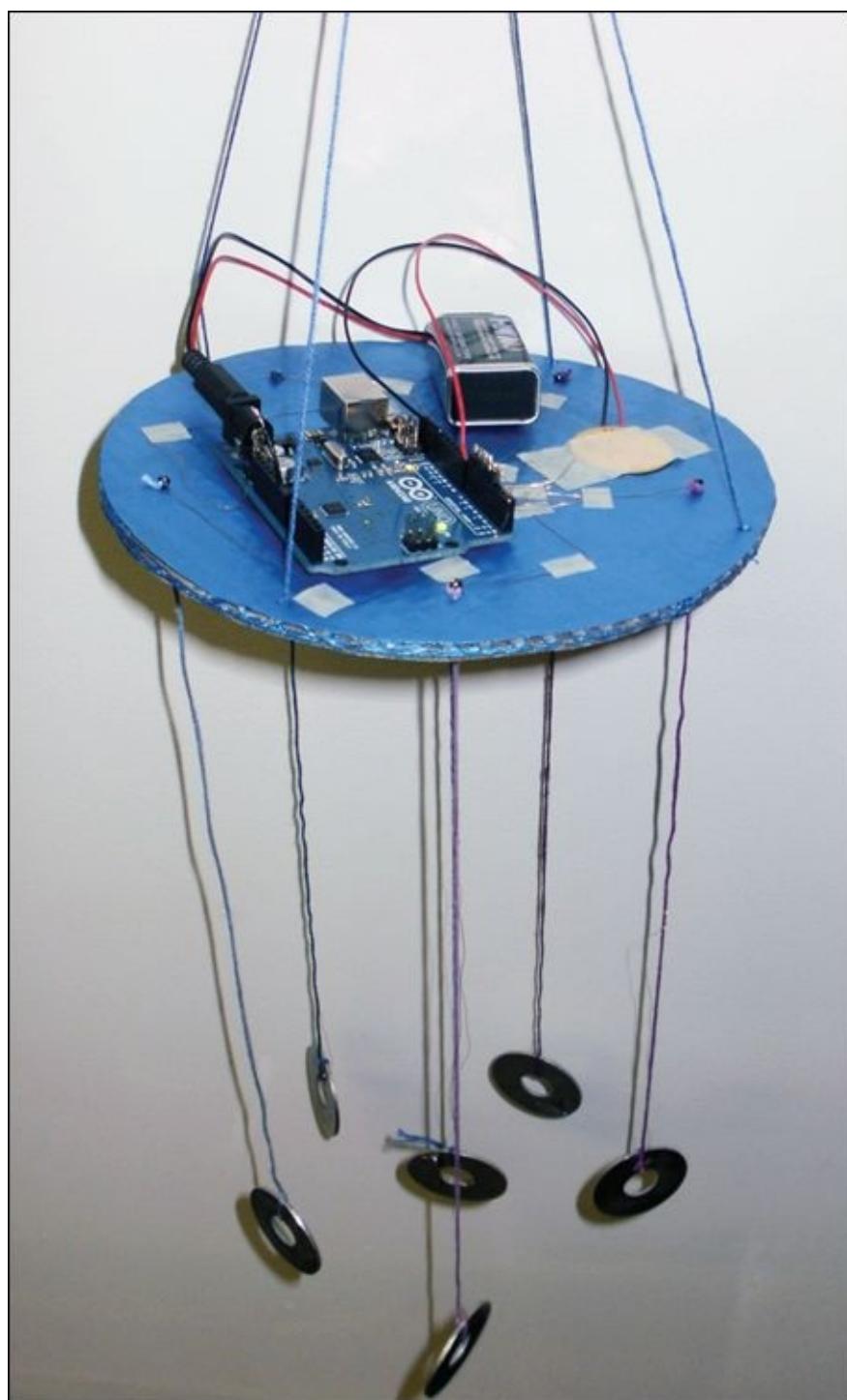
void loop() {
  // play 5 more tones
  int i;
  for( i=0; i<numPitches; i++){
    tone(piezoPin, pitches[i], 200);
    delay(210);
  }

  // wait 5 seconds before starting the loop over
  delay(5000);
}
```

# Building an Augmented Wind Chime

An augmented musical instrument is what can be called a “normal” musical instrument—such as a trumpet or piano—that has electronics added to it. The electronics let the musician use a computer or microcontroller to add to the sound that the instrument naturally makes.

As you have just discovered, the Arduino has a set of functions that can generate sound. You know that the Arduino can also read in information from its pins, so you can combine reading from pins to trigger different sounds. To complete this adventure in sound, you’re going to augment a wind chime (see [Figure 5-6](#)) so that it doesn’t just make the usual sounds when it’s buffeted by the wind but also plays tones produced by the Arduino!



## Figure 5-6 An augmented wind chime



You can watch a video on how to make your augmented wind chime on the companion site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).



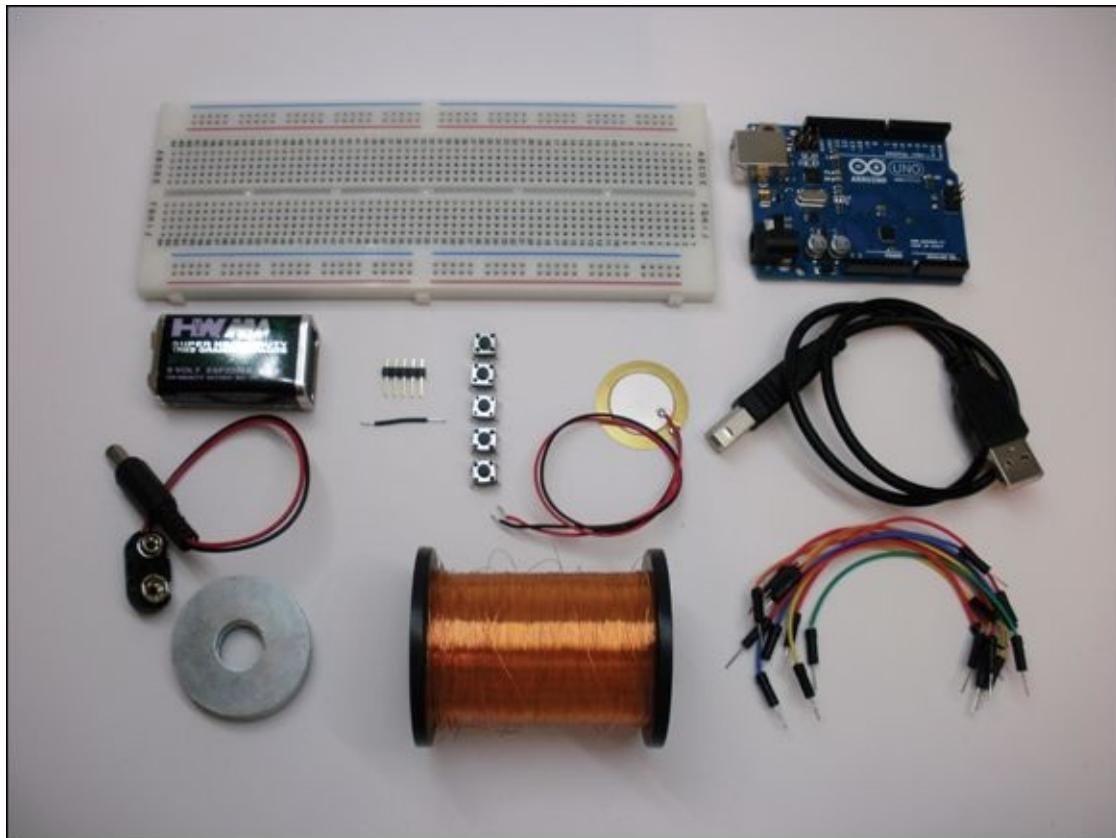
There are lots of augmented instruments that you can read about and learn from! Here are some of my favourites, but you can do some research online and find even more:

- Magnetic Resonance Piano: [www.eecs.qmul.ac.uk/~andrewm/mrp.html](http://www.eecs.qmul.ac.uk/~andrewm/mrp.html)
- Digi Didgeridoo: <http://createdigitalmusic.com/2009/12/digi-didgeridoo-augmented-wireless-digital-instrument-with-aboriginal-roots>
- Augmented Beatboxing: [www.cs4fn.org/music/beatboxing.php](http://www.cs4fn.org/music/beatboxing.php)

## What You Need

You need the following items to build your augmented wind chime. [Figure 5-7](#) shows the electronic components you need.

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 12 jumper wires
- 5 tactile pushbuttons
- A strip of 5 header pins
- A 9V battery
- A 9V battery-to-DC barrel jack connector
- Some thin wire
- Small piece of solid core wire
- Some string, ribbon or yarn
- 10 beads (plastic or glass large enough to pass string or ribbon)
- 6 washers or other conductive object to act as the chimes
- Some stiff cardboard or plastic to use as the base
- Masking tape
- A soldering iron
- Some solder
- Scissors or a utility knife
- A pencil or hole punch
- A multimeter with continuity test

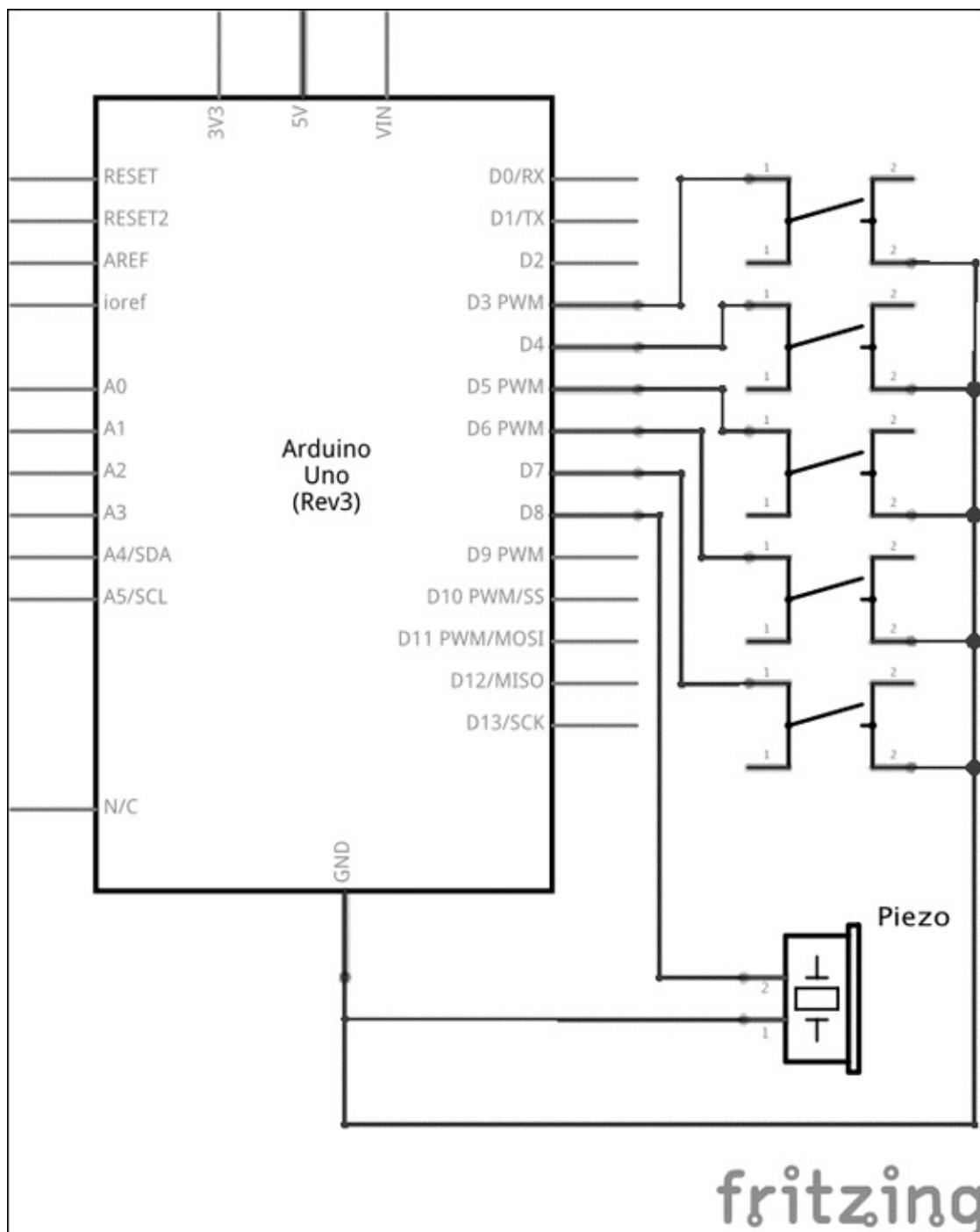


**Figure 5-7** The electronic components you need to make the wind chime

## Understanding the Circuit

There's not too much to the circuit for the wind chime; it combines switches with a piezo speaker. The fun part is that the final circuit uses materials that you don't normally see in electronics. Before experimenting with new materials, it's a good idea to build a prototype of the circuit on a breadboard to make sure it works as you expect it to.

[Figure 5-8](#) shows the circuit schematic for the wind chime. It looks very different to the finished wind chime! The chimes act as switches, even though they don't look like normal switches.



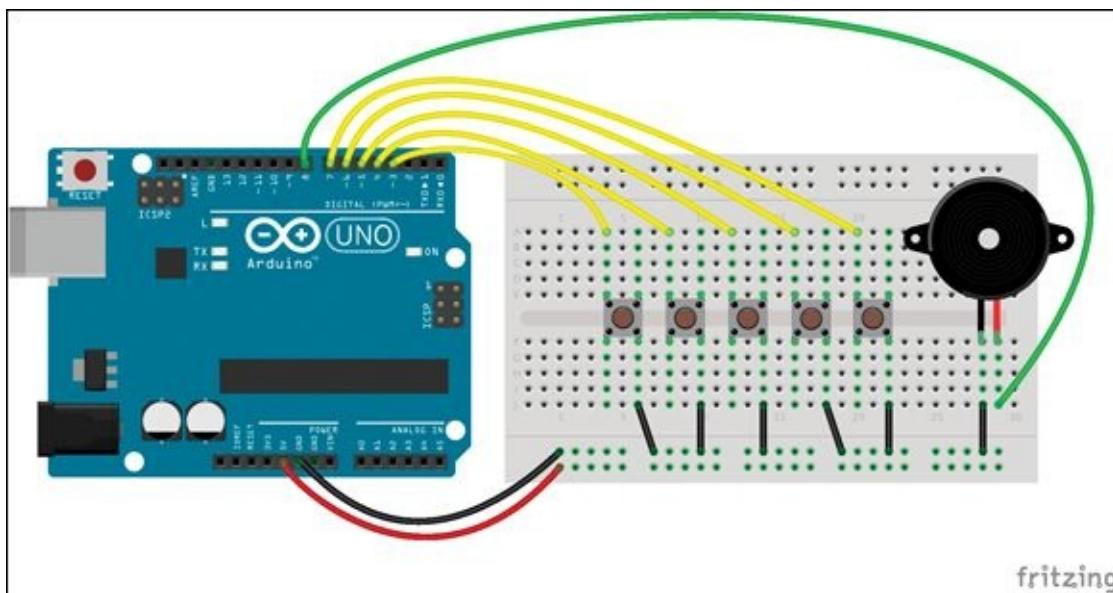
[Figure 5-8](#) Circuit schematic for the augmented wind chime

## Prototyping on a Breadboard

The circuit consists of five switches on digital pins and one piezo on another digital pin. Because you learned how to use the internal pull-up resistor in the Arduino Uno in [Adventure 3](#) instead of needing to add a resistor to the switch circuits, there aren't that many components.

Build the circuit shown in [Figure 5-9](#):

1. Place five tactile pushbuttons across the gap in the middle of the breadboard so that two of the legs are inserted in short rows above the gap and the other two legs are inserted in rows below the gap.
2. Use a jumper wire to connect a GND pin on the Arduino Uno to a long row along the bottom of the breadboard. If the breadboard is labelled with a black or blue line or a -, use that row.
3. Use five jumper wires to connect the bottom-right leg of each pushbutton to the long row connected to GND.
4. Use five jumper wires to connect the upper-left leg of each pushbutton to Pins 3 through 7 on the Arduino Uno.
5. Connect the red wire of the piezo to any empty short row on the breadboard. Use a jumper wire to connect that short row with Pin 8 on the Arduino Uno.
6. Connect the black wire of the piezo to the long row connected to GND.



[Figure 5-9](#) Breadboard prototype circuit

The wind chime circuit is a little different from the circuit in [Adventure 3](#) that used a switch. Instead of pushbutton switches, you're using conductive metallic items (such as washers or anything else you choose) connected to wires dangling from the wind chime's base. Each of these conductive items is connected to a digital pin. A sixth conductive item that's connected to GND hangs in the middle. When the wind causes the middle conductive chime to come into contact with one of the chimes connected to a pin, it is electrically the same as pushing the button on the combination safe in [Adventure 3](#).

## Writing the Code

Launch the Arduino IDE and open a new sketch. Type the following code in your sketch:

```
int chimes[] = {  
    3, 4, 5, 6, 7}; // array of pins for chimes  
int numChimes = 5; // total number of chimes  
int piezoPin = 8; // pin for piezo  
  
void setup(){  
    int i;  
    // set pinMode on all the chimes pins  
    for(i=0; i<numChimes; i++) {  
        pinMode(chimes[i], INPUT_PULLUP);  
    }  
}  
  
void loop(){  
    int i;  
    for(i=0; i<numChimes; i++) {  
        // read in value on pin  
        int value = digitalRead(chimes[i]);  
        // if LOW (meaning it has connected to ground)  
        if(value == LOW) {  
  
            // play the sound  
            tone(piezoPin, (100*i)+200, 30);  
            delay(1);  
        }  
    }  
}
```

Upload the sketch to your Arduino Uno connected to the tactile pushbuttons and piezo circuit you just built. You should hear a different pitch played when each button is pressed.

Of course, you can change what is played when each chime is triggered. Come up with your own musical algorithm!

# DIGGING INTO THE CODE



The main parts of the sketch use an array, a `for` loop to iterate through that array, and the `tone()` function. The array `chimes[]` stores each pin number that is connected to the conductive chimes (or prototyped with a pushbutton).

```
int chimes[] = {  
    3, 4, 5, 6, 7}; // array of pins for chimes  
int numChimes = 5; // total number of chimes  
int piezoPin = 8; // pin for piezo
```

The `setup()` then sets the pin modes for each of the pins connected to the pushbuttons or chimes. Because they are inputs that use the internal pull-up resistors in the Arduino Uno, you use the argument `INPUT_PULLUP`.

```
int i;  
// set pinMode on all the chimes pins  
for(i=0; i<numChimes; i++) {  
    pinMode(chimes[i], INPUT_PULLUP);  
}
```

The `for` loop in the `loop()` checks the value of each pin. When a chime connects to `GND`, a `tone()` is played. Because you've used an internal pull-up resistor, you know that the value of a pushbutton or chime is `HIGH` when it's not connected to ground, and then the value changes to `LOW` when it is connected to ground. The frequency of the `tone()` is determined by which chime triggered the sound.

```
int i;  
for(i=0; i<numChimes; i++) {  
    // read in value on pin  
    int value = digitalRead(chimes[i]);  
    // if LOW (meaning it has connected to ground)  
    if(value == LOW) {  
        // play the sound  
        tone(piezoPin, (100*i)+200, 30);  
        delay(1);  
    }  
}
```

## Making the Wind Chime

Now that you know that your sketch is working correctly and you have built a test circuit on your breadboard, you are ready to make your wind chime. The wind chime is constructed from a base from which hang six chimes—five outer chimes and a grounded inner chime. You can make the wind chime from any materials you like, but it's important that the chimes are conductive and that they are electrically connected to the Arduino Uno. Visit a hardware store and look through all the small metal fastenings to choose what you want to use as chimes. Washers come in many different sizes, but you might prefer hexagonal nuts over circular washers.

## Making the Base

You can make the base from anything that is strong enough to support your chimes and can also hold an Arduino Uno and battery. Stiff cardboard or plastic are good options.

First, cut a circle from the base material that's approximately 6 inches in diameter. Poke six small holes in it; the strings and wires pass through these holes. Five of the holes should be evenly distributed around the outside of the base, and the sixth hole should be in the center.

Poke four more holes around the edges of the base. These are for the strings to hang the chime.

## Making the Chimes

When you choose what material to use to make the chimes, you need to remember two important characteristics: you need to be able to solder wires to it and it must conduct electricity. To test if you can solder to it, just try to do it! Whatever material you have chosen, it will probably take a few more seconds to get hot enough to solder than something small like a wire, so be patient.

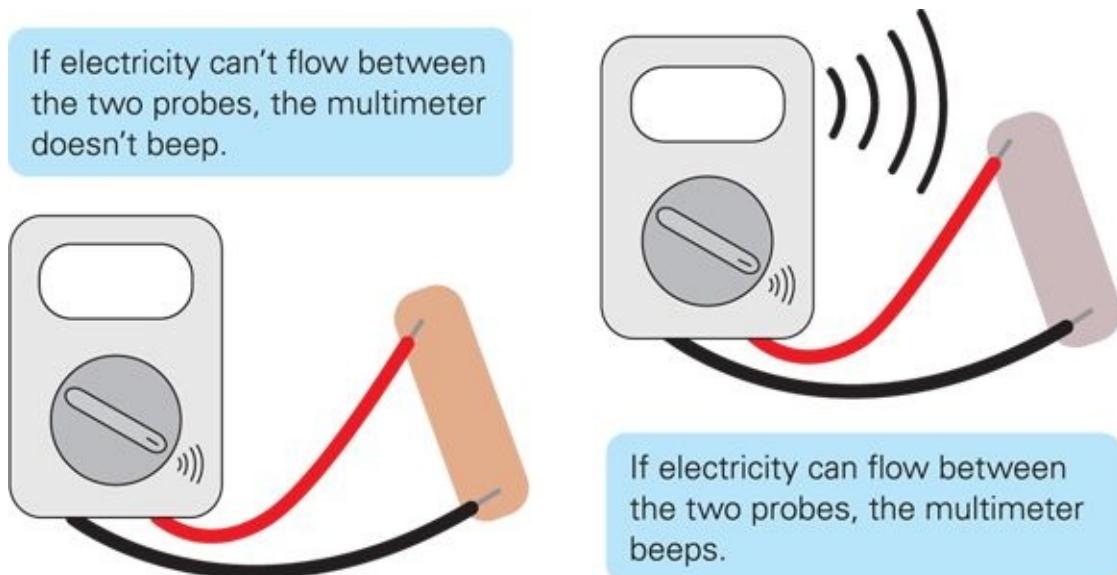


Only solder when an adult is nearby to help! It takes a lot of heat from the iron to get your chime hot enough to melt solder. It also takes a longer time to cool down after you've soldered your wire to it. Be very careful and wait at least 5 minutes before picking up something you've soldered.

To check whether the material conducts electricity, you need a multimeter. A multimeter measures multiple things (so it's a pretty good name), including voltage and resistance. You need a meter that also measures continuity. Continuity indicates whether current can flow between the two probes attached to the meter, which indicates conductivity. Not all multimeters have a continuity test, so pay attention to the listed features of the multimeter before you buy it.

Test your potential chime by touching it in two different spots on the chime with the probes (see [Figure 5-10](#)). If the multimeter beeps, your material is conductive. If it

doesn't, you should find something else to use.



**Figure 5-10** Conductivity test



If you are using very thin wire than doesn't have a plastic sleeve around it, it may still have a thin coating of insulation on it. If you have problems soldering it or it fails a continuity test, you can scrape off the insulating coating with some sandpaper or a nail file.

After you have chosen your material for your chimes, solder a wire to each of them. The wire should be long enough to reach from where you want the chime to hang through the base and to the Arduino Uno. Wrap the wire several times around your chime (see [Figure 5-11](#)) and make sure you have a strong electrical connection and then solder it to the chime.



**Figure 5-11** A chime

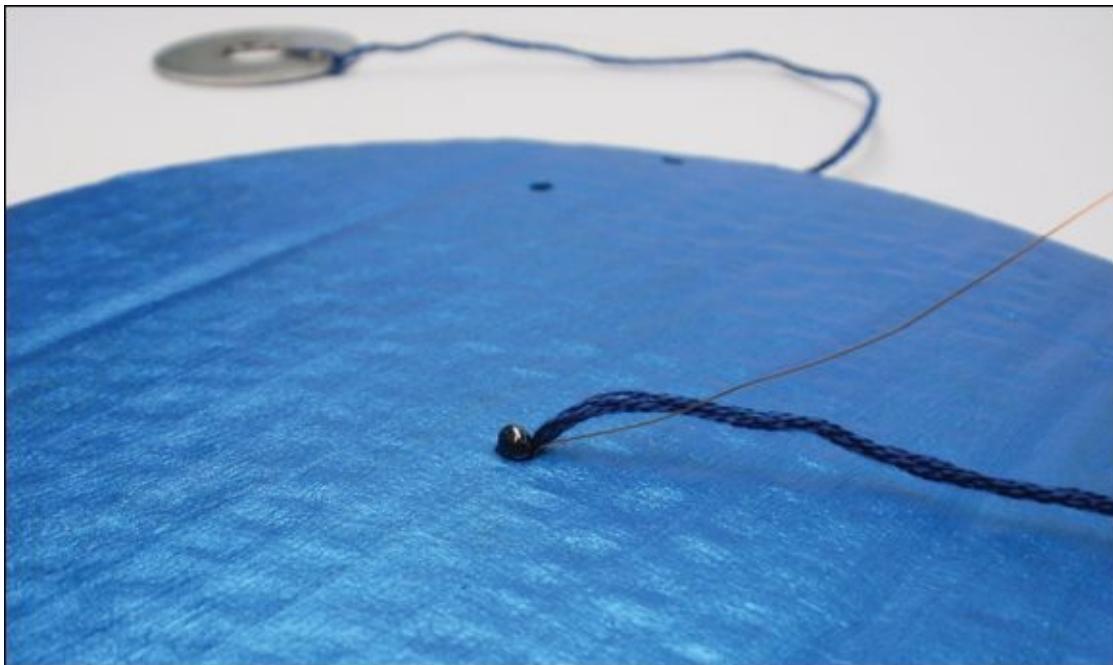
As yet, the wire is too weak to support the chime by itself. Chandeliers are held up by a strong chain with a wire that runs along it to light up the bulbs. In the same way, to give your wind chime extra support, the next section will show you how to use string or ribbon to hang the chime.

## Attaching the Chimes

The most important thing to remember is that the string or ribbon should bear all the weight of the chime; the wire is just there to conduct the signal from the Arduino to the chime. The wire should not bear any weight.

Tie a piece of string or ribbon around the chime as shown in [Figure 5-12](#). Bring the wire and ribbon up through the hole together and then thread a bead onto the string or ribbon—not the wire. Tie a knot in the string or ribbon to keep the bead in place and keep the chime from pulling the string back through the base (see [Figure 5-12](#)).

Do this with all six chimes.

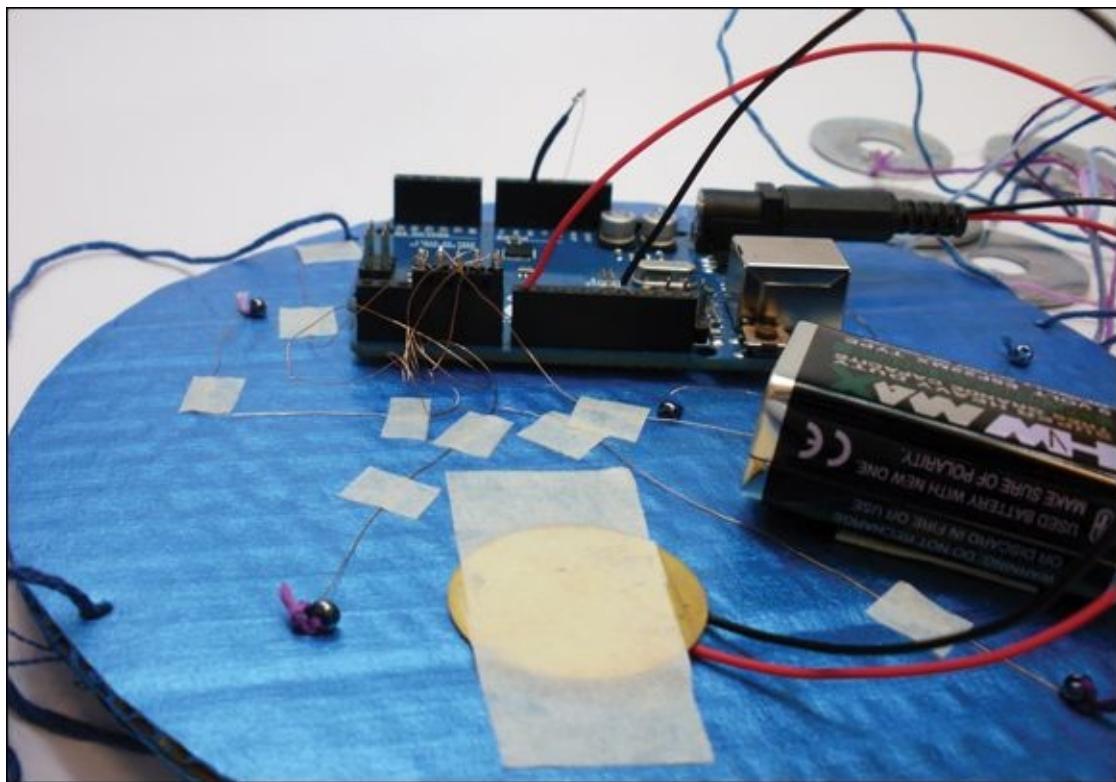


**FIGURE 5-12** A chime attached to the base

## Connecting the Electronics

You are almost finished! Now to complete the final few steps:

1. Solder a small section of solid-core wire to the chime hanging from the center. Plug that wire into GND on the Arduino board.
2. Tape the Arduino board and piezo to the top of the base. Connect the black wire of the piezo to a GND pin on the Arduino board and connect the red wire to Pin 8.
3. Solder each of the wires from the five remaining chimes to a pin on the section of header pins. This is some tricky soldering, so just take your time. Place the header pins in a breadboard to hold the pins upright. Heat up a header pin with the soldering iron and coat it in solder. Then heat up the end of the wire and coat it in solder. Place the wire so it touches the header pin and heat up both again so that the solder coating them melts and connects them together. Repeat this process for the remaining pins.
4. Push the header pins into Pins 3 through 7.
5. Connect the battery to the Arduino board using the battery holder (see [Figure 5-13](#)) and attach the battery to the base using tape.



**Figure 5-13** Top of base

Cut two more pieces of string about 24" long. These will hang your wind chime. Fold them each in half and thread each end of the strings through one of the four holes on the base. The ends should stick out from the bottom of the base. Tie a bead onto each string to keep the string from pulling back through the base. Hang up your wind chime, and enjoy your augmented sounds!

# Further Adventures with Sound

You are now a sound savant! You can control sound along with light and motion from your circuit and code knowledge. Plus, you now know that you don't have to go to a special shop to buy materials for your circuits. You can test whether something conducts electricity and start using everyday household items in your circuits.

If you'd like to read more about how to use `tone()`, visit the Arduino documentation at <http://arduino.cc/en/reference/tone>.

You might want to check out some of these other examples and tutorials online:

- <http://arduino.cc/en/Tutorial/Tone>
- <http://arduino.cc/en/Tutorial/Tone2>
- <http://arduino.cc/en/Tutorial/Tone3>
- <http://arduino.cc/en/Tutorial/Tone4>
- <http://itp.nyu.edu/physcomp/labs/labs-arduino-digital-and-analog/tone-output-using-an-arduino/>
- <https://learn.adafruit.com/adafruit-arduino-lesson-10-making-sounds>

If you are curious about exactly how the Arduino makes sound, check out the Wikipedia page on Pulse Width Modulation (PWM) at [http://en.wikipedia.org/wiki/Pulse-width\\_modulation](http://en.wikipedia.org/wiki/Pulse-width_modulation). [Adventure 6](#) shows you how PWM is used to control light instead of sound.

**Arduino Command Quick Reference Table**

Command	Description
<code>[]</code>	Indicates that the variable is an array of variables rather than a single variable. See also <a href="http://arduino.cc/en/Reference/Array">http://arduino.cc/en/Reference/Array</a> .
<code>tone</code>	Plays a sound with a given frequency. If a duration is given as well, the sound plays only for that length of time; otherwise, the sound plays until <code>noTone</code> is called. See also <a href="http://arduino.cc/en/Reference/Tone">http://arduino.cc/en/Reference/Tone</a> .
<code>noTone</code>	Stops the tone from playing. See also <a href="http://arduino.cc/en/Reference/NoTone">http://arduino.cc/en/Reference/NoTone</a> .



**Achievement Unlocked:** Inspirational engineer of sound!

## In the Next Adventure

In the next adventure, you use even more materials that you wouldn't expect in an electrical circuit. You also find out how to control a colour-changing LED!



## Adventure 6

### Adding Libraries

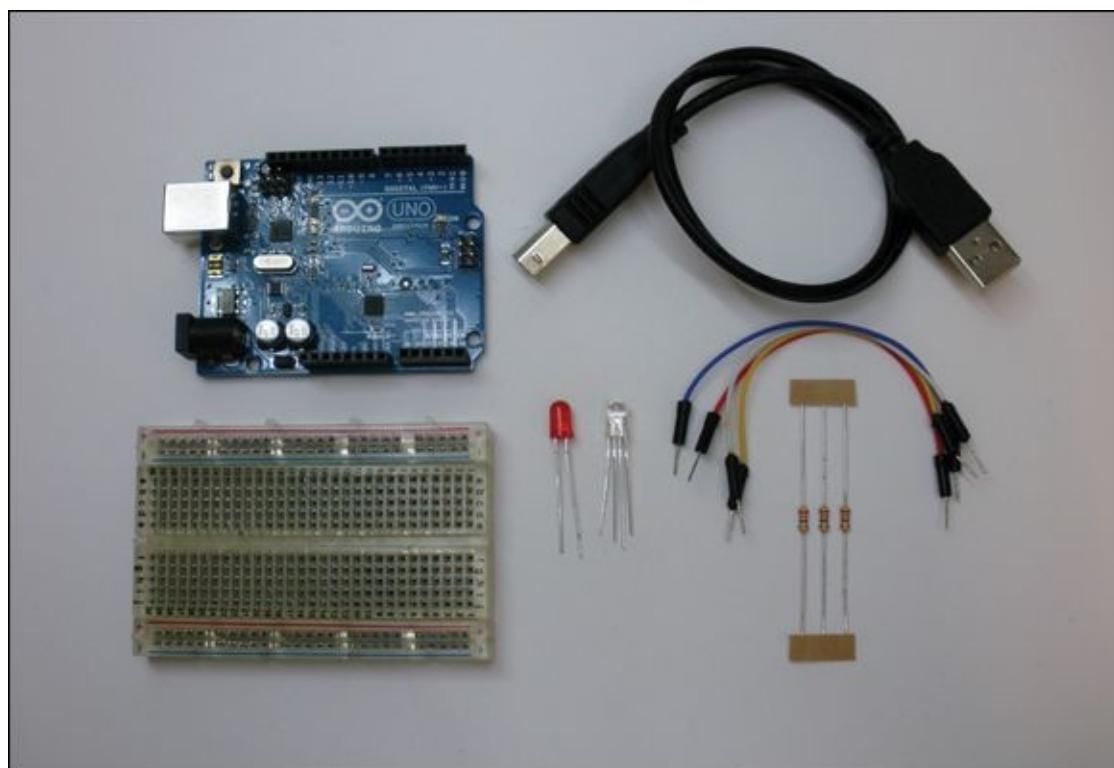
IT'S TIME TO push the boundaries of your Arduino! In previous adventures you used Digital and Analog Pins on your Arduino Uno, but what's the difference between a digital and analogue signal? You have output on a Digital Pin in [Adventures 1 through 3](#), have read in from a Digital Pin in [Adventures 2](#) and [5](#), and read in from an Analog Pin in [Adventure 2](#), but what about outputting analogue signals? Well, that comes next. But then what? After you've tackled outputting analogue signals, is that it? Is that the end of Arduino coding? Not at all!

You can push your Arduino even further by using libraries. This allows you easily to incorporate clever functions, which have been written by other people, into your sketches. In this adventure you will use a library that lets you turn (almost) anything you like into a touch sensor. You are then going to build a magical crystal ball that glows when you wave your hands over it.

# What You Need

You need the following items for the first part of this adventure. The electronic components are shown in [Figure 6-1](#).

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 4 jumper wires
- 1 LED
- 1 RGB common cathode LED
- 3  $220\Omega$  resistors
- 1  $10\text{ M}\Omega$  resistor



[Figure 6-1](#) The electronic components you need for the first part of this adventure

# Analogue Out

If you have completed [Adventures 1, 2 or 4](#), you know that `digitalWrite()` can do two things: output 5V when set to `HIGH` or output 0V when set to `LOW`. You then use `digitalRead()` to read in whether a pin is connected to 5V or 0V. You can check out [Adventure 3](#) if you need a refresher on using `digitalRead()` to read from a push button.

You also know from [Adventures 2](#) and [3](#) that if you want to measure a voltage on a pin that is between 0V and 5V, you need to use `analogRead()`. It returns a number between 0 and 1023 that corresponds to the input voltage. So it stands to reason that if you want to output a voltage between 0V and 5V, there is probably a function called `analogWrite()` that would let you do that. That's absolutely correct!

But first, what exactly is the difference between an analogue and digital signal? A **digital signal** is an electrical signal that can only be one of two things—either on or off. When represented in electricity, it's either 5V (on) or 0V (off). When represented in code, it's either `HIGH` (on) or `LOW` (off).

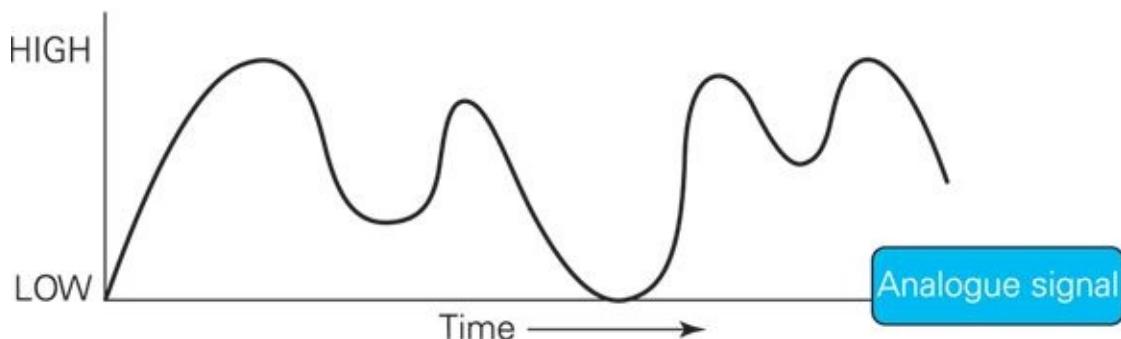
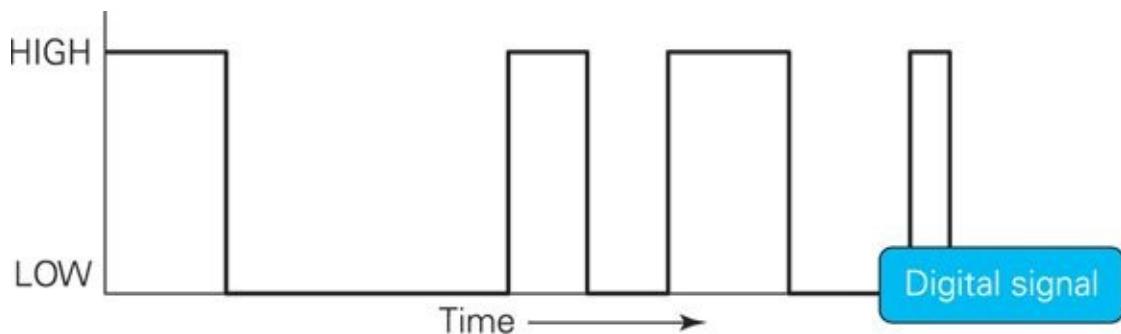


A **digital signal** is a signal that is only either on or off, `HIGH` or `LOW`. On the Arduino Uno, a `HIGH` signal is 5V and a `LOW` signal is ground.

An **analogue signal** is a signal that can be values between on and off. When represented in electricity, it can be any voltage between 0V and 5V. When represented in code, it can be any number between 0 and 1023. [Figure 6-2](#) further illustrates the difference between analogue and digital signals.



An **analogue signal** is a signal that varies between `LOW` and `HIGH`. On the Arduino Uno, an analogue signal can be measured as a number between 0 for ground and 1023 for 5V. An analogue signal can be output as a value between 0 for 0V and 255 for 5V.



**Figure 6-2** Analogue and digital signals

You can output a digital signal using `digitalWrite()` and can use any Digital Pin on the board. You can also read in a digital signal using `digitalRead()` using any Digital Pin and read in an analogue signal using any Analog Pin. To output an analogue signal, you use `analogWrite()`, but you can only use special Digital Pins. Outputting a value between `HIGH` and `LOW` is trickier for a microcontroller than outputting a digital signal, so there are only some of the pins can do that. These are nicely marked on your Arduino board with the `-` symbol. On the Arduino Uno, these are Pins 3, 5, 6, 10 and 11 (see [Figure 6-3](#)).



**Figure 6-3** The pins that support `analogWrite()`

The other important thing to know is that not all Arduino pins can use `analogWrite()`.



Reading and writing `HIGH` and `LOW` to and from a pin is super-easy for a microcontroller like the Arduino Uno to do. All the input and output pins on the Arduino board can do this—and are very good at it. You can even use analog pins to input or output a digital signal when you have run out of available digital pins. Reading in or outputting a voltage that is between `HIGH` and `LOW` is harder to do and requires some special functionality in the microcontroller. This is why `analogRead()` only works on `A0` through `A5`. And `analogWrite()` only works on pins marked with a `~`.

Just like `digitalWrite()`, `analogWrite()` takes two arguments. The first argument defines which pin should be used and the second determines what voltage should be output. This second argument is a little different from other arguments. You don't use `HIGH` or `LOW`; instead you use a number between 0 and 255, with 0 used for 0V and 255 for 5V. The following example code, outputs a signal on Pin 6 that is roughly one third of 5V, so set the second argument to 83:

```
analogWrite(6, 83);
```

If you wanted to output almost the maximum (5V), you would use a number just under 255:

```
analogWrite(6, 249);
```

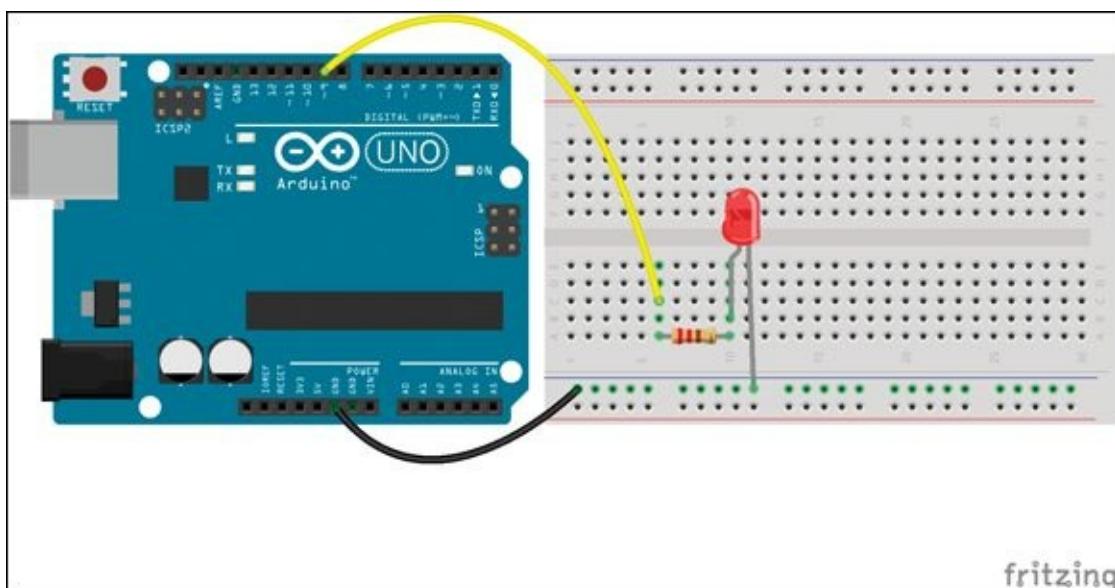
## Fading an LED

Why would you want to output a voltage between 0V and 5V? There are lots of reasons, but one common use is to smoothly fade an LED on and off.

Start the Arduino IDE and open File⇒Examples⇒03.Analog⇒Fading. Build an LED circuit with a current limiting resistor on Pin 9 (see [Figure 6-4](#)):

1. Use a jumper wire to connect a GND pin on the Arduino Uno to a long row along the bottom of the breadboard. If the breadboard is labelled with a black or blue line or -, connect the pin to that row.
2. Insert the short leg of the LED into the row now connected to a GND pin.
3. Insert the long leg of the LED into any nearby short row. Insert one leg of the resistor into the same short row.
4. Insert the other resistor leg into any other short row. Use a jumper wire to connect that short row to Pin 9 on the Arduino Uno.

Upload the example and see what happens. You should see the LED fade on and off.



[Figure 6-4](#) LED circuit for fading an LED

This example combines `analogWrite()` with the `for` loop you have already been using in [Adventures 3 through 5](#). The first `for` loop slowly increases the voltage output to the LED:

```
// fade in from min to max in increments of 5 points:  
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {  
  // sets the value (range from 0 to 255):  
  analogWrite(ledPin, fadeValue);  
  // wait for 30 milliseconds to see the dimming effect  
  delay(30);  
}
```

The second `for` loop does the opposite, and decreases the voltage:

```
// fade out from max to min in increments of 5 points:
```

```
for(int fadeValue = 255; fadeValue >= 0; fadeValue -=5) {  
    // sets the value (range from 0 to 255):  
    analogWrite(ledPin, fadeValue);  
    // wait for 30 milliseconds to see the dimming effect  
    delay(30);  
}
```

## CHALLENGE



► Increase the speed at which the LED fades on and off by adjusting the `delay()` time. Now try increasing the fading speed by changing `fadeValue`.

# DIGGING INTO THE CODE



Using the `analogWrite()` function actually requires a little bit of a trick, because it doesn't really output a steady voltage somewhere between 0V and 5V. In fact, it uses **pulse width modulation (PWM)**. You might have noticed that it says PWM on your Arduino board next to the `-` symbol. PWM outputs a signal that switches back and forth between on and off at different speeds. This happens so fast that when you light up an LED using PWM, your eyes don't see the switching back and forth. Instead they see something in between, like an LED at only half power.



**Pulse width modulation (PWM)** is how the Arduino board generates an output signal between 0V and 5V. The signal switches quickly between `LOW` and `HIGH` and the resulting output voltage is between the two voltages.

Figure 6-5 shows three example PWM signals. The top one is a signal that is mostly off, so the LED appears only dimly on. The middle signal is on half the time and off half the time, so the LED appears roughly half as bright as an LED set to `HIGH`. The bottom signal is on almost all the time, so the LED appears almost as bright as one that is set to `HIGH`.



Figure 6-5 Pulse width modulation examples

The ratio of how long the signal is output to `HIGH` versus `LOW` determines how bright the LED appears. This ratio is called the **duty cycle**. The more time the signal is `HIGH`, the higher the voltage appears to be and the brighter the LED seems to shine.



The **duty cycle** is the ratio of time a signal is **HIGH** versus **LOW** in a given cycle. In PWM, the higher the duty cycle, the higher the output voltage.

## Mixing Light

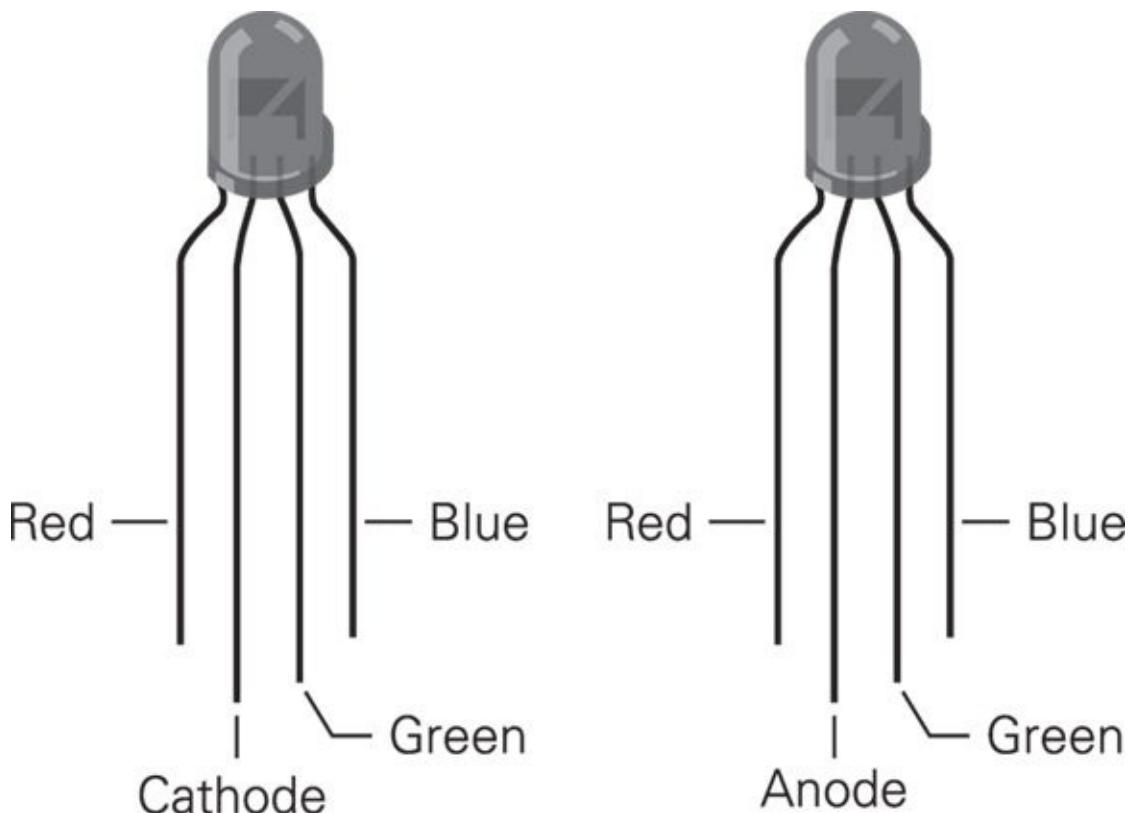
There are many different kinds of LED, and they come in all shapes and sizes. The kind you've used in previous adventures is called through-hole LEDs—that just means they have legs and if you want to attach them to a circuit board the board has to have holes for the legs to go through.

LEDs also come in many different colours. Red, yellow and green are the most common, but you can buy other colours such as blue and orange. There are also LEDs that are really three LEDs put into what looks like one LED. These are called **RGB LEDs**, which stands for red–green–blue LEDs.



An **RGB LED** (red–green–blue light-emitting diode) is a single LED with four legs that contains three lights: one red, one green and one blue. The three lights share either a common anode or a common cathode.

There are two kinds of RGB LED, and both of them have four legs (see [Figure 6-6](#)). Three of the legs are for the colours (red, green and blue). The fourth leg is a shared leg, either a shared positive leg (an **anode**) or a shared negative leg (a **cathode**). For both types, you need three current limiting resistors, as you would with three separate LEDs.



[Figure 6-6](#) RGB LEDs



An **anode** is the positive leg of a directional component, such as the long leg of an LED.



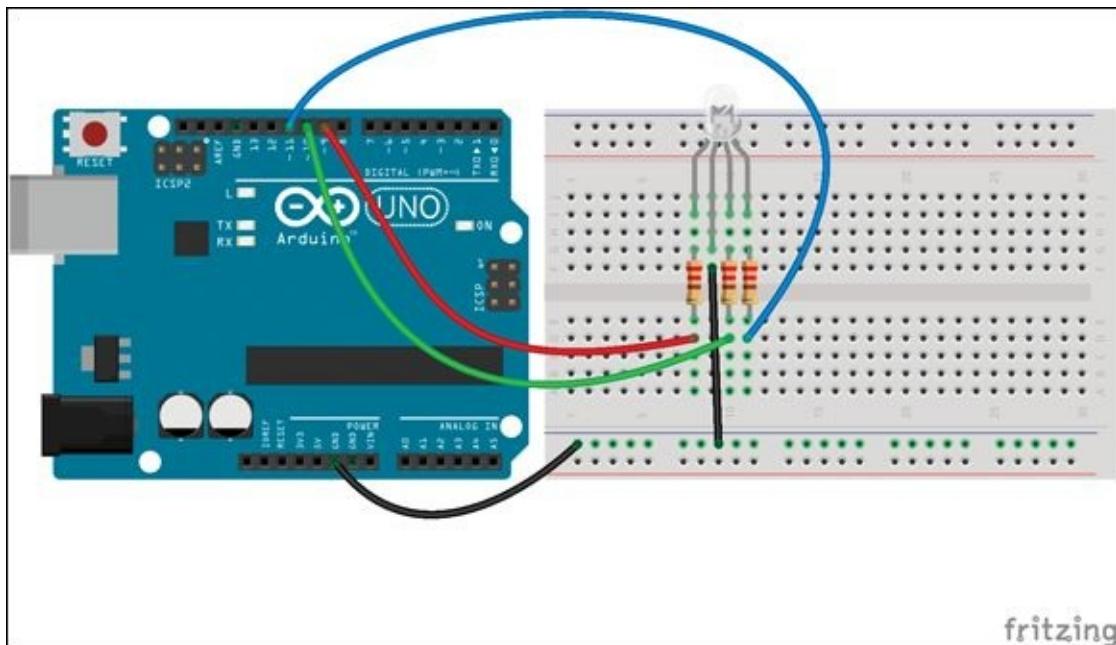
A **cathode** is the negative leg of a directional component, such as the short leg of an LED.

The big difference between the two kinds of RGB LED is that the common anode LED circuit shares a common power source, such as 5V, and the common cathode LED shares a common ground. It's a bit easier to think about how electricity flows with a common cathode RGB LED, so that's the one we'll use.

## Wiring the Circuit

The longest leg of the LED is the cathode—the leg that goes to ground. The other three legs are connected to the red, green and blue lights within the LED. To have the best control over the lights, use three Arduino pins that can use `analogWrite()`. Pins 9, 10 and 11 are good choices. Build the circuit shown in [Figure 6-7](#):

1. The only fiddly part when working with RGB LEDs is figuring out which leg controls which colour. The best way to determine this is to connect each LED leg individually to 5V and see what colour lights up. Remember to use the current limiting resistor; don't directly connect 5V to the LED.
2. Use a jumper wire to connect a GND pin on the Arduino Uno to a long row along the bottom of the breadboard. If the breadboard is labelled with a black or blue line or  $-$ , connect the pin to that row.
3. Place the RGB LED in the breadboard so that each leg is in its own short row. Use a jumper wire to connect the longest leg of the LED to the long row connected to the GND pin.
4. Insert one leg of a  $220\Omega$  resistor into the same row as each of the colour legs of the LED. Bend the resistor over the gap in the middle of the board, and insert the leg of each LED into its own short row.
5. Find the red leg and connect the resistor now connected to it to Pin 9 with a jumper wire; repeat to connect the resistor connected to the green leg to Pin 10; and the resistor connected the blue leg to Pin 11.



**Figure 6-7** Circuit connecting an RGB LED to an Arduino board

## Writing the Code

The code for controlling an RGB LED looks just like code that controls three LEDs. Create a new sketch in the Arduino IDE and write the following code.



You can download all the code in this book that isn't from the examples that come with the Arduino IDE. You can find it on the companion site at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

```
// LED Pins
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void setup() {
    // set pins to OUTPUT
    pinMode(redPin, OUTPUT);
    pinMode(greenPin, OUTPUT);
    pinMode(bluePin, OUTPUT);
}

void loop() {
    // red
    for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
        analogWrite(redPin, fadeValue);
        delay(30);
    }

    for(int fadeValue = 255; fadeValue >= 0; fadeValue -=5) {
        analogWrite(redPin, fadeValue);
        delay(30);
    }
}
```

```
// green
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
    analogWrite(greenPin, fadeValue);
    delay(30);
}

for(int fadeValue = 255; fadeValue >= 0; fadeValue -=5) {
    analogWrite(greenPin, fadeValue);
    delay(30);
}

// blue
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
    analogWrite(bluePin, fadeValue);
    delay(30);
}

for(int fadeValue = 255; fadeValue >= 0; fadeValue -=5) {
    analogWrite(bluePin, fadeValue);
    delay(30);
}

// blue + increasing red
digitalWrite(bluePin, HIGH);
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
    analogWrite(redPin, fadeValue);
    delay(30);
}
// turn blue off again
digitalWrite(bluePin, LOW);

// green + increasing red
digitalWrite(greenPin, HIGH);
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
    analogWrite(redPin, fadeValue);
    delay(30);
}
// turn green off again
digitalWrite(greenPin, LOW);
// turn off red
digitalWrite(redPin, LOW);

// blue + increasing green
digitalWrite(bluePin, HIGH);
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
    analogWrite(greenPin, fadeValue);
    delay(30);
}
// turn blue off again
digitalWrite(bluePin, LOW);
// turn off green
digitalWrite(greenPin, LOW);

// turn all on to make white
```

```
digitalWrite(redPin, HIGH);
digitalWrite(greenPin, HIGH);
digitalWrite(bluePin, HIGH);

delay(2000);

// turn all off
digitalWrite(redPin, LOW);
digitalWrite(greenPin, LOW);
digitalWrite(bluePin, LOW);
}
```

Upload the sketch to your Arduino Uno connected to the RGB LED circuit you just built. The LED should repeatedly go through a colour sequence.

# DIGGING INTO THE CODE



The code goes through a light sequence that first lights up the different colours individually and then lights up different combinations of red, green and blue to create other colours. The top of the sketch start by defining variables for each pin:

```
// LED Pins  
int redPin = 9;  
int greenPin = 10;  
int bluePin = 11;
```

The `setup()` then sets the pin modes for each of the LED pins:

```
// set pins to OUTPUT  
pinMode(redPin, OUTPUT);  
pinMode(greenPin, OUTPUT);  
pinMode(bluePin, OUTPUT);
```

The `loop()` then starts the colour sequence. First, each colour on each pin is faded on and off, one by one, using two `for` loops for each pin. The following code is just for the red pin, but it is repeated for the green and blue pins:

```
// red  
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {  
    analogWrite(redPin, fadeValue);  
    delay(30);  
}  
  
for(int fadeValue = 255; fadeValue >= 0; fadeValue -=5) {  
    analogWrite(redPin, fadeValue);  
    delay(30);  
}
```

Then two pins are lit up at a time to show what the resulting light looks like. The following code is for combining blue and red:

```
// blue + increasing red  
digitalWrite(bluePin, HIGH);  
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {  
    analogWrite(redPin, fadeValue);  
    delay(30);  
}  
// turn blue off again  
digitalWrite(bluePin, LOW);
```

After blue and red light comes blue and green light. Finally, all three colours are turned on at the same time and then turned off before the whole `loop()` starts over again:

```
// turn all on to make white  
digitalWrite(redPin, HIGH);  
digitalWrite(greenPin, HIGH);  
digitalWrite(bluePin, HIGH);  
  
delay(2000);  
  
// turn all off  
digitalWrite(redPin, LOW);  
digitalWrite(greenPin, LOW);
```

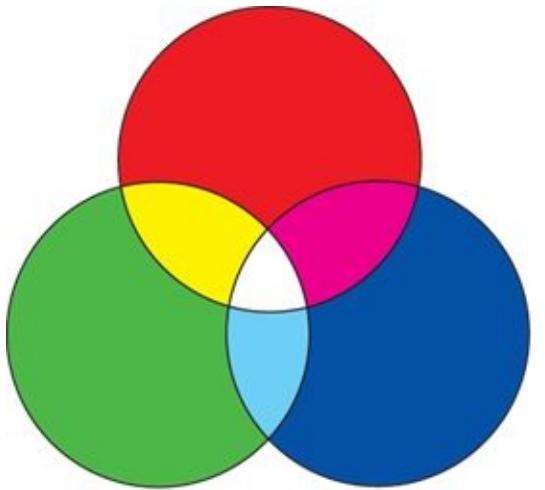
```
digitalWrite(bluePin, LOW);
```

You will see that you can still tell the difference between the red, green and blue LEDs inside the RGB LED when they are all on. It may look more white than when only one or two of the colours are on, but it won't have completely mixed. The RGB LED you are using is fairly cheap, and you can spend more money on ones that mix their colours better, but these are great to use when you are just starting out!

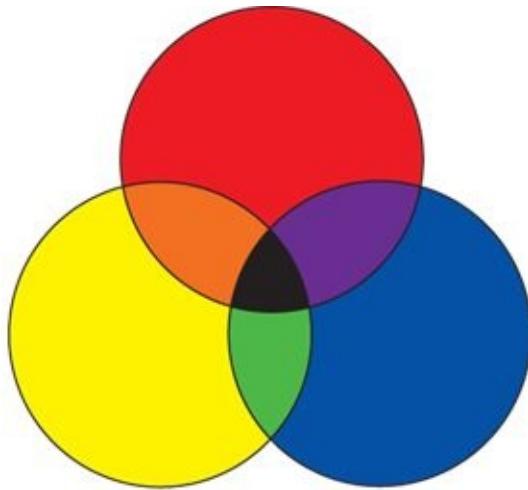


As you'll know if you've ever painted something, two colours can be mixed together to create a new colour. For example, if you mix blue and yellow paint you get green paint. Mixing all the paint colours together creates black—or at least a dark colour like black or brown. It can be tricky to produce black without being very precise with exactly what colours you mix together.

The physics behind mixing paint colours is called subtractive mixing. Mixing light is different to mixing paint, however. For example, if you mix red and green light it creates yellow; and if you mix red, green and blue light it results in white light. This process is called additive mixing. See [Figure 6-8](#) for examples of the two types of mixing. You can research additive mixing online to learn more about how to create different colours with your RGB light.



Additive Mixing  
*Light*



Subtractive Mixing  
*Paint*

[Figure 6-8](#) Mixing light versus mixing paint

# Capacitive Sensing

You probably don't realise it but you already interact with capacitive sensors every day; for example, most touchscreens on smartphones and music players use capacitive sensors. What's exciting is that you can use an Arduino to build your own capacitive sensor.

**Capacitance** is the ability to store an electrical charge. Have you ever walked across a carpeted room then touched something like a cat or a friend's arm and received an electric shock? That demonstrates that you store electrical charge. A capacitive sensor detects when something that stores charge is nearby.



**Capacitance** is the ability to store an electrical charge. Electrical components built especially to hold charge are called capacitors, but other objects—even people—also have capacitance.

The code that senses when someone is touching or near to a capacitive sensor is a bit complicated. Unless you are really interested in the details of how it works, you don't need to deal with all the ins and outs of the code—luckily, that's where libraries come in!

# Adding a Library

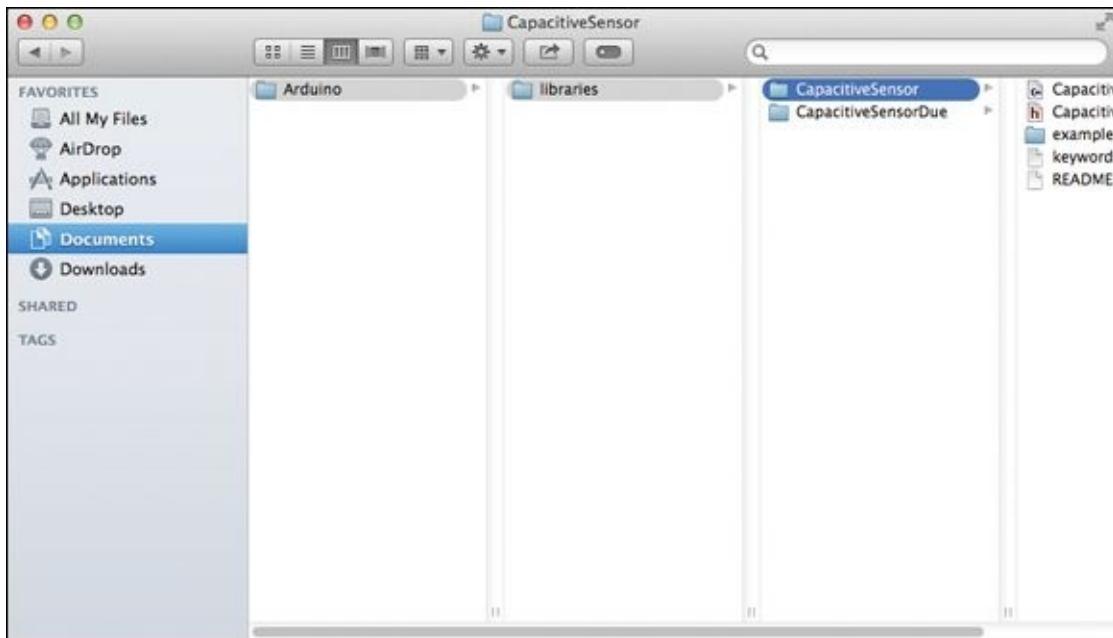
A **library** is a collection of functions that are bundled together. It's an easy way of writing bits of code that other people can use and is also an easy way of using bits of code written by others. It means you don't have to "reinvent the wheel" when other people have already done it for you. Some libraries are included with the Arduino IDE, but when you use a library that isn't included, you need to download it and put in a place where the IDE knows to look.



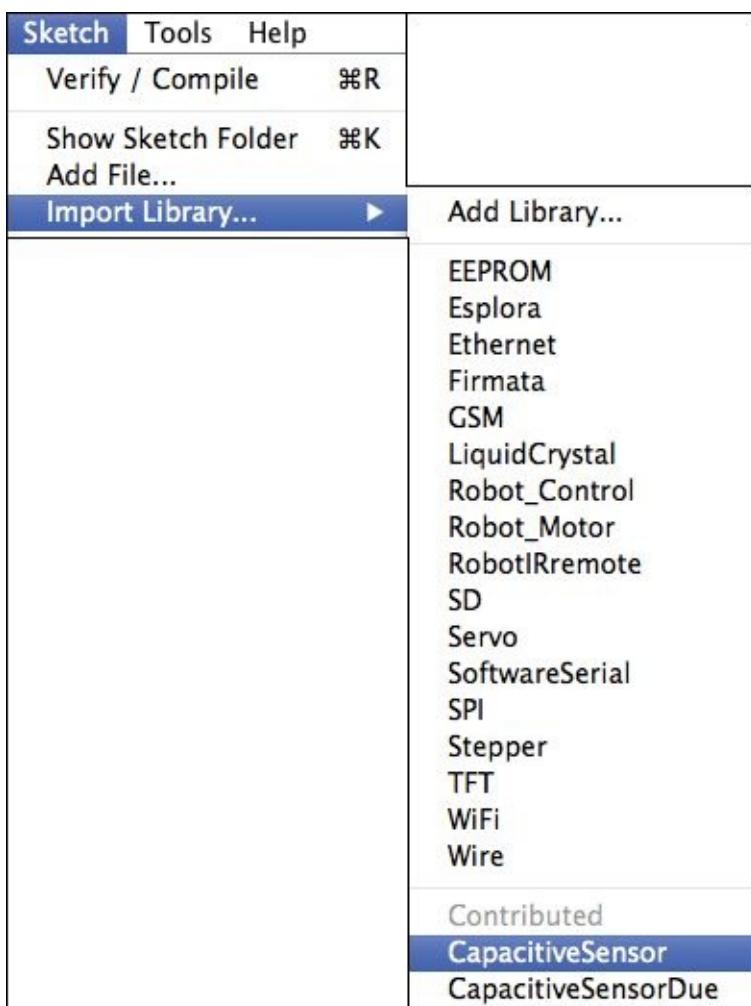
A **library** is a collection of reusable functions in code that can be imported and used in multiple sketches.

In this case, you're going to use the capacitive sensing library that you can download from <http://playground.arduino.cc/Main/CapacitiveSensor>.

1. Download and unzip the folder from <https://github.com/arduino-libraries/CapacitiveSensor/zipball/master>. The folder may be called something like `arduino-libraries-CapacitiveSensor-3e33f75`—the letters and numbers at the end of the file might differ. Inside should be a folder called `libraries` that holds two folders named `CapacitiveSensor` and `CapacitiveSensorDue`.
2. The Arduino IDE only looks for new libraries in one place: the `libraries` folder inside your sketchbook. Your sketchbook is a folder called `Arduino` inside your `Documents` or `My Documents` folder, depending on your operating system. Move the `CapacitiveSensor` and `CapacitiveSensorDue` folders and everything in them into the `libraries` folder in your sketchbook (as shown in [Figure 6-9](#)).
3. If the Arduino IDE is already open, close and restart it; otherwise, just open it.
4. To check whether the library has installed correctly, go to Sketch⇒Import Library and see if `CapacitiveSensor` and `CapacitiveSensorDue` appear in the list like in [Figure 6-10](#). Also go to File⇒Examples, where you should see `CapacitiveSensor` listed.



**Figure 6-9** Place the downloaded and unzipped folder in the libraries folder of the Arduino sketchbook.



**Figure 6-10** Check for the library and example in the menus.



If you don't see the example or library listed in your menus, try restarting the Arduino IDE. If that still doesn't make them appear, go back through the steps and make sure you have renamed the downloaded

folder correctly and have put it in the correct libraries folder.

## Wiring the Circuit

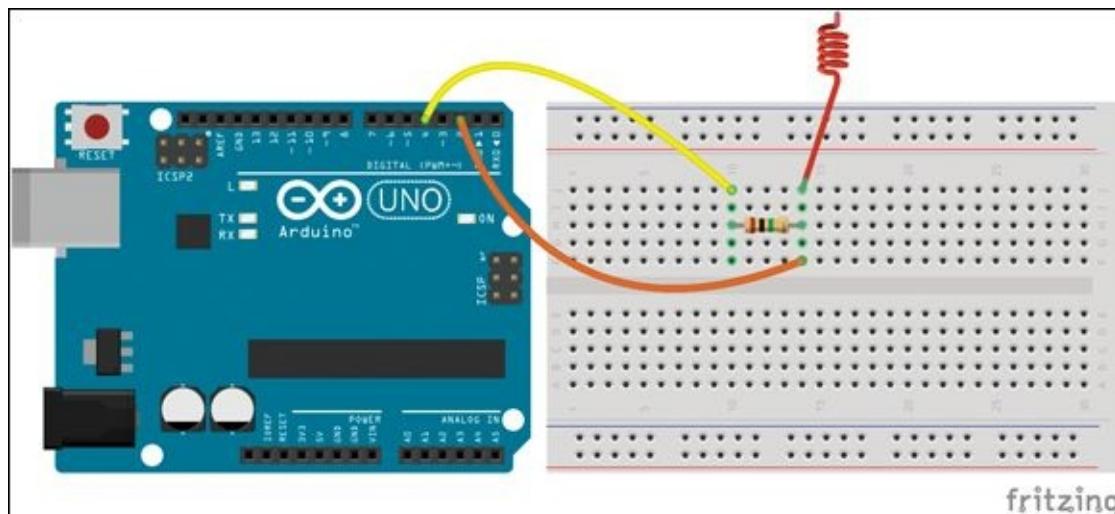
The circuit for a capacitive sensor uses two pins. One pin sends out a signal to an antenna, the second pin listens to that signal coming back in. The antenna can be anything conductive. Foil works well, but you can also always just use a wire without anything connected to it. When a capacitive object (like your finger, for example) is near the antenna, the received signal is changed according to how close the capacitive object is. The library reports back a number that corresponds to how close or far away the capacitive object is.



You may want to solder a plain paperclip to a wire (make sure the paperclip doesn't have a plastic coating; you can strip it with wire strippers if it does). Then you can use the paperclip to attach to an object like foil and use the wire to connect the foil to your breadboard.

The circuit for a capacitive sensor uses a resistor between the antenna and the sending pin. You need to use a very large resistor—at least  $10M\Omega$  (that's 10 million Ohms)! Build the circuit shown in [Figure 6-11](#):

1. Insert one leg of a  $10M\Omega$  resistor in the middle of the breadboard. Insert the other leg into any other row.
2. Use a jumper wire to connect one side of the resistor to Pin 2 on the Arduino Uno and a second jumper wire to connect the other side of the resistor to Pin 4 on the Arduino Uno.
3. Connect another jumper wire to the same short row on the breadboard that is connected to Pin 2. Don't connect the other end of the jumper wire to anything. This is now your antenna for your capacitive sensor.



[Figure 6-11](#) Capacitive sensing circuit

# Writing the Code

When you are using a library, the first step is to tell the Arduino IDE in your code what library you want to use. You do this by using `#include` and then the name of the file that describes the library, like this:

```
#include <CapacitiveSensor.h>
```

The file extension is `.h`. The `<` and `>` mean that the library is located where the Arduino IDE would expect it to be: the **libraries** folder in the sketchbook.

The library uses an object called **CapacitiveSensor**. This object handles all the nitty-gritty details of interacting with the sensor. You need to create a new variable that is the type **CapacitiveSensor**. It only takes two arguments: the pin from where the signal is sent and the pin the sensor is connected to:

```
CapacitiveSensor handSensor = CapacitiveSensor(outputPin, ↵ sensorPin);
```

The **capacitiveSensor()** function is called to read from the sensor. Call the function type **handSensor.capacitiveSensor()**, because the function belongs to the **CapacitiveSensor** variable **handSensor**:

```
long sensorValue = handSensor.capacitiveSensor(30);
```

The returned value is stored in a **long** data type. A **long** is like an **int**, but it can hold much smaller and bigger numbers. The returned number from the capacitive sensor might be a larger number than an **int** can hold, so it's best to use the larger data type.



A **long** is a data type that can hold whole integer numbers from -2,147,483,648 to 2,147,483,647.

When you put all the code together, it looks like the following sketch:

```
#include <CapacitiveSensor.h>

// capSense pins
int sensorPin = 2;
int outputPin = 4;

CapacitiveSensor handSensor = CapacitiveSensor(outputPin, ↵ sensorPin);

void setup() {
    // begin serial communication
    Serial.begin(9600);
}

void loop() {
    // read in the value from the sensor
    long sensorValue = handSensor.capacitiveSensor(30);
    // print the value
    Serial.println(sensorValue);

    // wait for a short while before continuing
}
```

```
    delay(10);  
}
```

Upload the sketch to your Arduino Uno and open the Serial Monitor in the Arduino IDE on your computer. Touch the end of the jumper wire acting as your antenna and watch what happens to values being printed.

# DIGGING INTO THE CODE



There isn't a lot to the sketch—it's quite short, but let's go over it in more detail. At the top are the variables for the sketch. There are two pins that the capacitive sensor needs: the pin connected to the resistor and the pin connected to the resistor and antenna:

```
// capSense pins  
int sensorPin = 2;  
int outputPin = 4;
```

Inside the `setup()`, the only thing that happens is that serial communication starts:

```
// begin serial communication  
Serial.begin(9600);
```

The `loop()` is also brief. It reads in the current value from the capacitive sensor and saves it in the variable `sensorValue`. That variable is then printed to the Serial Monitor. A `delay()` is used to pause the loop() before the whole process repeats again.

```
// read in the value from the sensor  
long sensorValue = handSensor.capacitiveSensor(30);  
// print the value  
Serial.println(sensorValue);  
  
// wait for a short while before continuing  
delay(10);
```



What happens when you don't touch the exposed metal pin of the jumper wire antenna and you only touch the plastic coating on the wire?



Try adding more  $10M\Omega$  resistors to increase the sensitivity of your sensor. That means the sensor will be able to detect your hand from farther away. Add them in series—end to end. When you combine resistors in series, the total resistance is the sum of the individual resistors. This is useful as it's practically impossible to buy resistors with values greater than  $10M\Omega$ , but you can put multiple resistors in series and to the sensor circuit they look like one big resistor.

# Building a Crystal Ball

Capacitive touch sensors can appear to be quite magical, even though you know it's just electrical engineering. So let's exaggerate the experience and create a crystal ball that mysteriously lights up when you wave your hand near it, like in [Figure 6-12](#).

You can make your own "crystal" with papier maché tissue paper over a balloon. It's a great way to make a semi-transparent sphere that will light up nicely when your RGB LED is inside.

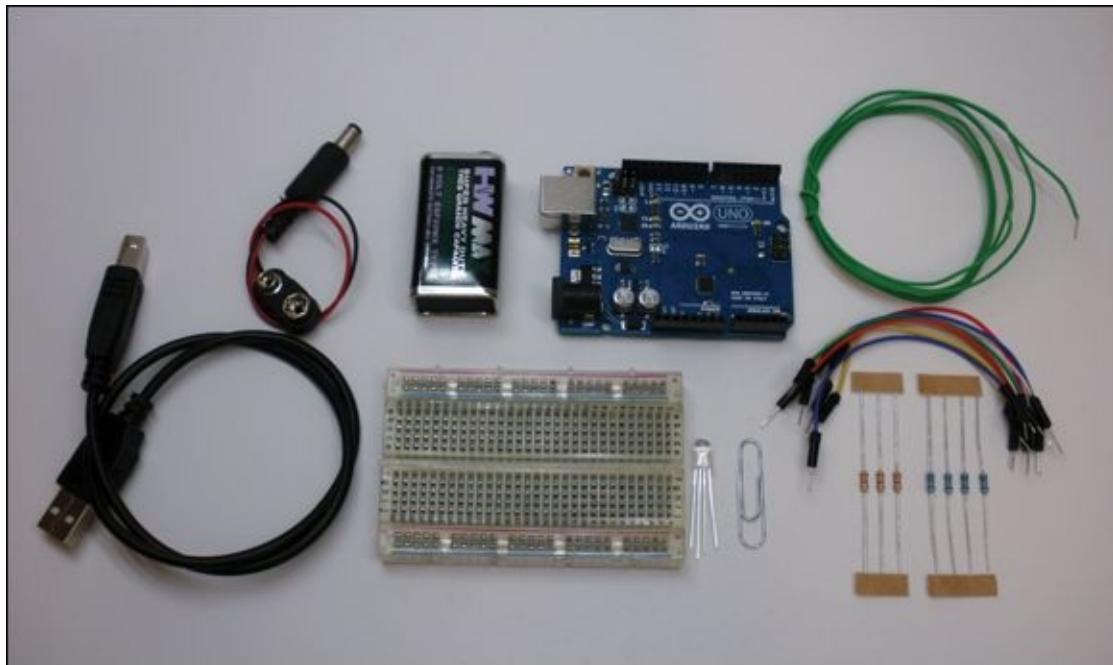


[Figure 6-12](#) A touch-sensitive crystal ball

## What You Need

You will need the following items to make your crystal ball. The electronic components are pictured in [Figure 6-13](#).

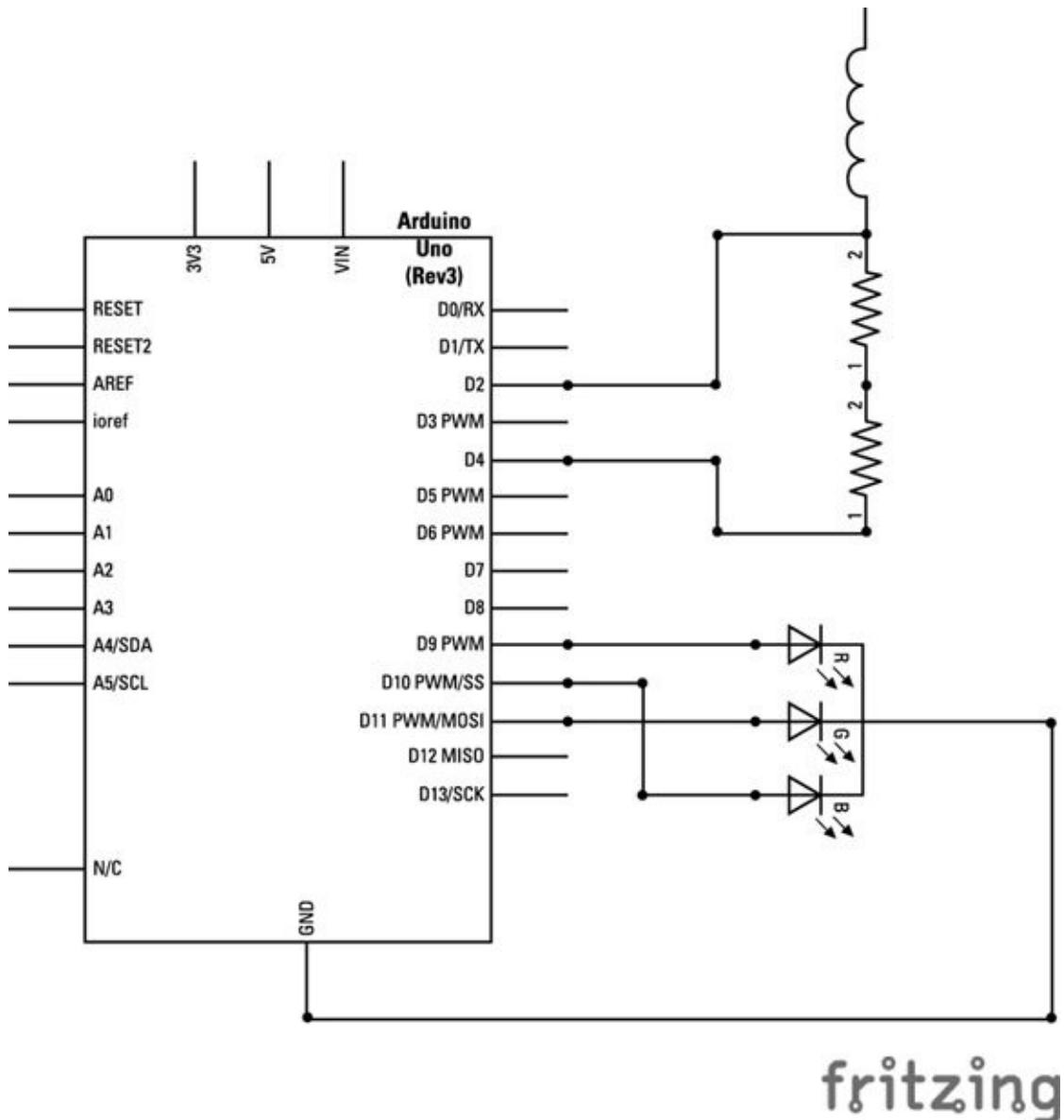
- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- 8 jumper wires
- Solid core wire
- 3  $220\Omega$  resistors
- 4  $10M\Omega$  resistors
- An RGB common cathode LED
- A paperclip
- 9V battery
- 9V battery-to-DC barrel jack connector
- 3 sheets white tissue paper
- A balloon
- Thin cardboard or thick paper
- Aluminium foil
- A soldering iron
- Solder
- Scissors or utility knife
- PVA glue
- Paintbrush



**Figure 6-13** The electronic components you need to make the crystal ball

## Understanding the Circuit

The circuit for the crystal ball is a combination of the RGB LED and capacitive-sensing circuits you made earlier in this adventure. It might be easier to understand the circuit by looking at a circuit schematic, instead of looking at a breadboard, so take a look at [Figure 6-14](#).

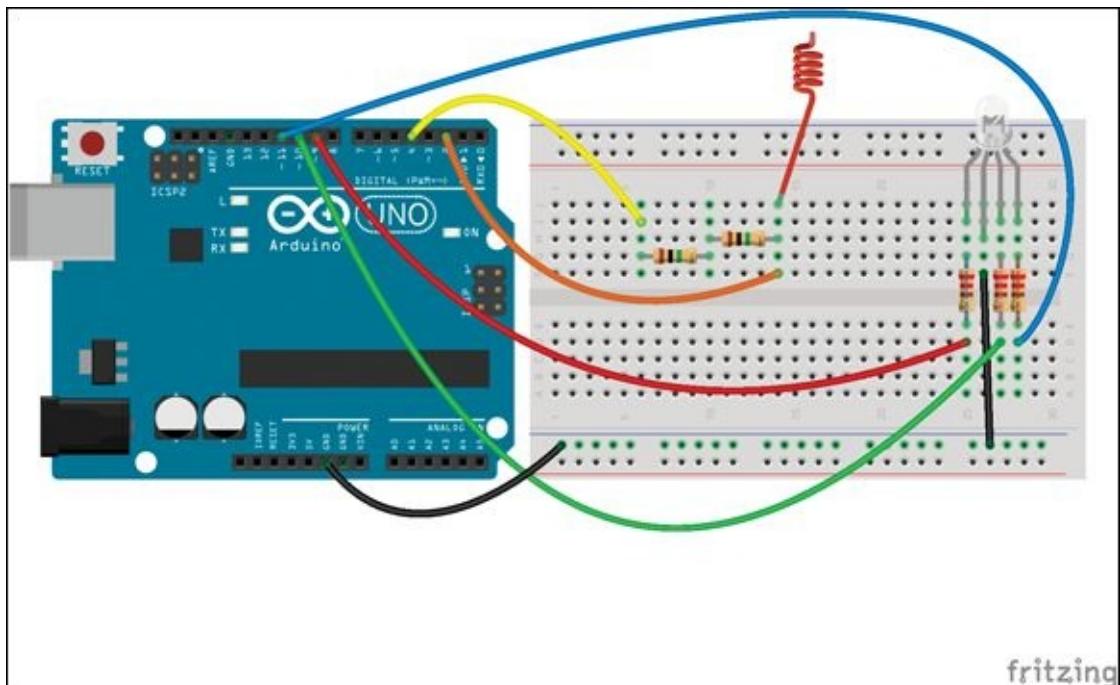


[Figure 6-14](#) Circuit schematic for the crystal ball

## Prototyping on a Breadboard

If you've worked through the earlier adventures in this book you're now an experienced engineer, so you know how important it is to try things out on a breadboard before you start soldering everything together. However, I still want to remind you to do it, just in case you feel like rushing through this step. Build the circuit shown in [Figure 6-15](#):

1. Use a jumper wire to connect a GND pin on the Arduino Uno to a long row along the bottom of the breadboard. If the breadboard is labelled with a black or blue line or -, connect the pin to that row.
2. Place the RGB LED in the breadboard so that each leg is in its own short row. Use a jumper wire to connect the longest leg of the LED to the long row connected to the GND pin.
3. Insert one leg of a  $220\Omega$  resistor into the same row as each of the colour legs of the LED. Bend the resistor over the gap in the middle of the board, and insert the leg of each LED into its own short row.
4. Find the red leg and connect the resistor now connected to it to Pin 9 with a jumper wire; repeat to connect the resistor connected to the green leg to Pin 10; and repeat a second time to connect the resistor connected to the blue leg to Pin 11.
5. At the other end of the breadboard, insert the legs of a  $10M\Omega$  resistor into two different rows in the middle of the breadboard.
6. Insert the leg of a second  $10M\Omega$  resistor into the same row as one of the legs of the first resistor. Insert the last leg into an empty row.
7. Connect a jumper wire from Pin 2 to one of the resistors (just don't connect it to the row where the resistors are connected to each other). Connect another jumper wire into this same row. This acts as your antenna for your prototype circuit.
8. Connect a jumper wire from Pin 4 to the remaining resistor leg that isn't connected to anything else.



**Figure 6-15** Breadboard prototype circuit

After you have built your base for your crystal ball from paper and foil, you use the breadboarded circuit to determine how many resistors you would like for your capacitive touch sensor. Somewhere between one and four should be right. You can add more resistors to your two resistors on the breadboard end to end the way the two resistors are already connected.

## Writing the Code

The first few lines of the sketch are a mash-up of the RGB LED sketch and capacitive sensing sketch from earlier in this adventure. In this code, the variables for the pins and the sensor are set up. You will be building your base in the next section. It will be covered in foil and will be the antenna for your circuit, but first create a new sketch in the Arduino IDE with the following code:

```
#include <CapacitiveSensor.h>

// LED pins
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

// capSense pins
int sensorPin = 2;
int outputPin = 4;

// touch threshold
int threshold = 1000;

CapacitiveSensor handSensor = CapacitiveSensor(outputPin, @@@
sensorPin);

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  long sensorValue = handSensor.capacitiveSensor(30);
  Serial.println(sensorValue);

  // if above the threshold
  if(sensorValue > threshold) {

    // calculate color value based on sensor reading
    int redValue = map(sensorValue, threshold, 90000, 0, 255);
    int greenValue = map(sensorValue, threshold, 20000, 0, 255);
    int blueValue = map(sensorValue, threshold, 30000, 0, 255);

    // turn on led
    analogWrite(redPin, redValue);
    analogWrite(greenPin, greenValue);
    analogWrite(bluePin, blueValue);
  }
  else{
    // otherwise turn off led
    digitalWrite(redPin, LOW);
    digitalWrite(greenPin, LOW);
  }
}
```

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

Upload the sketch to your Arduino Uno, and touch the jumper wire acting as the antenna. You should see the light turn on and change colours. Open the Serial Monitor in the Arduino IDE to monitor the values being read by the touch sensor.

# DIGGING INTO THE CODE



The main variable you use to adjust your interaction with your crystal ball is `threshold`. This variable plays an important role in the `loop()`. It determines how sensitive your crystal ball is by keeping the LED turned off until a big-enough value is read from the sensor:

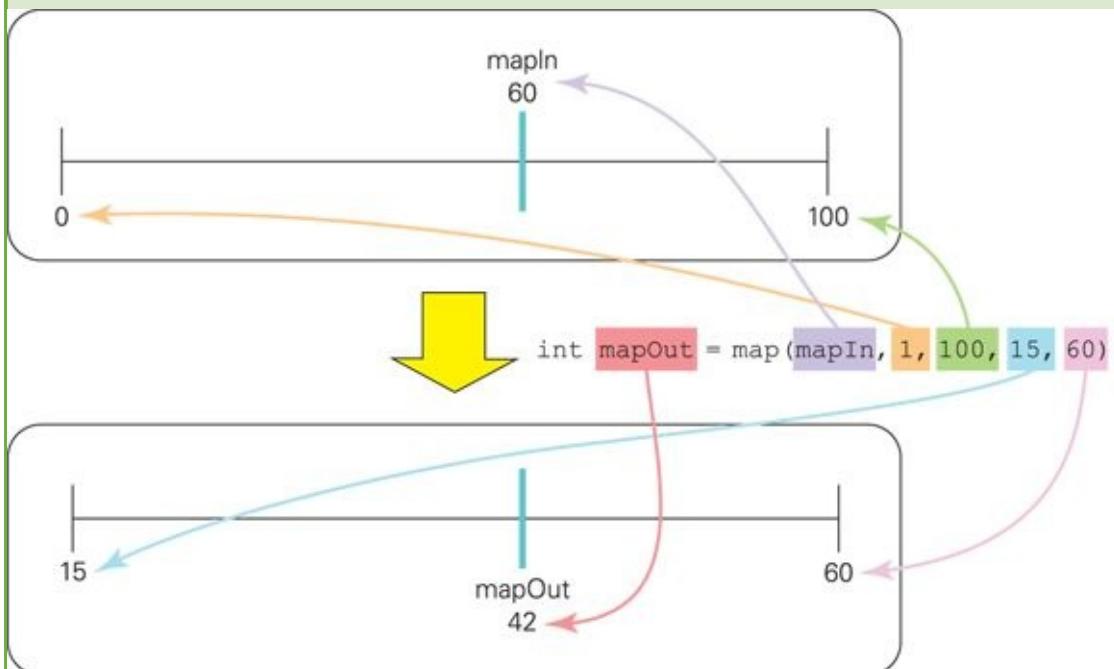
```
// touch threshold  
int threshold = 1000;
```

Adjust this number in the Serial Monitor. You may have to adjust this number multiple times as your antenna changes from a prototype jumper wire to your crystal ball.

In the `loop()`, `threshold` determines when the lights turn on:

```
// if above the threshold  
if(sensorValue > threshold) {
```

Within that `if` statement, there are three lines of code using a new function: `map()`. This function takes a number within a range of values and translates it to a new range of values. This is useful because the range of numbers that can be read in from the sensor can get very large, but `analogWrite()` can't handle anything larger than 255. The `map()` function helps by scaling the value from the sensor to a new number that's within the acceptable range. [Figure 6-16](#) illustrates how this works.



[Figure 6-16](#) Mapping a value to a new range

The first argument of `map()` is the number to be mapped. The second and third arguments are the current lowest and highest values those numbers can be. The fourth and fifth numbers are the new lowest and highest values the mapped numbers can be:

```
int redValue = map(sensorValue, threshold, 90000, 0, 255);  
int greenValue = map(sensorValue, threshold, 20000, 0, 255);  
int blueValue = map(sensorValue, threshold, 30000, 0, 255);
```

In the sketch, the maximum of the starting range is different for each colour. This is to make each of the colours respond slightly differently to the same sensor value. For example, a value of 3000 creates a `redValue` of 6, `greenValue` of 28 and a `blueValue` of 18. Try out some different values in the three `map()` functions to get the colours you like best.

## Making the Crystal Ball

Now to make the crystal ball itself. You are going to use papier maché to create a thin sphere that will let through light. The base on which the ball sits is thin cardboard or thick paper covered in aluminium foil. The base serves as the capacitive sensor. The electronics fit nicely inside the base and, if you use a battery to power the Arduino, it will create a perfect magical illusion.

## Making a Papier Maché Ball

The “ball” of the crystal will be papier maché that gets its shape from a balloon. You use tissue paper instead of a thicker paper so that the light from the RGB LED is still visible.

1. Blow up a balloon to the size that you want your crystal ball. Remember that if you want the Arduino board to fit inside the base, your balloon needs to be big enough to rest in a base that can hold the board.
2. Water down about a tablespoon or 25 ml of white PVA glue with a half to full tablespoon or 15–25 ml of water.
3. Cut up three sheets of white tissue paper into approximately one-inch by one-inch squares.
4. Use a paint brush to apply the glue mixture to a small section of the balloon starting at the top opposite of the knot.
5. Place a tissue paper square on the glued area and brush more glue on top of the paper.
6. Repeat steps 4 and 5 working towards the knot of the balloon, taking care to overlap the squares only a tiny bit, until the top two-thirds of the balloon is covered.
7. Leave the balloon to dry for at least half a day or overnight.
8. Repeat with another layer of tissue paper. You may need to apply up to three layers of tissue. You want the dried papier maché to feel firm enough to support its shape after you have popped and removed the balloon.
9. When the papier maché is dry and feels firm enough to hold its shape, pop the balloon. Carefully peel away the balloon from the inside of the papier maché. If needed, trim the paper to create a neat edge to the “crystal”. You will end up with an object like the one shown in [Figure 6-17](#).
10. Cut a strip of thicker paper or thin cardboard about 2” wide. Wrap it in aluminium foil.
11. Curl the foil-covered paper into a ring just big enough to rest the ball on and will also fit around the Arduino Uno (see [Figure 6-18](#)). Secure the ring with a staple or tape.
12. Attach the ball to the ring with tape or glue.



**Figure 6-17** Papier maché crystal ball



**Figure 6-18** Aluminium foil–covered base

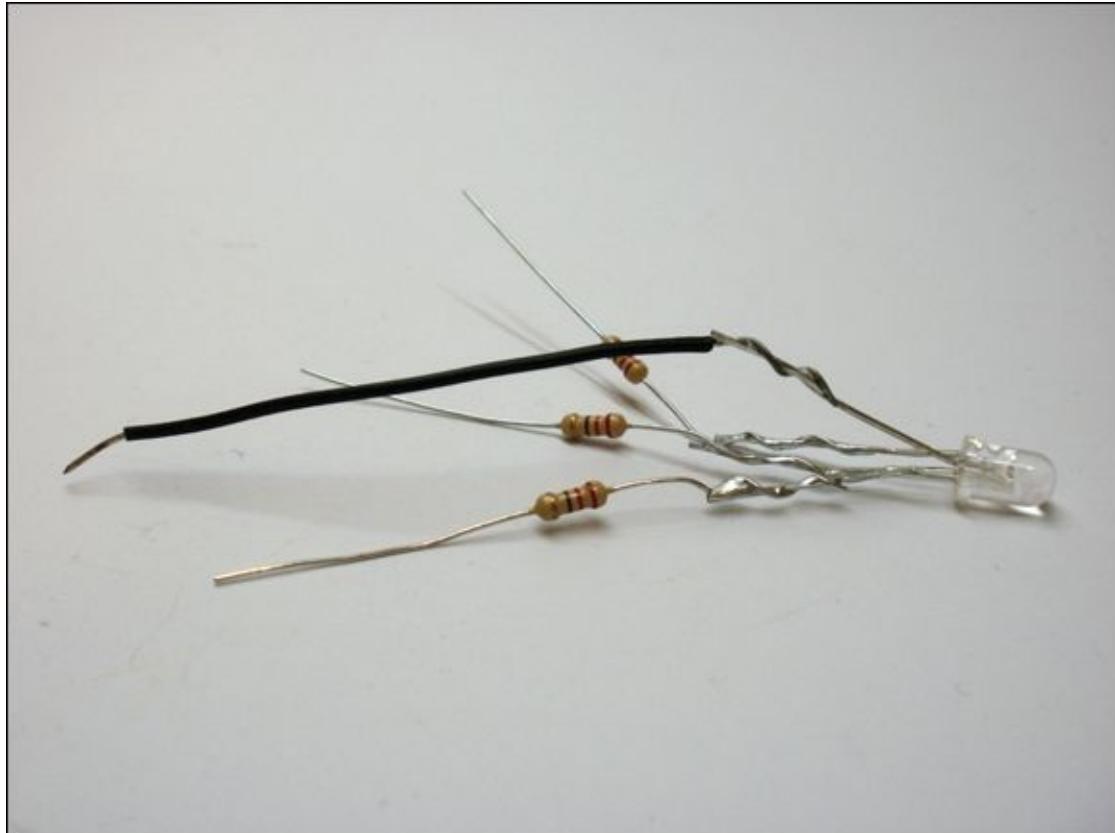
## Soldering the Electronics

There aren't many parts to this circuit, but be sure to prototype it on a breadboard first or you won't know how many resistors to use for your sensor.



Soldering can be dangerous as the iron gets very hot. Only solder when an adult is nearby to help.

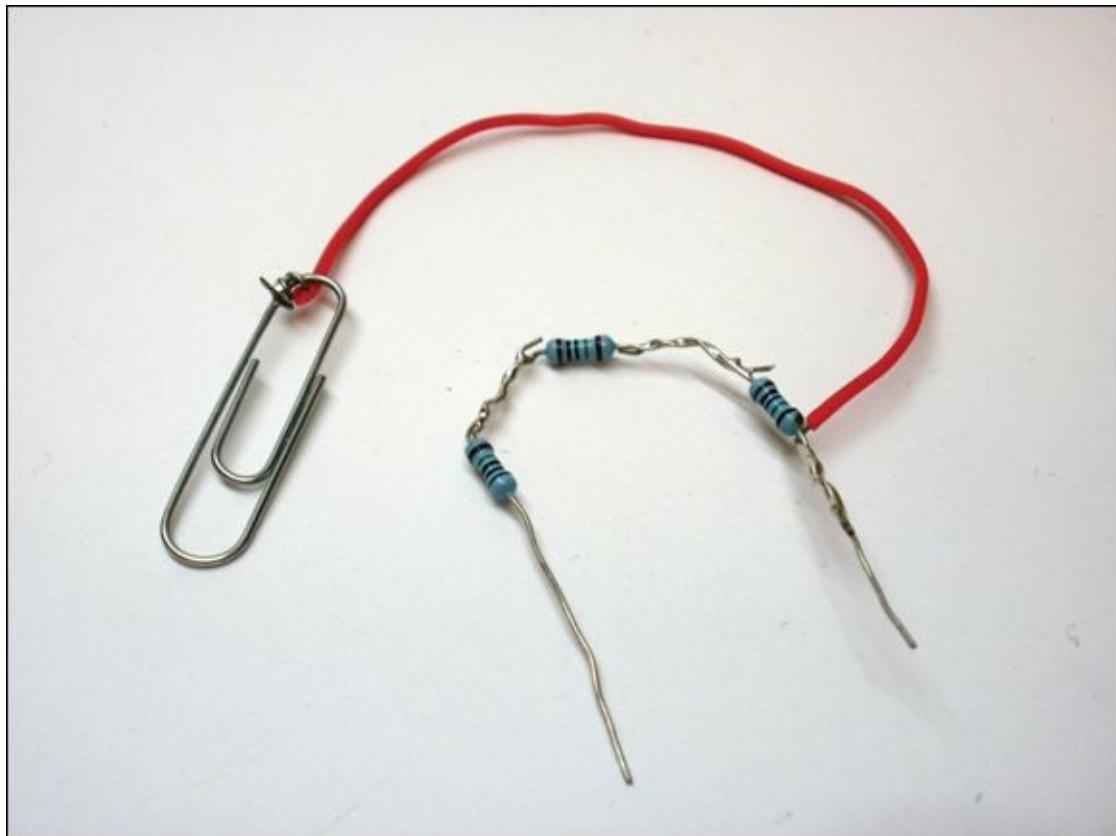
Solder a  $220\Omega$  resistor onto each of the shorter three legs of the RGB LED. Cut a piece of solid-core wire the same length as a resistor and solder it to the cathode of the RGB LED (see [Figure 6-19](#)).



**Figure 6-19** Soldered LED circuit

Solder together the number of 10M-ohm resistors you need for your circuit.

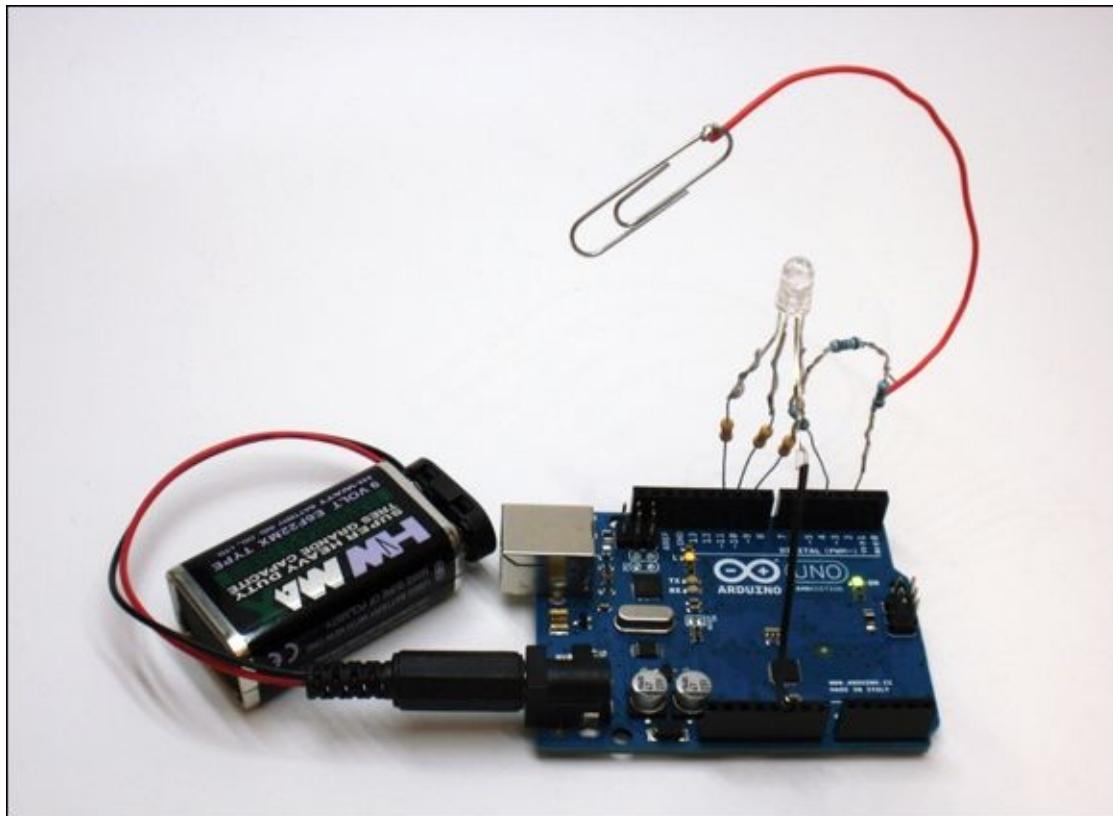
Cut a piece of solid-core wire about 3" long. Solder one end to a paperclip and the other end to the last 10M-ohm resistor in your chain of resistors (see [Figure 6-20](#)).



**Figure 6-20** Soldered sensor circuit

## Connecting the Electronics

Insert the end of the resistors not soldered to the wire into Pin 4 and the wire connected to the paperclip into Pin 2. Insert the resistors connected to the red, green and blue parts of the RGB LED into Pins 9, 10 and 11 (see [Figure 6-21](#)). Attach the paperclip onto the aluminium foil base of the crystal ball.



**Figure 6-21** Completed crystal ball circuit

Power the Arduino board using a 9V battery and DC barrel jack connector. Try out your crystal ball by waving your hands over it! The crystal ball should only light up when your hand is near and change colours when your hand gets closer to the aluminium base. Remember, you may need to adjust the threshold value until you get the interaction that you want.

# Further Adventures with Libraries

You have learned how to strike out on your own and start using other libraries with your Arduino. A whole world of possibilities has opened up! Explore the Arduino Playground (<http://playground.arduino.cc/>) to see what other libraries are available.

If you'd like to read more on how to use capacitive sensing, visit the Arduino Playground documentation at <http://playground.arduino.cc/Main/CapacitiveSensor>.

## Arduino Command Quick Reference Table

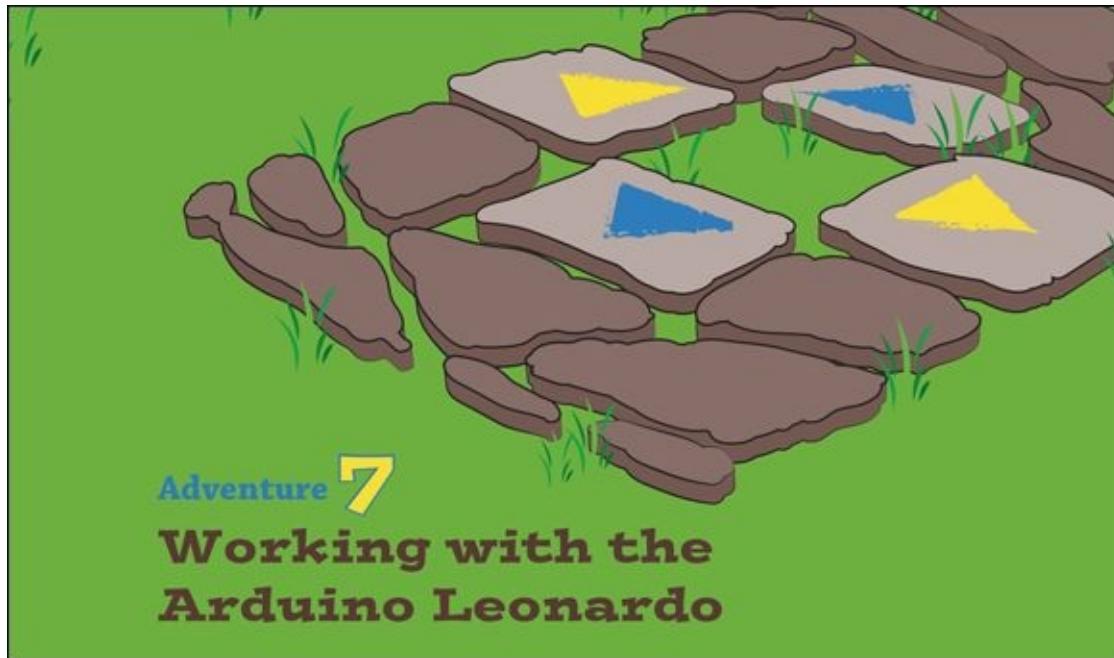
Command	Description
analogWrite()	Outputs a voltage between 0 and 5V on a specified pin. A value of 0 outputs 0V and 255 outputs 5V. See also <a href="http://arduino.cc/en/Reference/AnalogWrite">http://arduino.cc/en/Reference/AnalogWrite</a> .
CapacitiveSensor	Library for creating capacitive sensors. See also <a href="http://playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense">http://playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense</a> .
long	A long is a data type that can hold whole integer numbers from -2,147,483,648 to 2,147,483,647. See also <a href="http://arduino.cc/en/Reference/Long">http://arduino.cc/en/Reference/Long</a> .
map()	Takes a value within an initial range and maps it to a new range. See also <a href="http://arduino.cc/en/Reference/Map">http://arduino.cc/en/Reference/Map</a> .



**Achievement Unlocked:** You have expanded your knowledge with libraries!

## In the Next Adventure

In the next adventure, you start exploring other types of Arduino boards to create a custom computer game controller!

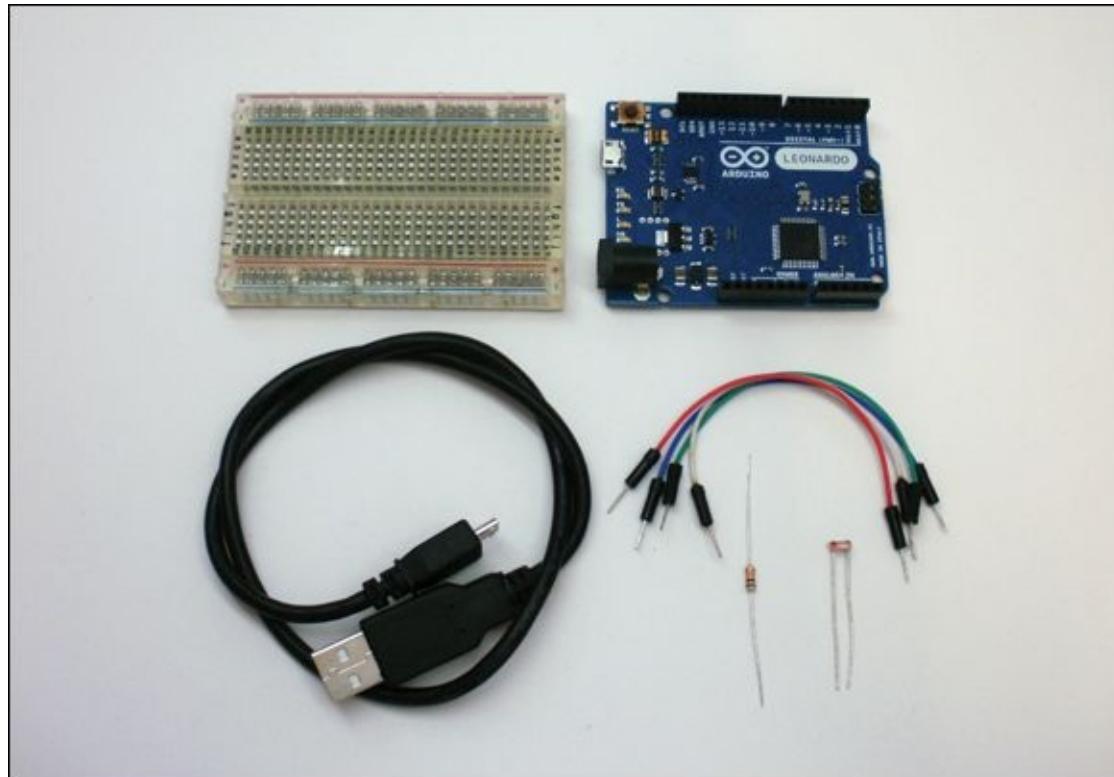


**WHEN YOU SELECT** your Arduino Uno board from Tools⇒Board in the Arduino IDE, you might notice that that it is just one item in a very long list of board names. All the others sound just as exciting. So, what makes them different from your Uno? There's not room in this book to describe all of them but in this adventure you will get to know one of them—the Arduino Leonardo. The Leonardo has a special skill that the Uno doesn't have: it can make a computer believe that the Arduino board is a USB keyboard or mouse.

For this adventure, you also use a sensor that detects whether it's in bright light or shadow. You use this to create a USB game controller. The sensor means you will be able to wave your hands in the air above the controller to play computer games!

# What You Need

You need the following items for the first part of the adventure (see [Figure 7-1](#)):



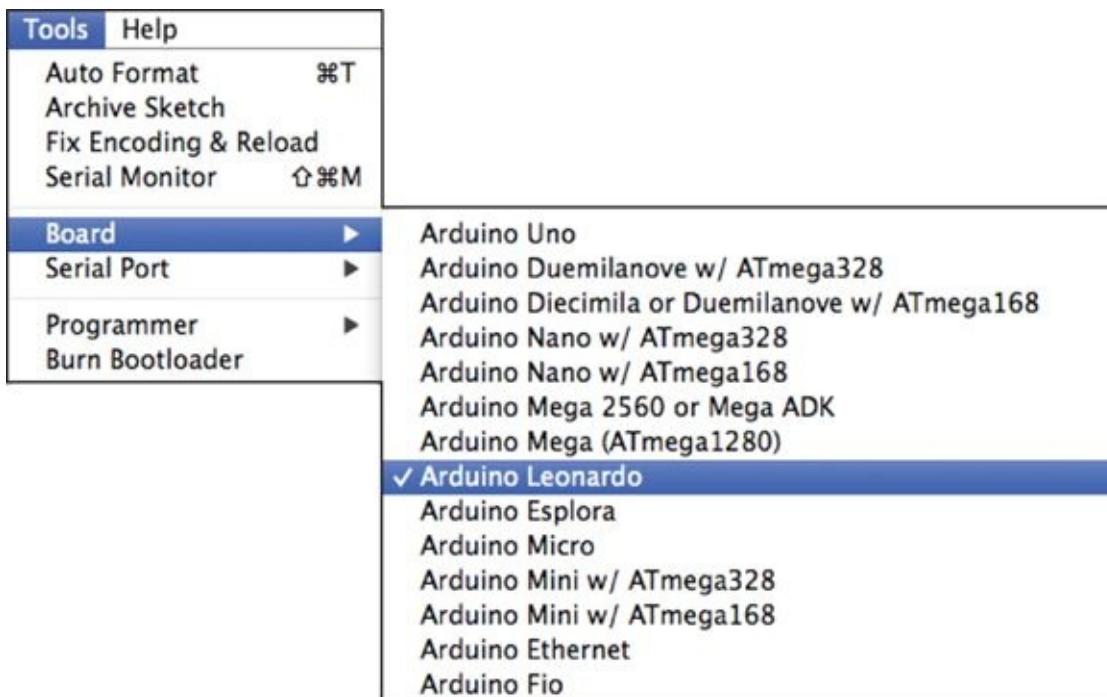
[Figure 7-1](#) The electronic components you need for the first part of this adventure

- A computer
- An Arduino Leonardo
- A USB Micro cable
- A breadboard
- 4 jumper wires
- A light-dependent resistor
- A  $10k\Omega$  resistor

# Introducing the Arduino Leonardo

One of the many great things about the Arduino platform is that you can choose a different board for your project without having to change the code. You can use almost all the code you have learned to write for the Arduino Uno with the Arduino Leonardo as well. You even use the same Arduino integrated development environment (IDE) to upload the code.

The main thing you have to do differently is that when you select which board you are using from Tools⇒Board in the IDE, you need to select Arduino Leonardo, as shown in [Figure 7-2](#).

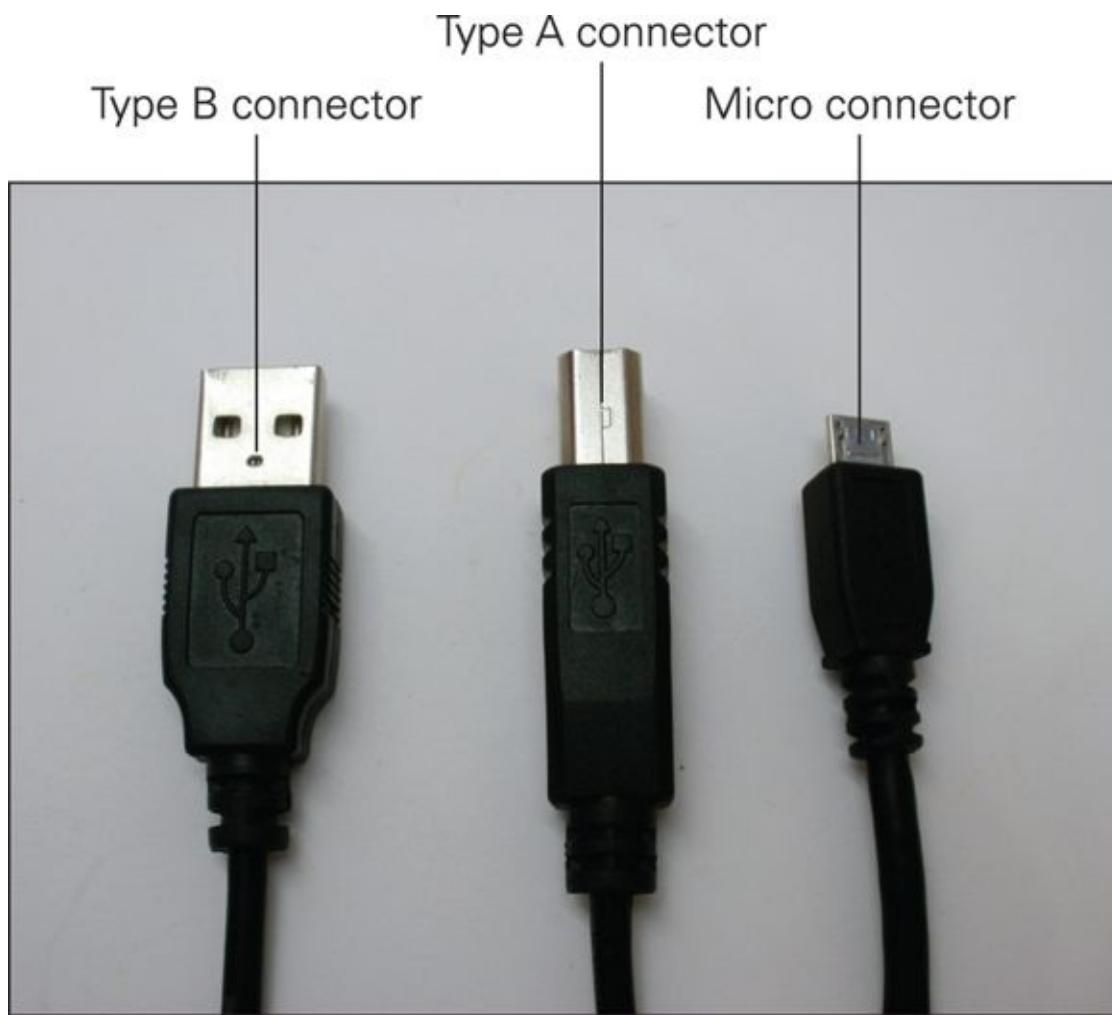


[Figure 7-2](#) Selecting the Arduino Leonardo from Tools⇒Board in the Arduino IDE

## Connecting Your Leonardo for the First Time

The first thing you might notice when connecting your Arduino Leonardo to your computer for the first time is that it doesn't have the same kind of USB connector as your Arduino Uno. You can't use the same USB cable for both boards.

The connectors on USB cables come in different sizes and shapes, as shown in [Figure 7-3](#). What you might think of as a "normal" USB cable has a Type A connector at one end, which goes into your computer, and a Type B connector at the other end, which goes into your Arduino Uno. The Arduino Leonardo uses a USB Micro connector instead of a Type B connector. This doesn't do anything different from a normal USB cable; it still lets the Leonardo get power from a computer and can be used to talk with the computer. The only difference is that the connector has a different shape. That means you have to keep track of two different kinds of USB cable!



[Figure 7-3](#) USB connectors

One of the features the Leonardo has that the Uno doesn't is that, to the computer, the board can seem like a USB keyboard. Because of this, when you connect your Leonardo, your computer might pop open a window that says the computer has detected a new keyboard. Just close the window. You don't need to click Continue or set up a new device; you program your Arduino Leonardo using the Arduino IDE just like you have been doing with your Arduino Uno.



If the computer you are using to program your Arduino Leonardo runs on Windows 7, you may need to install some additional drivers. Connect your Leonardo and wait for the automatic driver installation process. If nothing happens after a few minutes, go to <http://arduino.cc/en/Guide/ArduinoLeonardoMicro?from=Guide.ArduinoLeonardo> for instructions about how to install the drivers yourself.

# DIGGING INTO THE CODE



Most of the time, the code that you write for your Uno works the same way on the Leonardo, but it's good to know what might be a little different.

With your Arduino Uno, every time you open the Serial Monitor in the Arduino IDE, the Uno resets. The `setup()` function then runs once before going into `loop()`. The same isn't true for your Leonardo. The Leonardo doesn't reset when the Serial Monitor is opened. That means that if you want to print something to the Serial Monitor from the `setup()` function, it doesn't appear; by the time the Serial Monitor opens, the `print` statement from the `setup()` function would already have passed and you'd only see messages from the loop.

In the Arduino IDE, create a new sketch with the following code. Upload it to your Leonardo and then open the Serial Monitor:

```
void setup() {  
    Serial.begin(9600);  
    Serial.println("Hello from the setup!");  
  
}  
  
void loop() {  
    Serial.println("And hello from the loop!");  
    delay(1000);  
}
```

You only see the message "And hello from the loop!" printed over and over again.

Now try pressing the reset button on the Leonardo board (it's next to the USB Micro connector). The messages stop printing to the Serial Monitor completely! This is because the connection between the Serial Monitor and the Leonardo was broken when you pressed reset. You need to close and reopen Serial Monitor to let the Arduino Leonardo know that the Serial Monitor is there and waiting.

So what does that all mean? Does that mean you can't ever print messages from the `setup()` function of an Arduino Leonardo to read in the Serial Monitor? That would be frustrating! It can be really useful to print messages, as it helps you to know what is going on inside the board and fix any problems with your code. Well, the good thing is there is a way around this.

You can tell the Leonardo to wait and not do anything until a serial connection to something like Serial Monitor is opened.

Try uploading the following code to your Leonardo, and then open the Serial Monitor:

```
void setup() {  
    Serial.begin(9600);  
    while(!Serial); // sit and wait for a serial connection  
    Serial.println("Hello from the setup!");  
  
}  
  
void loop() {  
    Serial.println("And hello from the loop!");  
    delay(1000);  
}
```

You should now see the "Hello from the setup!" message before the "And hello from the loop!" messages. If you try pressing reset on the board, you see that you still need to close and reopen the Serial Monitor to see the printing messages again.

# Acting Like a Keyboard

You can send messages to the computer from your Arduino Uno using functions like `Serial.println()`, but you need a special program like the Serial Monitor in the Arduino IDE to be able to read those messages. The Arduino Leonardo can send messages that don't need a special program for the computer to understand. It can send messages that look like keys being pressed on a keyboard. Any program that responds to key presses (like a word processing program or a computer game) can understand those messages.

Start by creating a new sketch in the Arduino IDE. Write out the following code to create an empty `setup()` and `loop()`:

```
void setup() {  
}  
  
void loop() {  
}
```

Before you start turning your Arduino Leonardo into a keyboard, it's very important that you give it an off switch. Your Leonardo overrides your computer's keyboard the same way as plugging a USB keyboard into your computer does. If your Leonardo is constantly typing messages at your computer, it can be difficult to tell your computer to do anything else—including uploading a new sketch to your Leonardo.

To prevent this from happening, in the `loop()` you are going to first check to see if an input pin is set to 0. If it is, then don't print any keyboard messages. You don't even need to wire up a switch like you did in [Adventure 3](#); you can just use a jumper wire! Here's how.

First, at the top of your sketch, before the `setup()`, create a new variable to store your switch pin number:

```
int switchPin = 4;
```

Inside the `{` and `}` of `setup()`, set that pin to be an input with the pull-up resistor turned on:

```
pinMode(switchPin, INPUT_PULLUP);
```

Then in `loop()`, add the following code to check if `switchPin` has been connected to ground. If it has, continue on to print the message; otherwise do nothing:

```
// read the pushbutton:  
int switchState = digitalRead(switchPin);  
  
// if the switch is open (not connected to ground),  
if (switchState == LOW) {  
    // add keyboard code here  
}  
delay(500);
```

The purpose of `delay()` at the end is to slow down how often the loop repeats. That way,

when messages are being sent, they aren't sent too fast.

Now if you connect a wire between `switchPin` (Pin 4) and GND, the code that you put inside the `if` statement is run. If you don't connect anything to the `switchPin`, the pull-up resistor causes the value read from `switchPin` to be 1 and the code inside the `if` statement is skipped.

Time to start adding some keyboard messages!

The `Keyboard` functions look a lot like the `Serial` functions. In the `setup()` function you need to start the Leonardo's keyboard messaging by calling:

```
Keyboard.begin();
```

You can then write messages using `Keyboard`:

```
Keyboard.println("This is your Leonardo acting like a keyboard.");
```

When the code is all put together, you get the following sketch:

```
int switchPin = 4;           // input pin for switch

void setup() {
  // make the switchPin and input
  // with an internal pull-up resistor
  pinMode(switchPin, INPUT_PULLUP);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // read the pushbutton:
  int switchState = digitalRead(switchPin);

  // if the switch is open (not connected to ground),
  if (switchState == LOW) {
    Keyboard.println("This is your Leonardo acting like a keyboard.");
  }
  delay(500);
}
```

Upload the code to your Leonardo and then open any word processing program. Use the jumper wire to connect your `switchPin` to GND. You should see your Leonardo typing out messages like those in [Figure 7-4!](#)



## Figure 7-4 The Leonardo typing in a word processing program



Don't create a runaway keyboard! If you find you are having problems uploading a new sketch to your Leonardo, there is something you can try. You can manually reset the Arduino Leonardo and tell it to listen for a sketch to be uploaded. Hold down the button on the Leonardo and then click on the Upload button in the Arduino IDE. Wait until the IDE says "Uploading..." and then release the button. Your sketch should then finish uploading onto your board.



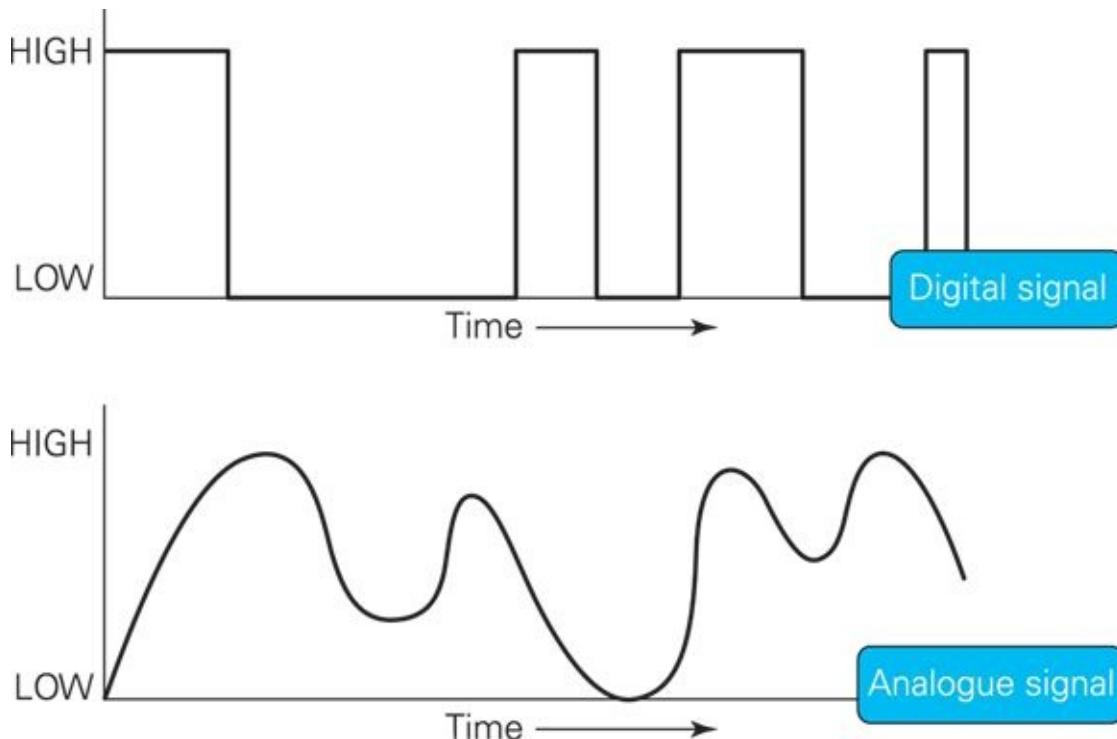
Did you know you can come up with a new idea using an Arduino, then make and sell your idea as a product in a store? The MaKey MaKey is one example of an Arduino project you can buy in a store. The MaKey MaKey is based on the Leonardo Arduino—its name comes from "Make anything into a keyboard". You now know a lot about the code behind how it works. If you'd like to learn more about the MaKey MaKey, go to <http://makeymakey.com>, where you can download the Arduino code used to program the MaKey MaKey and read how it works. You can even turn your Arduino Leonardo into a MaKey MaKey!

This is all because the Arduino is open source hardware and the Arduino is open source software. You can read more about what that means online at [http://en.wikipedia.org/wiki/Open-source\\_software](http://en.wikipedia.org/wiki/Open-source_software) and [http://en.wikipedia.org/wiki/Open-source\\_hardware](http://en.wikipedia.org/wiki/Open-source_hardware).

# Sensing Light

In [Adventure 2](#) you were introduced to analogue inputs, using the potentiometer to control a status message sign. Here's a little refresher about how analogue inputs work.

An analogue input on an Arduino board is an input that can read in voltages between ground and 5V. A digital input can read in *either* ground or 5V, and it can't tell if an input is only 2.3V or any other value *between* ground and 5V (see [Figure 7-5](#)).



[Figure 7-5](#) Analogue and digital signals

There are a lot of different sensors that output analogue signals and not digital signals—that is they don't detect only whether something is on or off; they also measure how much there is of something. Microphones measure sound, accelerometers measure movement and light sensors measure light. All of these sensors output an analogue signal, which would be read into an Arduino board as an analogue input. The type of light sensor you work with in this adventure is a **light-dependent resistor** (LDR). An LDR is a resistor that changes its resistance according to how much light it is exposed to. Sometimes LDRs are also called photoresistors. I like the name LDR because it describes exactly how the resistor works—it depends on the light.



A **light-dependent resistor** (LDR) changes its resistance according to how much light it is exposed to. It is also sometimes called a photoresistor.



Recky says Use a multimeter to measure the resistance of your LDR. Connect one probe to one leg of the LDR and the other probe to the other leg. Set the multimeter to measure resistance and see what value you get. Try

shining a light on the LDR to see what happens. What happens when you cover the LDR and block out light?

## Building the Circuit

An Arduino board can only measure voltage, but an LDR only changes its resistance, which makes for a bit of a puzzle. How can you get the Arduino to see how the LDR is responding to brighter or darker environments if the LDR only changes resistance and not voltage? The trick is to use some clever circuit design.

Voltage, current and resistance are all connected. You can't change one of those things in a circuit without changing the others. The way they are all related is defined by **Ohm's Law**, which is shown in [Figure 7-6](#).

Ohm's Law

$$V = I \times R$$

Voltage in Volts = Current in Amps × Resistance in Ohms

Can also be rearranged to:

$$R = \frac{V}{I}$$

Resistance in Ohms =  $\frac{\text{Voltage in Volts}}{\text{Current in Amps}}$

[Figure 7-6](#) Ohm's Law defines how voltage, current and resistance are related.



**Ohm's Law** is the mathematical relationship between voltage, current and resistance. Voltage equals current multiplied by the resistance—or, put another way,  $V=IR$ .

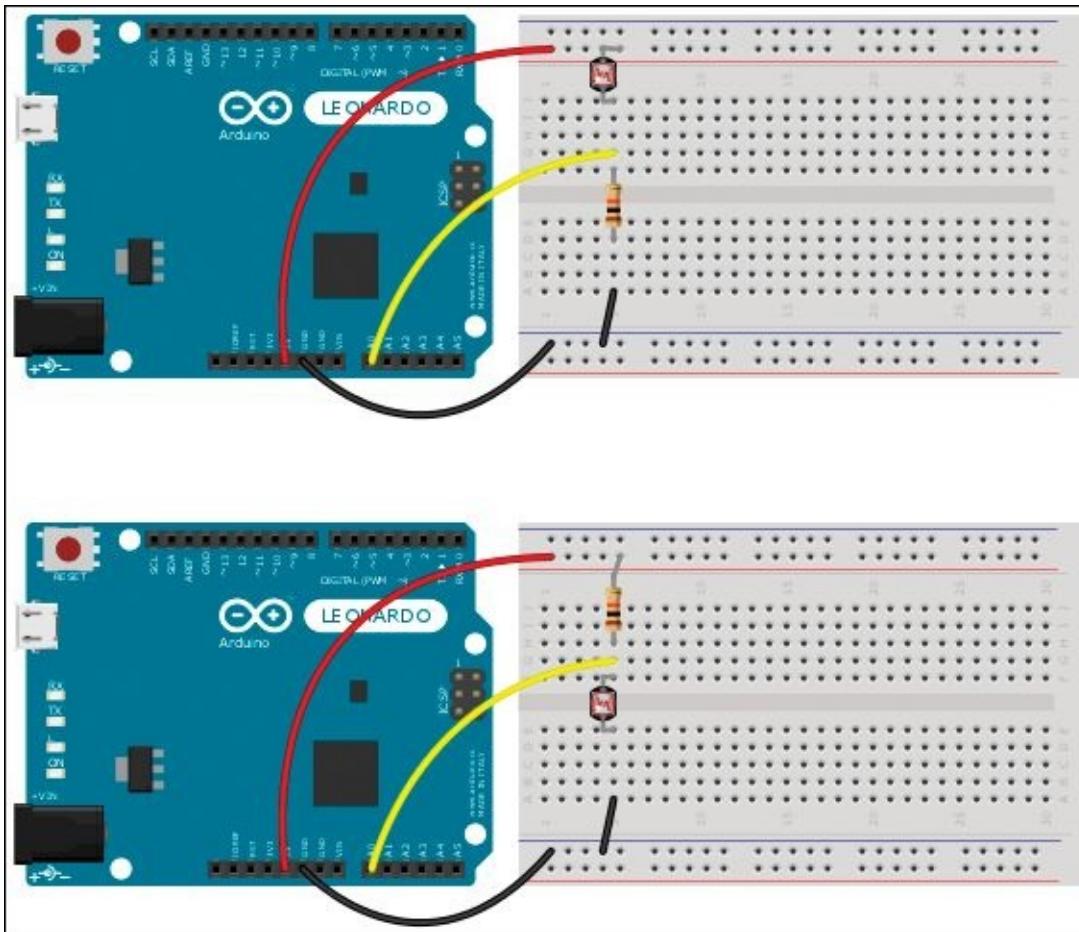
You don't have to worry too much about the details of that equation right now. The important thing to know is that you can design a circuit that changes the voltage when you change the resistance. The type of circuit you are going to build to do this is called a **voltage divider**.



A **voltage divider** is a circuit that outputs a fraction of the input voltage. It is a useful circuit for translating a change in resistance into a change in voltage.

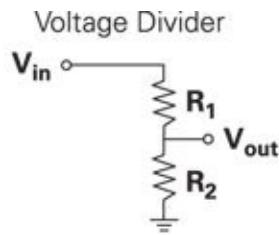
The voltage divider you are going to make has two different resistors. One of the resistors is a “normal” fixed-value resistor; the other is a variable resistor that changes its value. You are going to use an LDR as the variable resistor here, but in a future project you could build this circuit with another sensor that is a variable resistor.

When the resistance of the LDR goes up or down, the output voltage goes up or down. So when more or less light is shown on the LDR in the voltage divider circuit, the output voltage goes up and down. Whether you put the LDR in the top or bottom position affects whether the voltage goes up or down when you block out light from the LDR. [Figure 7-7](#) shows two ways that an LDR can be built into a voltage divider circuit.



[Figure 7-7](#) Two voltage divider circuits, one with an LDR as the top resistance and the other with an LDR as the bottom resistance

Some people find it easier to understand how something works when it is described with a mathematical equation. If you would like to work out how the voltage output from the circuit changes when you change the resistor values, you can use the equation shown in [Figure 7-8](#). If you don't enjoy working with numbers and equations, then you can just ignore that for now and instead have a go at building the circuit so you can actually see what is happening.



$$V_{\text{out}} = V_{\text{in}} \times \frac{R_2}{R_1 + R_2}$$

Output Voltage in Volts = Input Voltage in Volts  $\times \frac{\text{Bottom Resistor in Ohms}}{\text{Top + Bottom Resistors in Ohms}}$

**Figure 7-8** The equation to calculate how different resistor values in a voltage divider change the output voltage

## Writing the Code

One of the great things about working with Arduino boards is that the board doesn't care what sensor is connected to its input pins. It only cares if it is outputting an analogue or digital signal. If you've worked through [Adventure 2](#), you already have seen all the Arduino code to read in the values from your LDR—it's the same code you would use to read from a potentiometer. Open the example code at File=>Examples=>01.Basics=>AnalogRead in the Arduino IDE to see an example sketch that reads in the value from analogue input A0 and then prints that value to the Serial Monitor.

In the AnalogRead sketch, the `setup()` function starts the serial communication:

```
// the setup routine runs once when you press reset:  
void setup() {  
    // initialize serial communication at 9600 bits per second:  
    Serial.begin(9600);  
}
```

The `loop()` function reads in the value on Analog Pin A0 and saves it in a variable called `sensorValue`. That variable is then printed to the Serial Monitor:

```
// the loop routine runs over and over again forever:  
void loop() {  
    // read the input on analog pin 0:  
    int sensorValue = analogRead(A0);  
    // print out the value you read:  
    Serial.println(sensorValue);  
    delay(1);           // delay in between reads for stability  
}
```

# CHALLENGE



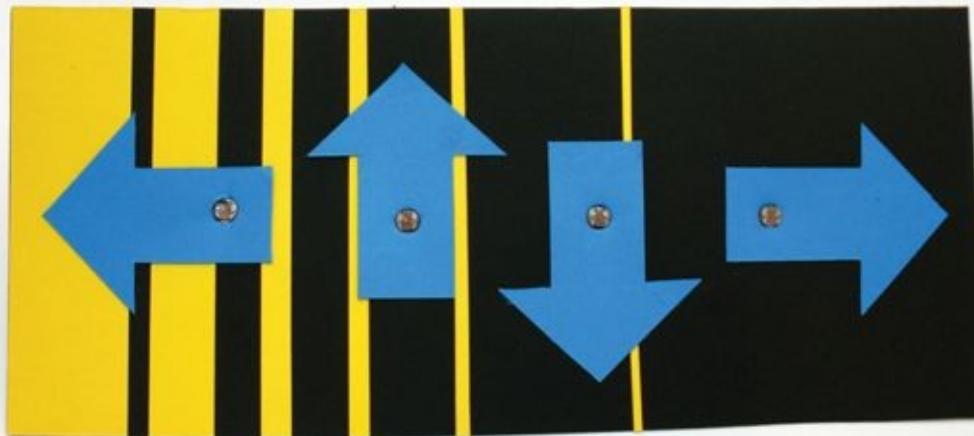
Build the voltage divider circuit with the LDR in the top position and upload the AnalogReadSerial sketch to your Arduino board. Open Serial Monitor and observe what happens when you shine or block light from the sensor.

Swap the LDR and fixed value resistor so the LDR is now in the bottom position. Watch what values are printed in the Serial Monitor. When might you want to use the first circuit, and when might you prefer to use the second one?

# Building a Game Controller

You now know how to get an Arduino Leonardo to act like a USB keyboard. Do you also like to play computer games? If you do, you probably can think of some games you can play using only the keyboard as input. If you don't play a lot of computer games, that's okay; I can point out some that you can try!

If you can get your Leonardo to act like a keyboard that controls a computer game, then you can start designing your own game controller like the one in [Figure 7-9](#). That opens up a wide world of different sensors that you can use to play a game. Boring buttons are a thing of the past! You're going to build a controller that uses light to control key presses, but you could use any sensor you like!



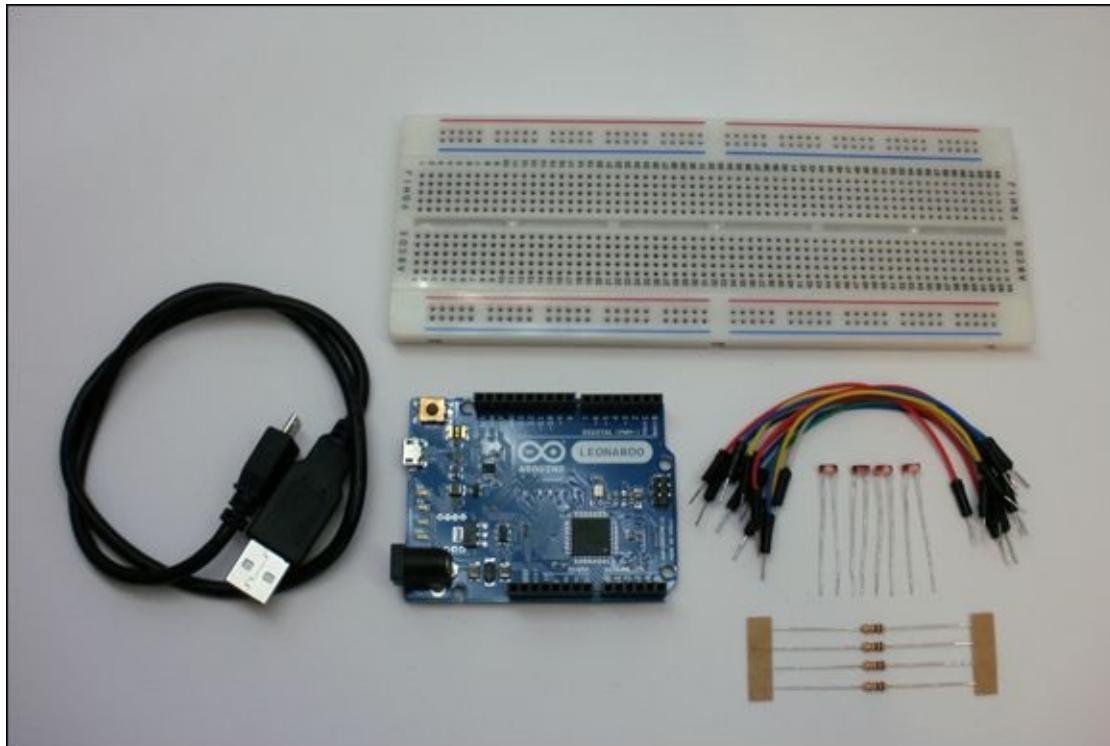
[Figure 7-9](#) Arduino Leonardo game controller



You watch a video on how to build your controller on the companion site ([www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino)).

## What You Need

You need the following supplies to make a game controller ([Figure 7-10](#) shows the electronic components you need):



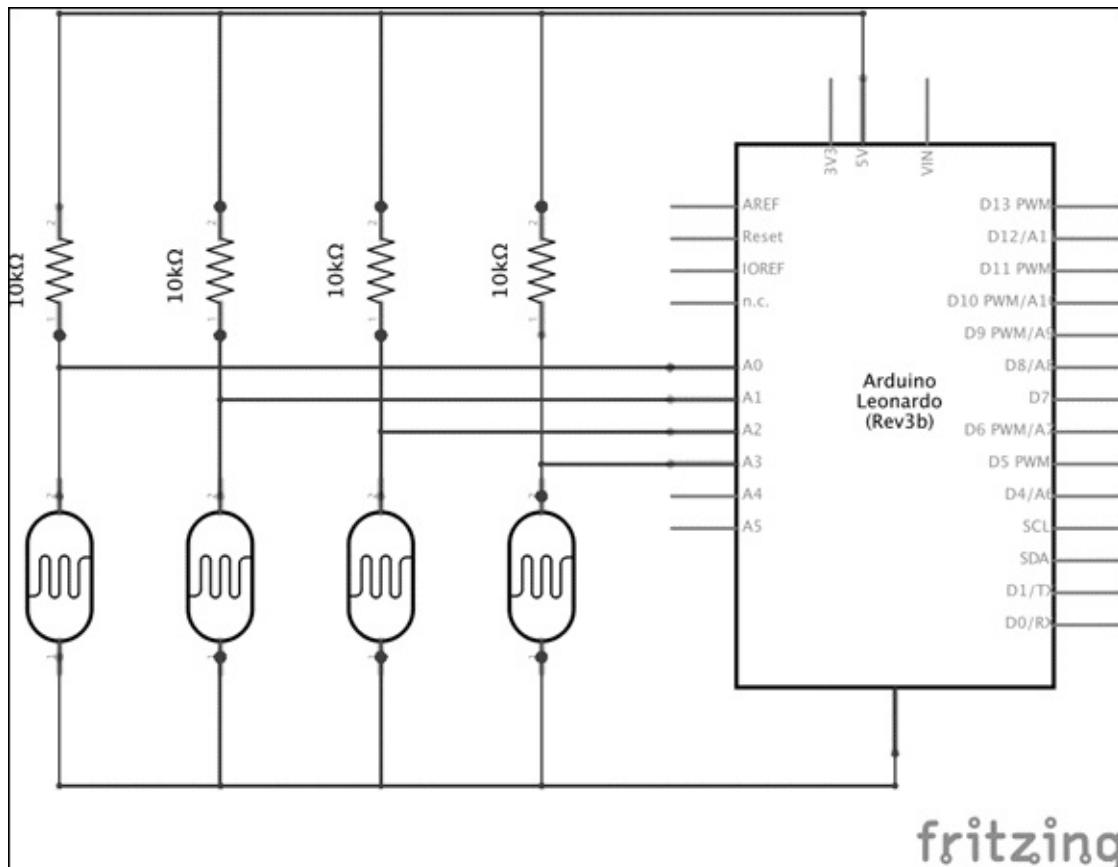
[Figure 7-10](#) The electronic components you need to make the game controller

- A computer
- An Arduino Leonardo
- A USB Micro cable
- A breadboard
- 10 jumper wires
- 4  $10\text{k}\Omega$  resistors
- 4 light-dependent resistors
- A piece of paper or card to cover the breadboard
- Some markers or coloured pencils
- A pair of scissors or a utility knife

## Building the Circuit

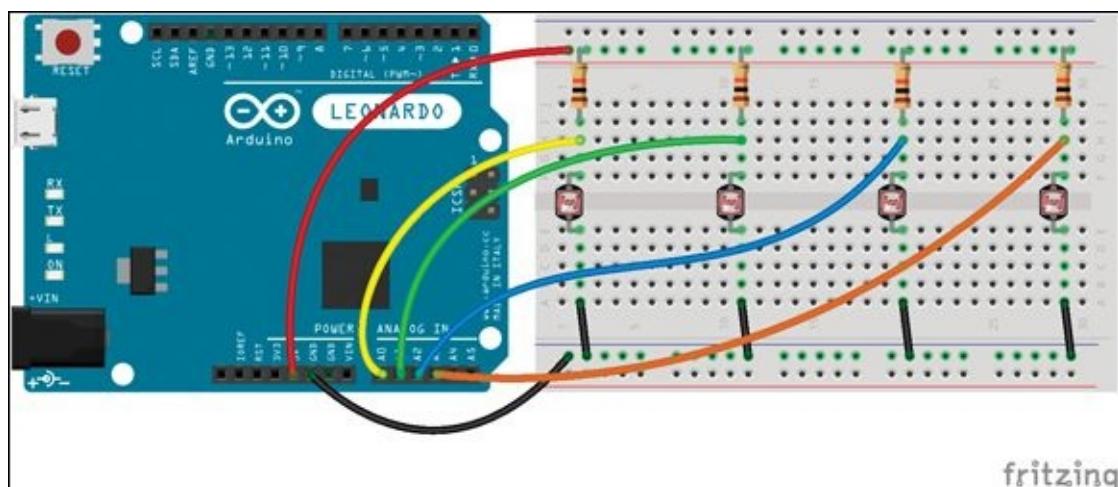
The circuit for the game controller doesn't need to have any extension wires soldered to it, so you can build it directly on your breadboard.

The game controller circuit consists of the same circuit repeated four times. There are four LDRs; each one controls a different arrow key. Each LDR is in its own voltage divider circuit with a fixed value resistor. [Figure 7-11](#) shows the circuit schematic for the game controller.



[Figure 7-11](#) Circuit schematic for the game controller

Use the following steps to build the circuit in [Figure 7-12](#) on your breadboard. It's good to space out the LDRs so they aren't too close to each other. You don't want to accidentally block the light from a LDR when you are trying to block the LDR next to it:



**Figure 7-12** The game controller circuit

1. Position the four LDRs by spanning the gap in the middle of the breadboard.
2. Connect each of the resistors between one of the long rows along the top of the breadboard and the LDRs.
3. Connect a jumper from the bottom leg of each of the LDRs to a long row on the bottom of the breadboard.
4. Connect the long row along the top, connecting the resistors to the 5V pin on the Arduino Leonardo.
5. Use a jumper wire to connect the long row along the bottom of the breadboard with the other jumper wires to a GND pin on the Arduino Leonardo.
6. Using four jumper wires, connect pins A0, A1, A2 and A3 to each of the LDRs. One end of the jumper wire plugs into the pin on the Arduino Leonardo and the other end plugs into the same row as the resistor leg and LDR leg.

## Writing the Code

In order to make sure your circuit is working properly, you first program the Leonardo to print messages to the Serial Monitor before programming it to act like a keyboard. After you know everything is okay with the circuit and you've figured out the thresholds for when the sensor should trigger a message, you replace the serial messages with keyboard messages.

Start by creating a new Arduino sketch and creating an empty `setup()` and `loop()`:

```
void setup() {  
}  
  
void loop() {  
}
```

At the top of the sketch before `setup()`, declare and initialise the variables for the input pins. There are five inputs: the switch pin and one input for each of the four LDRs:

```
int switchPin = 4;  
int leftSensor = A0;  
int rightSensor = A1;  
int upSensor = A2;  
int downSensor = A3;
```

Inside `setup()` start serial communication and set the pin mode for the `switchPin`:

```
// make the switchPin an input with an internal pull-up resistor  
pinMode(switchPin, INPUT_PULLUP);  
// initialize control over the keyboard:  
Serial.begin(9600);
```

Inside the `loop()`, check the state of `switchPin` and create an `if` statement that is true if `switchPin` is `LOW`:

```
// read the pushbutton:  
int switchState = digitalRead(switchPin);  
// if the switch is open (not connected to ground),  
if (switchState == LOW) {  
  
}  
delay(50);
```

The `delay()` makes sure messages can't be sent too quickly.

At the top of your code with your other variables, add the following lines of code:

```
int rightThreshold = 400;  
int leftThreshold = 400;  
int upThreshold = 400;  
int downThreshold = 400;
```

These variables keep track of when each LDR circuit will trigger a message.



You probably need to use a value other than 400 for your LDRs. You may need to make the values higher or lower, and each LDR might even require a different value! Later you will figure out the best values for your controller by trying out different values, but you aren't ready to do that yet! You need to finish the sketch first.

Inside your **if** statement, add the following code. Even though it's long, it's just the same thing repeated four times—once for each LDR:

```
// RIGHT ARROW
int rightValue = analogRead(rightSensor);
//Serial.println(rightValue);
if(rightValue > rightThreshold) {
    Serial.println("right arrow");
}

// LEFT ARROW
int leftValue = analogRead(leftSensor);
//Serial.println(leftValue);
if(leftValue > leftThreshold) {
    Serial.println("left arrow");
}

// UP ARROW
int upValue = analogRead(upSensor);
//Serial.println(upValue);
if(upValue > upThreshold) {
    Serial.println("up arrow");
}

// DOWN ARROW
int downValue = analogRead(downSensor);
//Serial.println(downValue);
if(downValue > downThreshold) {
    Serial.println("down arrow");
}
```

In each of the four blocks of code, the value from the analog pin is read in and saved to a variable. If that variable is less than the threshold value for the LDR, then the message is printed.

Inside each block of code is a line of code commented out. If you uncomment that line (by deleting **//**) then the value of that pin is printed. This can be useful to help set your threshold values, but it also prints a lot of numbers and can be confusing. You probably want to uncomment only one of them at a time.

What follows here is the full code for trying out printing serial messages. Build your circuit and upload the code to your Arduino Leonardo.



Remember that your sketch won't check the values of the pins unless you have connected Pin 4 to GND!

```
int switchPin = 4;
int leftSensor = A0;
int rightSensor = A1;
int upSensor = A2;
int downSensor = A3;

// adjust these variables to values that
// work for your controller
int rightThreshold = 400;
int leftThreshold = 400;
int upThreshold = 400;
int downThreshold = 400;

void setup() {
    // make the switchPin an input // with an internal pull-up resistor
    pinMode(switchPin, INPUT_PULLUP);
    // initialize control over the keyboard:
    Serial.begin(9600);
}

void loop() {
    // read the pushbutton:
    int switchState = digitalRead(switchPin);
    // if the switch is open (not connected to ground),
    if (switchState == LOW) {
        // RIGHT ARROW
        int rightValue = analogRead(rightSensor);
        //Serial.println(rightValue);
        if(rightValue > rightThreshold) {
            Serial.println("right arrow");
        }

        // LEFT ARROW
        int leftValue = analogRead(leftSensor);
        //Serial.println(leftValue);
        if(leftValue > leftThreshold) {
            Serial.println("left arrow");
        }

        // UP ARROW
        int upValue = analogRead(upSensor);
        //Serial.println(upValue);
        if(upValue > upThreshold) {
            Serial.println("up arrow");
        }

        // DOWN ARROW
        int downValue = analogRead(downSensor);
        //Serial.println(downValue);
        if(downValue > downThreshold) {
            Serial.println("down arrow");
        }
    }
    delay(50);
}
```



As the sketches get longer, you may prefer to download them from the companion site ([www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino)) instead of typing them out.

Now open the Serial Monitor and try out your LDRs one by one, by holding your hand over each one. Adjust the threshold values until they print a message only when you want them to.

After you know what your thresholds should be, you can change your sketch to output key presses instead of serial messages.

Save your sketch and then create a new empty sketch. Copy and paste the sketch that you just saved into the new sketch. You are going to keep most of what you have already written and replace the lines of code that use **Serial** functions with **Keyboard** functions.

In the **setup()**, replace the line that starts serial communication with the following:

```
// initialize control over the keyboard:  
Keyboard.begin();
```

In each of the four blocks of code that print the serial message about which sensor was triggered, replace the **Serial.println()** with the following line, using **KEY\_RIGHT\_ARROW**, **KEY\_LEFT\_ARROW**, **KEY\_UP\_ARROW** and **KEY\_DOWN\_ARROW**:

```
Keyboard.press(KEY_RIGHT_ARROW);
```

Instead of printing a message, the Leonardo is sending a message that a key has been pressed. At the end of the loop, the following line of code sends the message that all the keys have been released:

```
Keyboard.releaseAll();
```

When you put it all together, you have the following sketch:

```
int switchPin = 4;  
int leftSensor = A0;  
int rightSensor = A1;  
int upSensor = A2;  
int downSensor = A3;  
  
// adjust these variables to values that  
// work for your controller  
int rightThreshold = 400;  
int leftThreshold = 400;  
int upThreshold = 400;  
int downThreshold = 400;  
  
void setup() {  
    // make the switchPin and input // with an internal pull-up resistor  
    pinMode(switchPin, INPUT_PULLUP);  
    // initialize control over the keyboard:  
    Keyboard.begin();  
}
```

```

void loop() {
    // read the pushbutton:
    int switchState = digitalRead(switchPin);
    // if the switch is open (not connected to ground),
    if (switchState == LOW) {
        // RIGHT ARROW
        int rightValue = analogRead(rightSensor);
        //Serial.println(rightValue);
        if(rightValue > rightThreshold) {
            Keyboard.press(KEY_RIGHT_ARROW);
        }

        // LEFT ARROW
        int leftValue = analogRead(leftSensor);
        //Serial.println(leftValue);
        if(leftValue > leftThreshold) {
            Keyboard.press(KEY_LEFT_ARROW);
        }

        // UP ARROW
        int upValue = analogRead(upSensor);
        //Serial.println(upValue);
        if(upValue > upThreshold) {
            Keyboard.press(KEY_UP_ARROW);
        }

        // DOWN ARROW
        int downValue = analogRead(downSensor);
        //Serial.println(downValue);
        if(downValue > downThreshold) {
            Keyboard.press(KEY_DOWN_ARROW);
        }
    }

    delay(50);
    Keyboard.releaseAll();
}

```

Build the circuit and upload the sketch. Open a spreadsheet program, such as Microsoft Excel, and test that you can use your controller to move to different squares just as you could with the arrow keys.



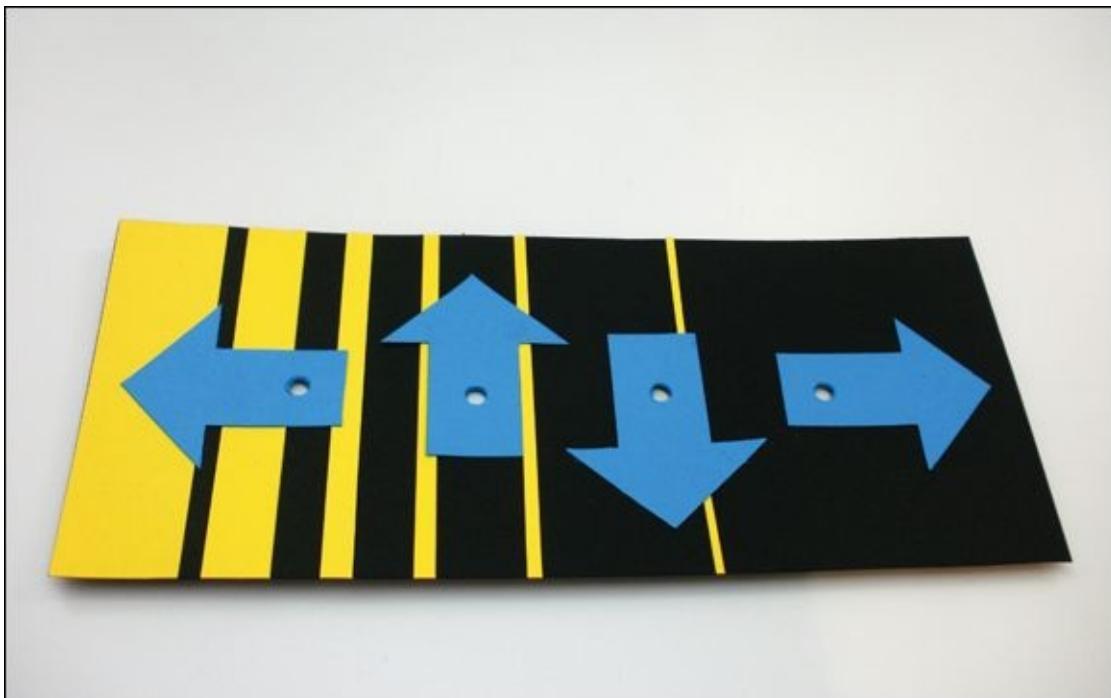
Does your game use keys other than the arrows? Or maybe it needs the arrow keys and the spacebar? You can read more about all the keys the Arduino Leonardo can press on the Arduino website at <http://arduino.cc/en/Reference/KeyboardModifiers>.

## Making the Controller Cover

Using paper or card, cut out a shape for your game controller. It can be a rectangle or any other shape. It doesn't have to look like any game controller you've ever seen before! It just needs to be big enough to cover your breadboard.

Decorate your cover however you'd like. It would be a good idea to label which key is triggered by which LDR. When you're at a tricky part in your game, you don't want to forget which sensor is which key and accidentally lose your game!

Using scissors, poke a hole where each LDR will be placed (see [Figure 7-13](#)). This is where the wire legs of the LDR will pass through the cover so that the top part of the sensor is above the cover, but the legs are connected to the circuit underneath on the breadboard.



[Figure 7-13](#) Cover without any circuitry

## Putting It All Together

Since you've already tested your controller with a spreadsheet program, the last step is to add your controller cover before playing your games with your new controller.

One by one, remove an LDR from the breadboard and poke its legs through the holes in the paper cover. Then connect that LDR back into the circuit on the breadboard. You can always solder longer wires onto the LDRs if you find it too tricky to connect the LDRs back into circuit on the breadboard.

Test your controller again with the spreadsheet program just to make sure you've connected everything correctly. After you've confirmed that it's all okay, load up your favorite game that uses the arrow keys and start playing!



If you are looking for some games to play with your new controller, 2048 at <http://gabrielecirulli.github.io/2048> is a good one to try. It's a number puzzle game. I also like the maze game at [www.primarygames.com/puzzles/mazes/mazerace](http://www.primarygames.com/puzzles/mazes/mazerace). It's much harder than it looks!

# Further Adventures with the Leonardo

You have now started working with more advanced circuits called voltage dividers. If you would like to learn even more about voltage dividers, you can start by checking out Sparkfun's tutorial at <https://learn.sparkfun.com/tutorials/voltage-dividers>.

For more information about the Arduino Leonardo and what you can do with it, visit the Arduino Leonardo page on the Arduino website at <http://arduino.cc/en/Guide/ArduinoLeonardoMicro?from=Guide.ArduinoLeonardo>.

You can also explore more examples using the **Keyboard** functions in the Arduino IDE by going to File⇒Examples⇒09.USB. Here, you will also find examples that turn your Arduino Leonardo into a mouse.

Boards like the Arduino Leonardo and Uno are open source hardware. That means that you can make new projects and even new boards that use the features of Arduinos, just like the MaKey MaKey. If you would like to learn more about open source hardware, check out the Open Source Hardware Association (OSHWA) at [www.oshwa.org/](http://www.oshwa.org/).

## Arduino Command Quick Reference Table

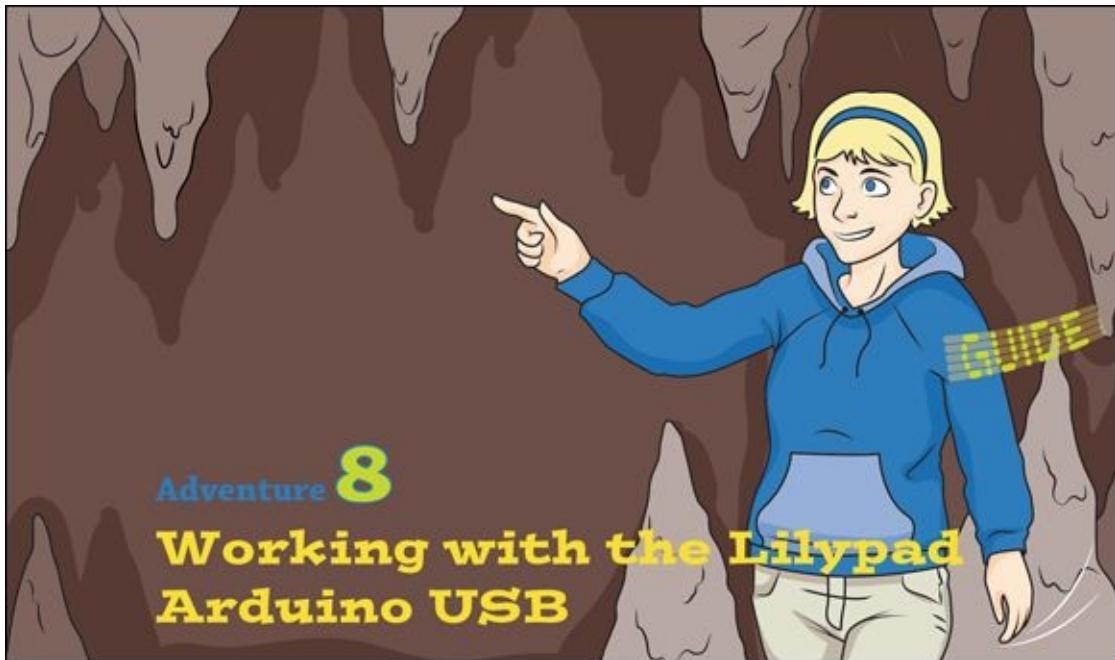
Command	Description
While(!Serial);	Causes the Leonardo to wait and do nothing until a serial connection is opened.
Keyboard.begin()	Begins keyboard functionality. See also <a href="http://arduino.cc/en/Reference/KeyboardBegin">http://arduino.cc/en/Reference/KeyboardBegin</a> .
Keyboard.println()	Sends a message to the computer as if it was typed on the keyboard. See also <a href="http://arduino.cc/en/Reference/KeyboardPrintln">http://arduino.cc/en/Reference/KeyboardPrintln</a> .
Keyboard.press()	Sends a message to the computer that a key is pressed. See also <a href="http://arduino.cc/en/Reference/KeyboardPress">http://arduino.cc/en/Reference/KeyboardPress</a> .
Keyboard.releaseAll()	Sends a message to the computer that all keys have been released. See also <a href="http://arduino.cc/en/Reference/KeyboardReleaseAll">http://arduino.cc/en/Reference/KeyboardReleaseAll</a> .



**Achievement Unlocked:** Expert constructor of a game controller. Well played!

## In the Next Adventure

In the next adventure, you continue your exploration of other types of Arduino boards to create an amazing wearable circuit!



**YOU HAVE ONLY** just begun to explore all the different ways to make a project with an Arduino! In [Adventure 7](#) you learned how to design and build your own game controller using the Arduino Leonardo. The Arduino board still looked exactly like an Arduino Uno, however, and you still built circuits the same way you would build your circuit for an Arduino Uno project—using breadboards, wire and solder.

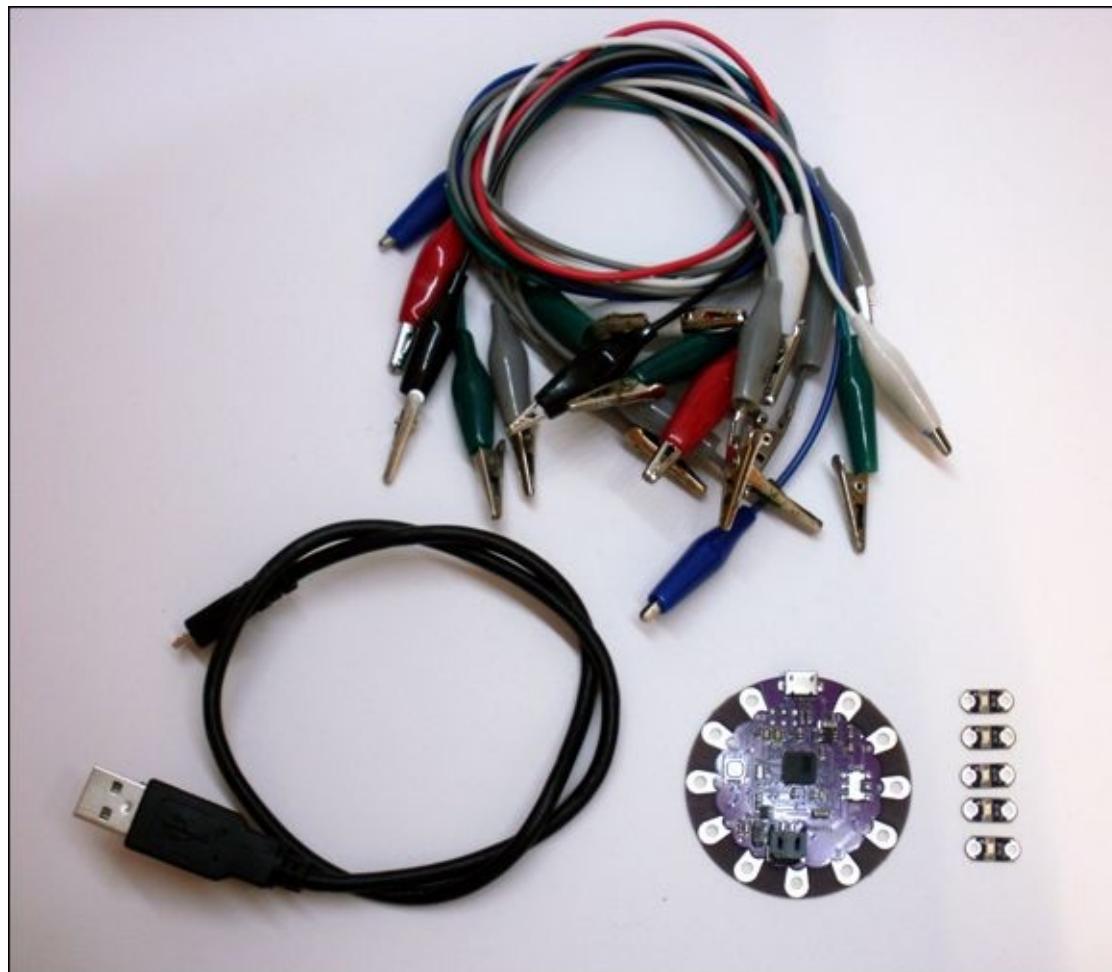
But what if, for example, you wanted to build an Arduino project that you could wear? You can't use stiff wires and hard solder if you want something to bend like fabric and be comfortable to wear. Luckily, conductive metals can be made into thread, which means you can sew a circuit with a needle and (conductive) thread. It's called making a soft circuit. And there's a particular type of Arduino board that has been built to make it easy to sew a microcontroller into your project: the Lilypad Arduino USB!

In this adventure you're going to create a hoodie with an Arduino and LEDs embedded in its sleeve. The LEDs will display a secret message that you program into your Lilypad Arduino USB. The secret message is stored on the Lilypad Arduino USB using arrays, so before you start writing the code for this adventure's project, I remind you how arrays work and show you how to push them to a second dimension.

# What You Need

You need the components shown in [Figure 8-1](#) for the first part of this adventure.

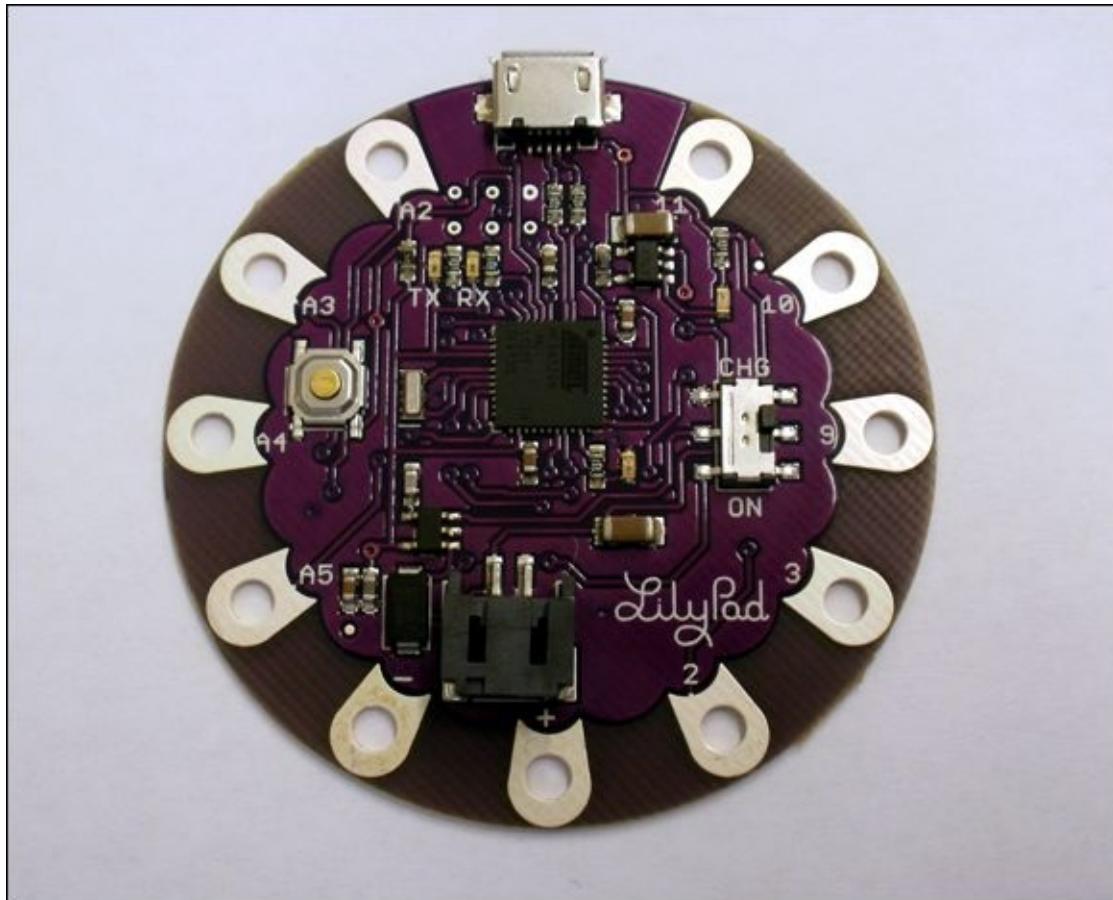
- A computer
- A Lilypad Arduino USB
- A USB Micro cable
- 10 alligator clips
- 5 Lilypad LEDs (or 5 LEDs and 5  $68\Omega$  or  $100\Omega$  resistors if not using Lilypad LEDs)



[Figure 8-1](#) The electronic components you need for the first part of this adventure

# Introducing the Lilypad Arduino USB

The Lilypad Arduino USB is a member of a family of Lilypad Arduinos. Although you could use any of the Lilypad Arduinos, the Lilypad Arduino USB shown in [Figure 8-2](#) has some features that make it a bit nicer to use, such as not needing a second board to upload sketches. But because they are all Arduinos, your code remains the same, so the code you write here will still work on a Lilypad Arduino, Lilypad Arduino Simple or Lilypad Arduino SimpleSnap.



[Figure 8-2](#) The Lilypad Arduino USB

The Lilypad Arduino USB has the same microcontroller chip as the Arduino Leonardo. That means you can turn it into a mouse or keyboard just like you can with an Arduino Leonardo, but it also means it is easier to program than the other Lilypad Arduinos. The Lilypad Arduino USB has a USB Micro connector on it—the same connector that is on the Arduino Leonardo. Programming the Lilypad Arduino USB is just like programming any of the other Arduino boards you have used so far: Connect the board to the USB cable and then connect the USB cable to the computer.

The other types of Lilypad Arduino don't have a USB Micro connector on them. Instead they have six pins on the top of the board. These pins connect to a FTDI board (see [Figure 8-3](#)), which has a USB connector on it. To program those boards, you first connect the FTDI board to the Lilypad Arduino, then connect the USB cable to the FTDI board and then connect the USB cable to the computer. So if you already have a Lilypad Arduino or can't find a Lilypad Arduino USB, don't worry! You can still use a different Lilypad

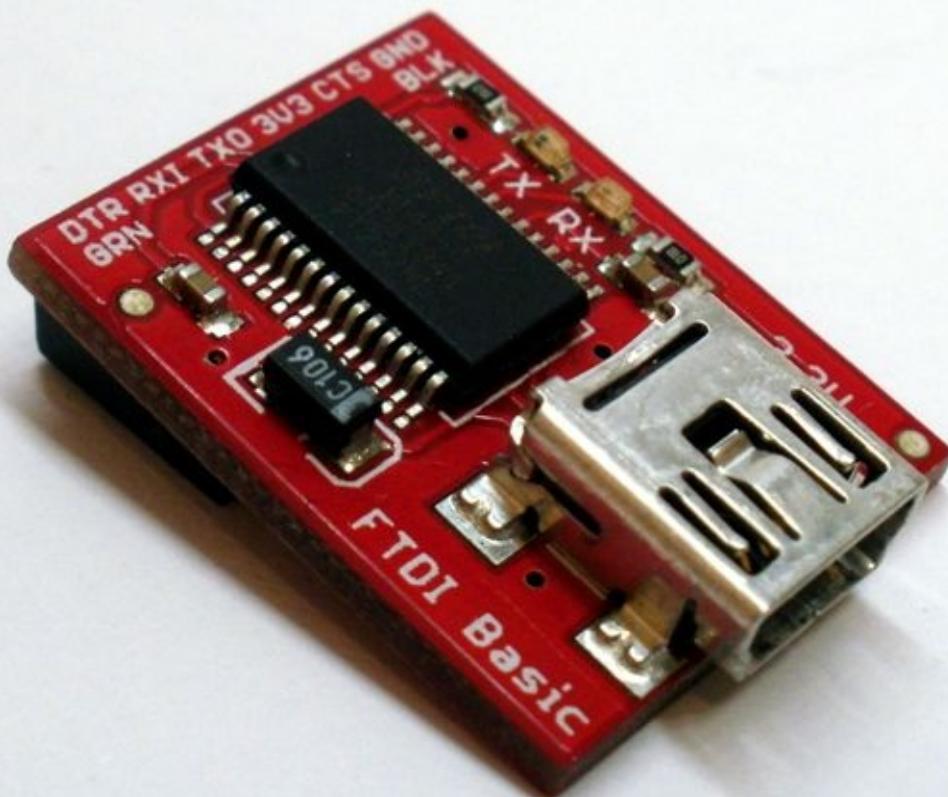
Arduino board; just make sure you also get an FTDI programming board.



FTDI stands for Future Technology Devices International, a company that makes a chip that was used in most Arduino boards. An FTDI chip translates the electrical signals sent by the computer over a USB cable into signals the Arduino can understand. The Arduino Uno uses a chip similar to the FTDI chip to do this translation for the microcontroller chip. It comes already included on the Arduino Uno's board.

The Lilypad Arduino doesn't have this extra chip on its board, so you have to connect the Lilypad Arduino to an FTDI board that has the chip on it whenever you want to upload new sketches. The Lilypad Arduino USB has a different microcontroller chip than the Arduino Uno or Lilypad Arduino. Its chip can handle the translation from the USB signal on its own without a second chip, so you don't need to use an FTDI board.

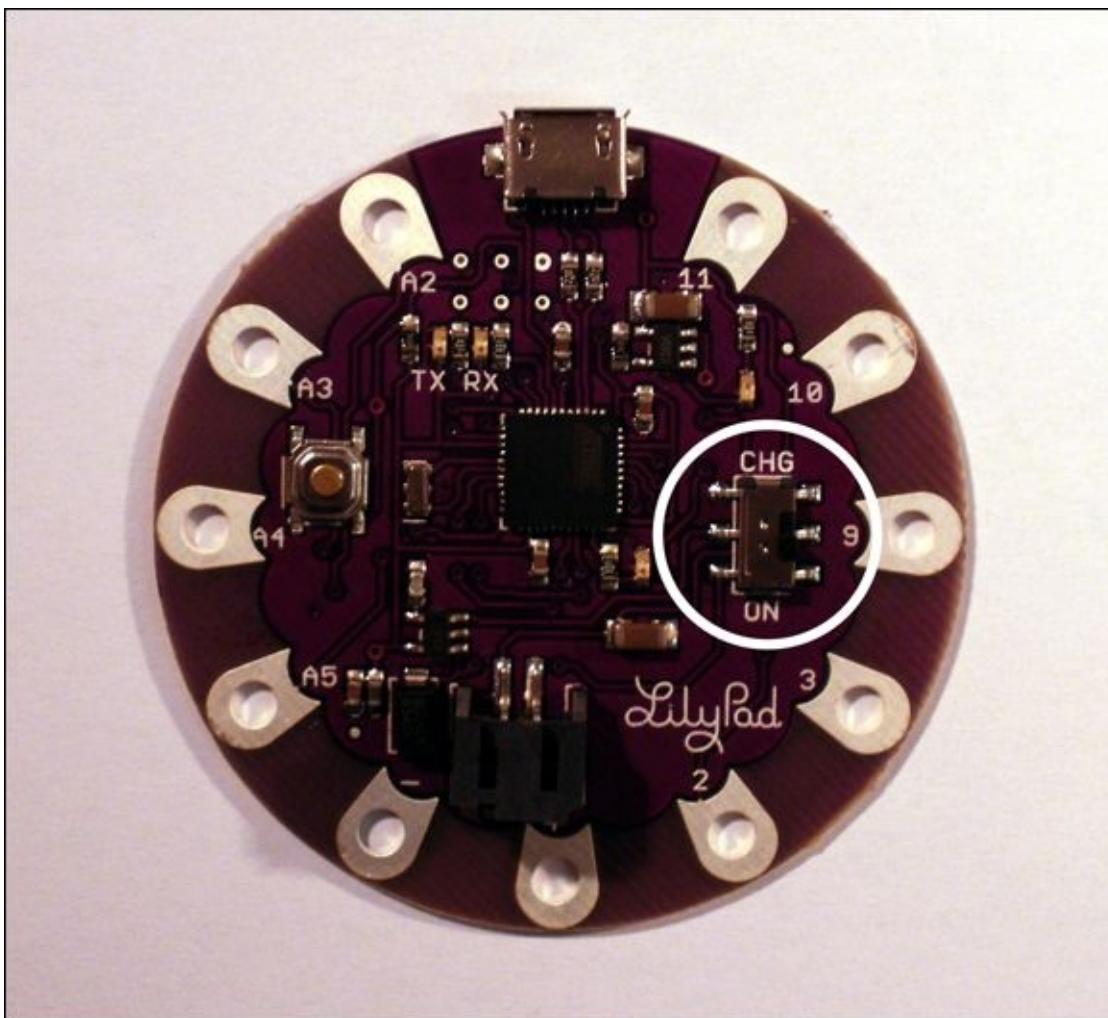
If you are interested in the design details that make the Arduino boards different from each other, check out Sparkfun's comparison guide at <https://learn.sparkfun.com/tutorials/arduino-comparison-guide/introduction>.



**Figure 8-3** An FTDI programming board, which you need if you are using a type of Lilypad Arduino other than a Lilypad Arduino USB

## Blinking from a Lilypad Arduino

The first and vitally important difference between the Arduino Uno and Leonardo and the Lilypad Arduino is that it has an ON switch! The board won't automatically turn on if you give it power, which can be confusing and frustrating if you don't know about the switch. You might even think your board is broken! But don't fret. The switch is on the top of the board, on the opposite side of the microcontroller chip from the reset button. [Figure 8-4](#) shows where it is.



[Figure 8-4](#) The Arduino Lilypad Arduino USB ON switch

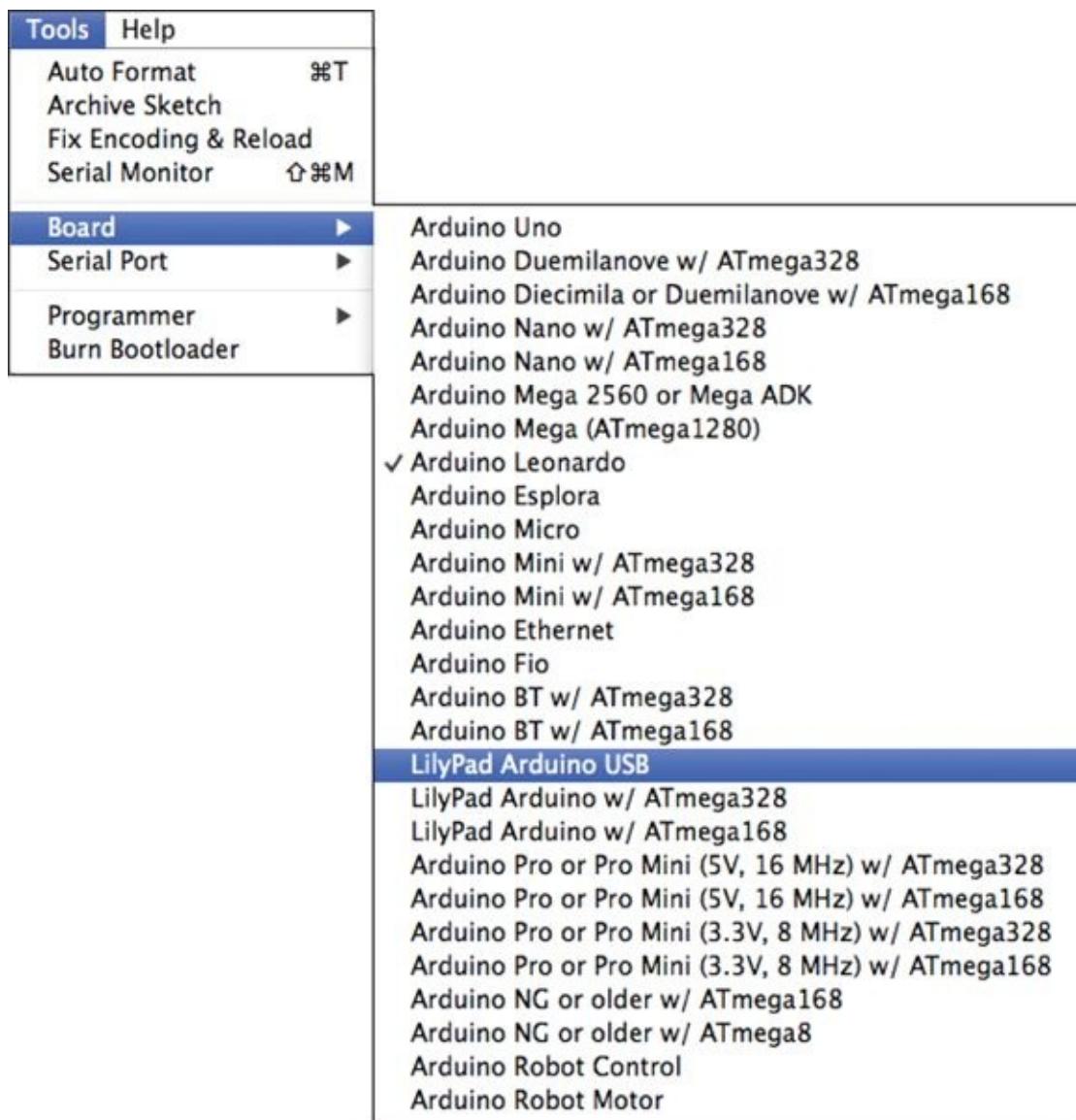


One of the things that the Lilypad Arduino USB can do that other boards can't do is charge a battery. This is convenient for wearable projects as they almost always need to be battery powered. The switch has two positions: ON and CHG. CHG is for charging a battery from the USB cable. You can't charge the battery and turn on the microcontroller at the same time. You will find out more about how to power the board with batteries later, when you build your hoodie.

Whenever you start using a new microcontroller or Arduino board, it's always good to begin with a "Hello World" program that demonstrates that your board is working correctly and you know how to program it. You first did this with your Arduino Uno in [Adventure 1](#). For Arduino, the Blink sketch is the "Hello World" program.

Open the Blink sketch by launching the Arduino IDE and going to File⇒Examples⇒01.Basics⇒Blink. You don't need to change any of the code and can immediately upload it to the board.

To upload it, you need to connect the board to your computer and then tell the Arduino IDE which board you are using. Go to Tools⇒Board and select Lilypad Arduino USB from the list, as shown in [Figure 8-5](#). Then click the Upload button.



[Figure 8-5](#) Select Lilypad Arduino USB from the list of boards

A light on the Lilypad Arduino USB starts blinking, turning on for one second then off for one second.

## CHALLENGE



New Arduino boards often come with Blink already running on them, so your board might already blink its LED on and off on Pin 13 for one second at a time. Change the timing in the Blink sketch so that the LED is off for only half a second instead of a full second. Upload it to the board and make sure the LED now blinks according to the new sketch.

# Prototyping Soft Circuits

Now that you know how to upload a sketch to your Lilypad Arduino USB, you can start connecting **sensors** and **actuators**. The puzzling thing is how are you supposed to do that? The Lilypad doesn't have pins that you can plug jumper wires into. It has big pads with holes that are large enough to allow you to sew conductive thread easily to create **soft circuits**, but you don't want to sew every circuit. That takes a lot of time. So how do you prototype with soft circuits? The answer is alligator clips (sometimes also called crocodile clips).



A **sensor** is a device that detects something in the real world such as light, sound or movement and translates it into an electrical signal. Examples include potentiometers and light-dependent resistors.

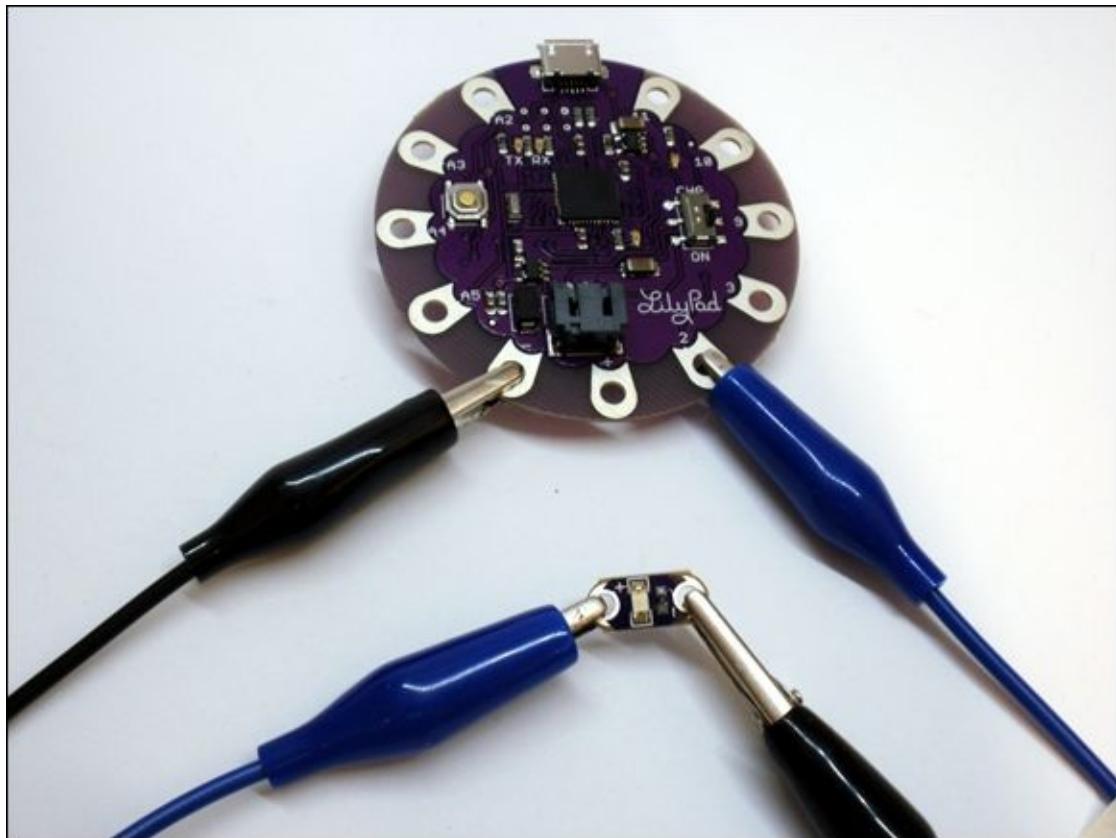


An **actuator** is a device that translates an electrical signal into a real-world action such as light, sound or movement. Examples include motors, lights and speakers.



**Soft circuits** are circuits built with flexible materials like conductive thread and fabric. They are often used in projects that are going to be worn.

**Alligator clips**, shown in [Figure 8-6](#), are wires with spring-loaded clips that resemble the jaws of an alligator. You attach one end of the clip onto the Lilypad Arduino USB pad and the other to the next part of the circuit, such as an LED.



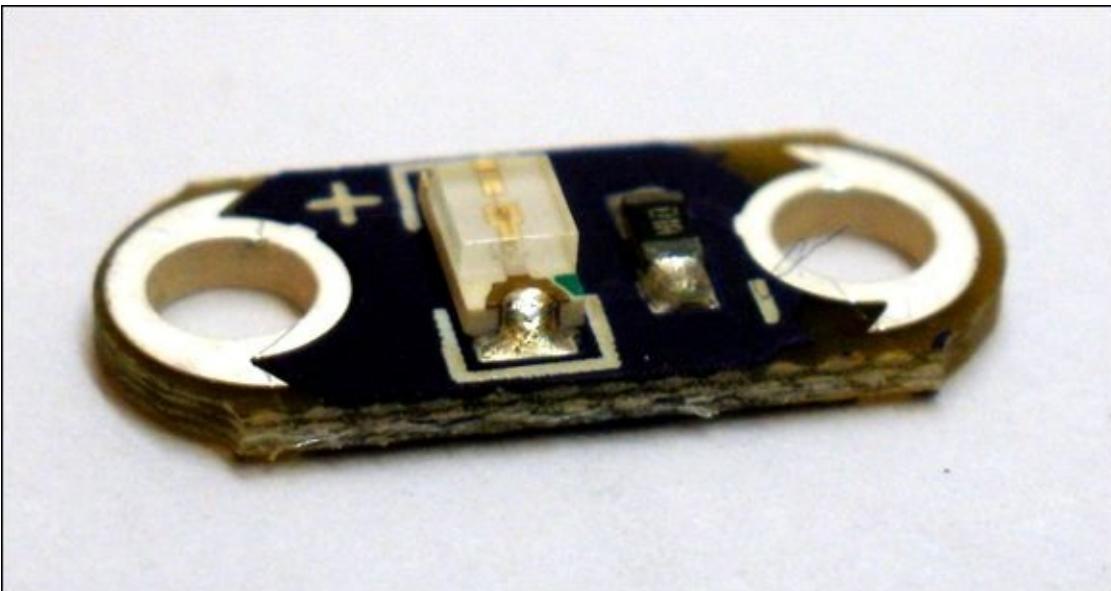
**Figure 8-6** Instead of jumper wires to connect components, use alligator clips when prototyping soft circuits.



**Alligator clips** are wires with spring-loaded clips that resemble the jaws of an alligator. They are useful for prototyping soft circuits or connecting components that don't use jumper wires.

The Lilypad Arduino USB operates at a lower voltage than the Arduino Uno and Leonardo. When you set a pin to **HIGH**, it outputs 3.3V instead of 5V. You don't have to worry about why it uses 3.3V instead of 5V, but it does mean that your LED doesn't need a current-limiting resistor with as high a resistance value. You can use a resistor that's anywhere from 68 to  $100\Omega$ .

You can also buy Lilypad LEDs like the ones in [Figure 8-7](#) for your soft circuit projects. These are LEDs on purple boards (that match the Lilypad Arduino boards), and they already have current-limiting resistors on the board with them. As a bonus, you don't have to sew as many components!



**Figure 8-7** LilyPad LEDs are sewable LEDs that already have current-limiting resistors.

## CHALLENGE



Use alligator clips to build a circuit with your Lilypad Arduino USB and an LED. If you have a regular LED, be sure to use a current-limiting resistor that is either 68 or  $100\Omega$  as well; otherwise, you can use a Lilypad LED.

Choose one of the pads to connect to your LED circuit. Note that you don't have as many to choose from as you do with other Arduino boards. Your options for digital pins are 2, 3, 9, 10 11. Choose a pin and then modify the Blink sketch to use your chosen pin instead of Pin 13. Upload your sketch and blink your LED!

# Getting Clever with Arrays

In [Adventure 5](#), you were first introduced to arrays for storing a list of values without needing to create a new variable for each one. To create a list of `ints`, create a variable that is the data type `int[]` and then list the numbers inside `{ }`, as shown in [Figure 8-8](#).

```
int listOfInts[] = { 7, 9, 5, 1, 9, 2 };  
listOfInts[1] is the int 9  
listOfInts[5] is the int 2
```

listOfInts

Item Number	0	1	2	3	4	5
Item Value	7	9	5	1	9	2

[Figure 8-8](#) A list of integers, also called a one-dimensional array

Lists are one-dimensional arrays, but arrays can have more than one dimension. They can have any number of dimensions you like, but it can be difficult to picture something with more than three dimensions. **Two-dimensional arrays** are something you probably are already used to—they are data arranged in rows and columns like a spreadsheet. [Figure 8-9](#) shows one way you can picture a two-dimensional array.



A **two-dimensional array** is data stored in rows and columns like in a spreadsheet.

```
int arrayOfInts[3][6] = {{7, 9, 5, 1, 9, 2},  
                         {9, 0, 3, 3, 2, 2},  
                         {5, 4, 2, 7, 8, 3}};
```

arrayOfInts[1][2] is the int 4

arrayOfInts[0][5] is the int 2

arrayOfInts		Item Number	0	1	2	3	4	5
0	1		7	9	5	1	9	2
1	2		9	0	3	3	2	2
2	3		5	4	2	7	8	3

**Figure 8-9** A two-dimensional array of integers stored in rows and columns

You will be using two-dimensional arrays to store the messages that your hoodie will display. Each letter of your message will be an array. Picture an LED sign at a bus stop or train station that displays letters and numbers. You can think of each character as taking up a rectangle of space—each letter or number has a width and height. Within that rectangle the lights are turned on in a pattern to show that letter or number. Those rectangles of LEDs for each letter or number are two dimensional arrays.

To create a two-dimensional array requires a little more work than creating a one-dimensional array. In a one-dimensional array, you don't have to say how many items are in your list. You can just list them between the `{` and the `}`, and the Arduino IDE counts them for you. When creating a two-dimensional array, however, you have to count the items yourself.

The following code creates a variable called `twoDArray` that has three columns and two rows:

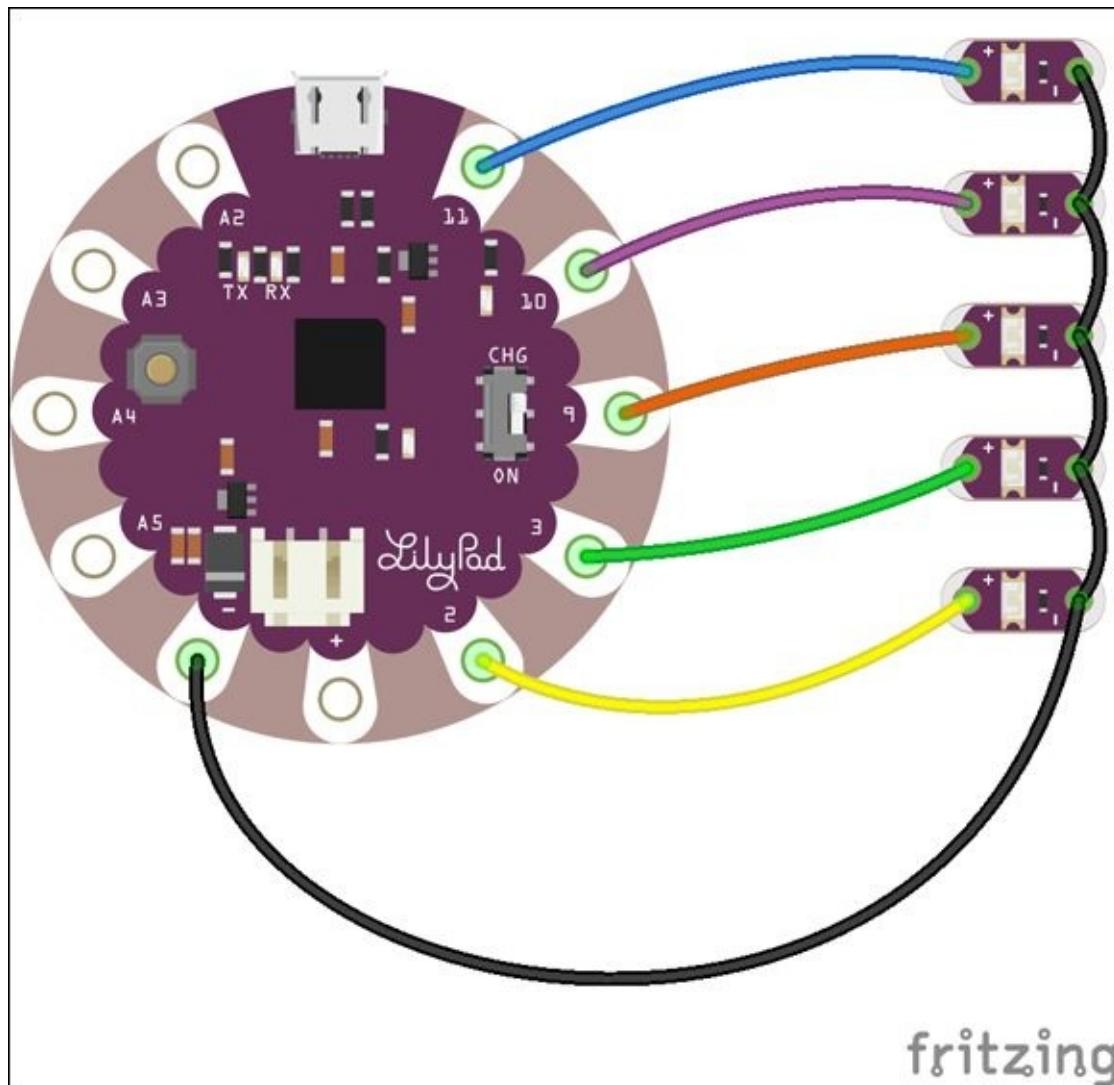
```
int twoDArray[2][3] = {{1, 2, 3},  
                      {4, 5, 6}};
```

You might want to store the number of columns and rows in variables as it is useful information to refer to later. If you want to do this, you need to add `const` in front of the variables. That means the variable's value won't ever change.

```
const numRows = 2;  
const numCols = 3;  
int twoDArray[numRows][numCols] = {{1, 2, 3},  
                                    {4, 5, 6}};
```

It might not yet be clear why you would ever want to use two-dimensional arrays, so let's

build a circuit that uses them. Build the circuit in [Figure 8-10](#) with an LED attached to Pins 2, 3, 9, 10 and 11. Start by connecting each pin to the positive side of one of the Lilypad LEDs. Then connect all the negative sides of the Lilypad LEDs to each other. Connect the negative side of the LED connected to Pin 2 to the GND pad of the Lilypad Arduino USB.



[Figure 8-10](#) Circuit for an array of LEDs



If you don't have Lilypad LEDs, replace them with current-limiting resistors that are 68 or  $100\Omega$  and normal LEDs.

You're now going to animate the LEDs and store each frame of the animation in a two-dimensional array. Start by opening the Arduino IDE and creating a new sketch. Create an empty `setup()` and `loop()`:

```
void setup() {  
}  
  
void loop() {
```

```
}
```

Create two variables that hold the number of rows and columns in the array at the top of the sketch. The array has one column for each LED (five columns) and one row for each frame of the animation. The simple animation for the example has nine frames (nine rows):

```
const int numLEDs = 5;
const int numFrames = 9;
```

Next create the array that holds each of the pin numbers that have LEDs:

```
int ledPins[] = {
  2, 3, 9, 10, 11};
```

Then type out the two-dimensional array of the animation:

```
int frames[numFrames][numLEDs] =
{
  {1, 0, 0, 0, 0},
  {0, 1, 0, 0, 0},
  {0, 0, 1, 0, 0},
  {0, 0, 0, 1, 0},
  {0, 0, 0, 0, 1},
  {0, 0, 0, 1, 0},
  {0, 0, 1, 0, 0},
  {0, 1, 0, 0, 0},
  {1, 0, 0, 0, 0}};
```

Add the following lines of code inside `setup()` to set the pin mode for each LED pin:

```
int i;
for(int i=0; i<numLEDs; i++) {
  pinMode( ledPins[i], OUTPUT);
```

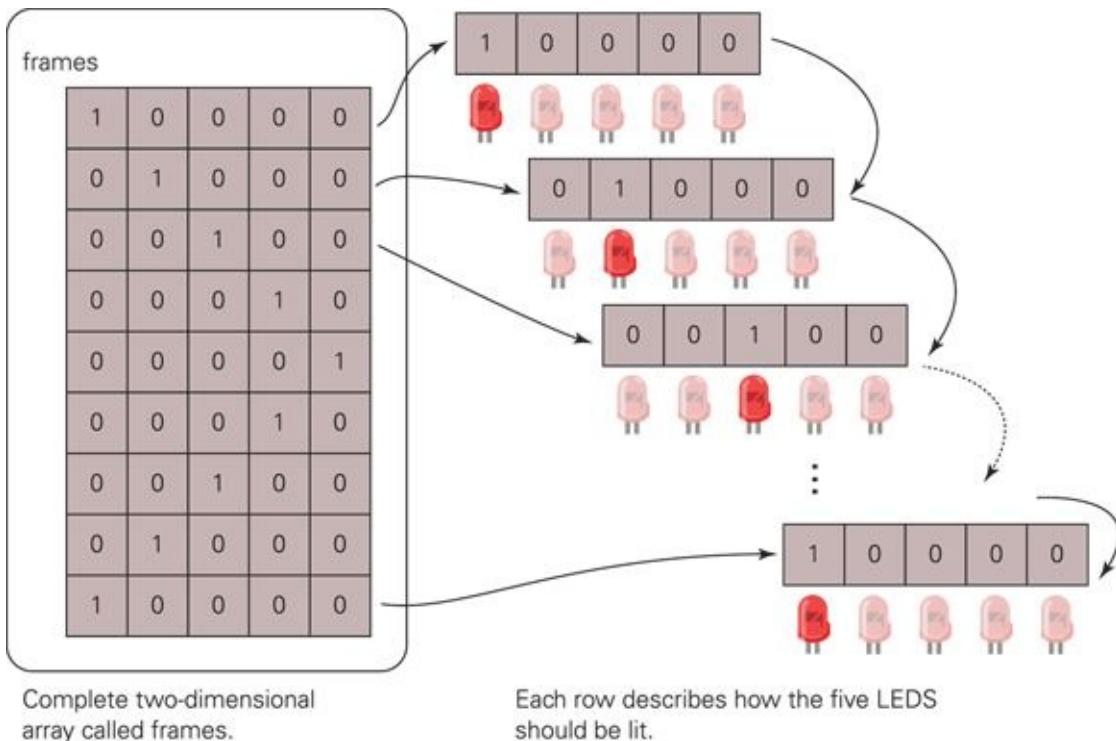
Inside `loop()` is where you start to see unfamiliar code! Add the following code to your sketch, and then you can go over it in more detail:

```
// variables to keep track of current frame and LED
int frame;
int led;
for( frame=0; frame<numFrames; frame++ ) {
  // iterate through each frame stored in a row
  for( led=0; led<numLEDs; led++ ) {
    // turn on or off the each LED in the frame
    digitalWrite( ledPins[led], frames[frame][led] );
  }
  delay(300); // pause between each frame
}
```

You have used `for` loops before, to go through each item in a list one by one. Because there are two dimensions to the `frames` array (you can think of it as a list of lists), two `for` loops are needed.

The first `for` loop goes through each row, which holds a frame of data describing what all the LEDs should do. The second `for` loop goes through that frame and sets the LED to be on or off. After the second `for` loop has turned each of the LEDs on or off, the sketch

pauses before continuing to the next frame. If it didn't do this, the animation would be too fast to see! [Figure 8-11](#) shows how the array is iterated over, row by row.



[Figure 8-11](#) Iterating over frames of an animation stored in a two-dimensional array

The following code is the full sketch. Build the circuit and upload the sketch to see the animation in motion:

```
const int numLEDs = 5;
const int numFrames = 9;

// pins that have LEDs
int ledPins[] = {
  2, 3, 9, 10, 11};

// frames of the animation
int frames[numFrames][numLEDs] =
{
  {1, 0, 0, 0, 0},
  {0, 1, 0, 0, 0},
  {0, 0, 1, 0, 0},
  {0, 0, 0, 1, 0},
  {0, 0, 0, 0, 1},
  {0, 0, 0, 1, 0},
  {0, 0, 1, 0, 0},
  {0, 1, 0, 0, 0},
  {1, 0, 0, 0, 0}};

void setup()
{
  // set pin modes to OUTPUT
  int i;
  for(int i=0; i<numLEDs; i++) {
    pinMode( ledPins[i], OUTPUT);
```

```
    }

void loop()
{
    // variables to keep track of current frame and LED
    int frame;
    int led;
    for( frame=0; frame<numFrames; frame++ ) {
        // iterate through each frame stored in a row
        for( led=0; led<numLEDs; led++ ) {
            // turn on or off the each LED in the frame
            digitalWrite( ledPins[led], frames[frame][led] );
        }
        delay(300); // pause between each frame
    }
}
```

# Passing Data Between Functions

If you want to do other things in `loop()` besides running the animation, it's useful to put the animation code into its own function. The next example shows you how do that.

Create a function called `displayAnimation()` and cut and paste the animation code from `loop()` into the function:

```
void displayAnimation() {  
    // variables to keep track of current frame and LED  
    int frame;  
    int led;  
    for( frame=0; frame<numFrames; frame++ ) {  
        // iterate through each frame stored in a row  
        for( led=0; led<numLEDs; led++ ) {  
            // turn on or off the each LED in the frame  
            digitalWrite( ledPins[led], frames[led][frame] );  
        }  
        delay(300); // pause between each frame  
    }  
}
```

Remember that you now need to call your new function from `loop()` or it will never run the animation. The new function `displayAnimation()` should be after `loop()` and `loop()` should look like the following

```
void loop()  
{  
    displayAnimation();  
}
```

Now that the code controlling the animation is nicely contained within one function, you can take advantage of how functions work. You can pass data to the function and then have the function change what it does according to that data.

In order to pass data, you need to create an argument. This is done in the same line of code where you give the function a name. You also give your argument a name and say what data type it will have.

Change your function to the following:

```
void displayAnimation(int animationSpeed) {
```

The `displayAnimation()` function now has one argument that is an `int`; this is `animationSpeed`. The function now needs to do something with `animationSpeed`. It will control the length of time the animation pauses between each frame. Use this variable to set the `delay()` after each frame.

Change the line of code that sets the delay in the function to the following:

```
delay(animationSpeed); // pause between each frame
```

Now you can call the function `displayAnimation()` with different values for the argument, to play the animation at different speeds! The following code is the full sketch. Upload it to your Lilypad Arduino USB with five Lilypad LEDs connected and watch the

animation described by the two-dimensional array:

```
// const means the value won't change
const int numLEDs = 5;
const int numFrames = 9;

// pins that have LEDs
int ledPins[] = {
  2, 3, 9, 10, 11};

// frames of the animation
int frames[numFrames][numLEDs] =
{
  {1, 0, 0, 0, 0},
  {0, 1, 0, 0, 0},
  {0, 0, 1, 0, 0},
  {0, 0, 0, 1, 0},
  {0, 0, 0, 0, 1},
  {0, 0, 0, 1, 0},
  {0, 0, 1, 0, 0},
  {0, 1, 0, 0, 0},
  {1, 0, 0, 0, 0}};

void setup()
{
  // set pin modes to OUTPUT
  int i;
  for(int i=0; i<numLEDs; i++) {
    pinMode( ledPins[i], OUTPUT);
  }
}

void loop()
{
  displayAnimation( 100 );
  displayAnimation( 500 );
  displayAnimation( 1000 );
}

void displayAnimation(int animationSpeed) {
// variables to keep track of current frame and LED
  int frame;
  int led;
  for( frame=0; frame<numFrames; frame++ ) {
    // iterate through each frame stored in a row
    for( led=0; led<numLEDs; led++ ) {
      // turn on or off the each LED in the frame
      digitalWrite( ledPins[led], frames[frame][led] );
    }
    delay(animationSpeed); // pause between each frame
  }
}
```

# CHALLENGE



Build the circuit for the LED animation shown in the section headed “Passing Data Between Functions.” Modify the code in the sketch to use a **for** loop to cycle through speeds for the animation from 50 to 500.

Remember that you can increment in steps higher than one, for example by creating the following **for** loop:

```
int i;  
for( i=0; i<100; i+=5) {  
}  
}
```

# Building a POV Hoodie

Have you ever waved a sparkler around and noticed that it seemed to leave a trail of light? Or maybe you've seen toys with a set of LEDs that display a word or image when you move them quickly. Both are examples of persistence of vision (POV) where your eyes and brain keep seeing a light path after the light source is moved or turned off. In the case of the sparkler, the glowing sparkler moves but your brain still sees where it used to be, so it looks like it's leaving a trail. With the toys, the LEDs are blinking on and off quickly in a pattern. Your brain puts all the different patterns together when the toy is moved and interprets the patterns as a word or image.

Excitingly, you can make your own POV display with the Lilypad Arduino USB and, because you are using a Lilypad, you can sew the circuit into clothes and wear it. You can then program your Lilypad to display a secret message that can be seen by taking a long exposure photograph (see [Figure 8-12](#)). Sewing the circuit on the shoulder of a hoodie works well, but you can sew it on any piece of clothing!

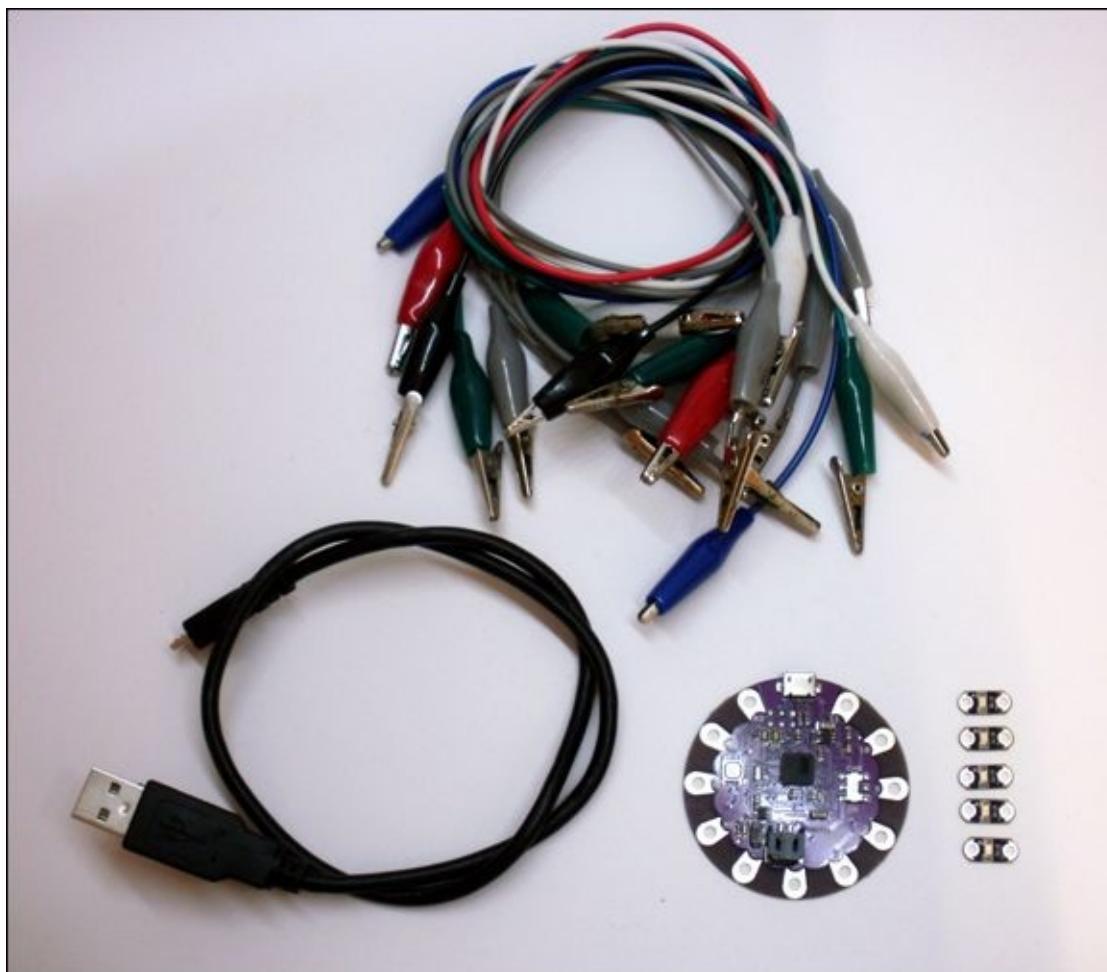


[Figure 8-12](#) Persistence-of-vision hoodie

## What You Need

You need the following items to build a POV hoodie. [Figure 8-13](#) shows the electronic components that you need:

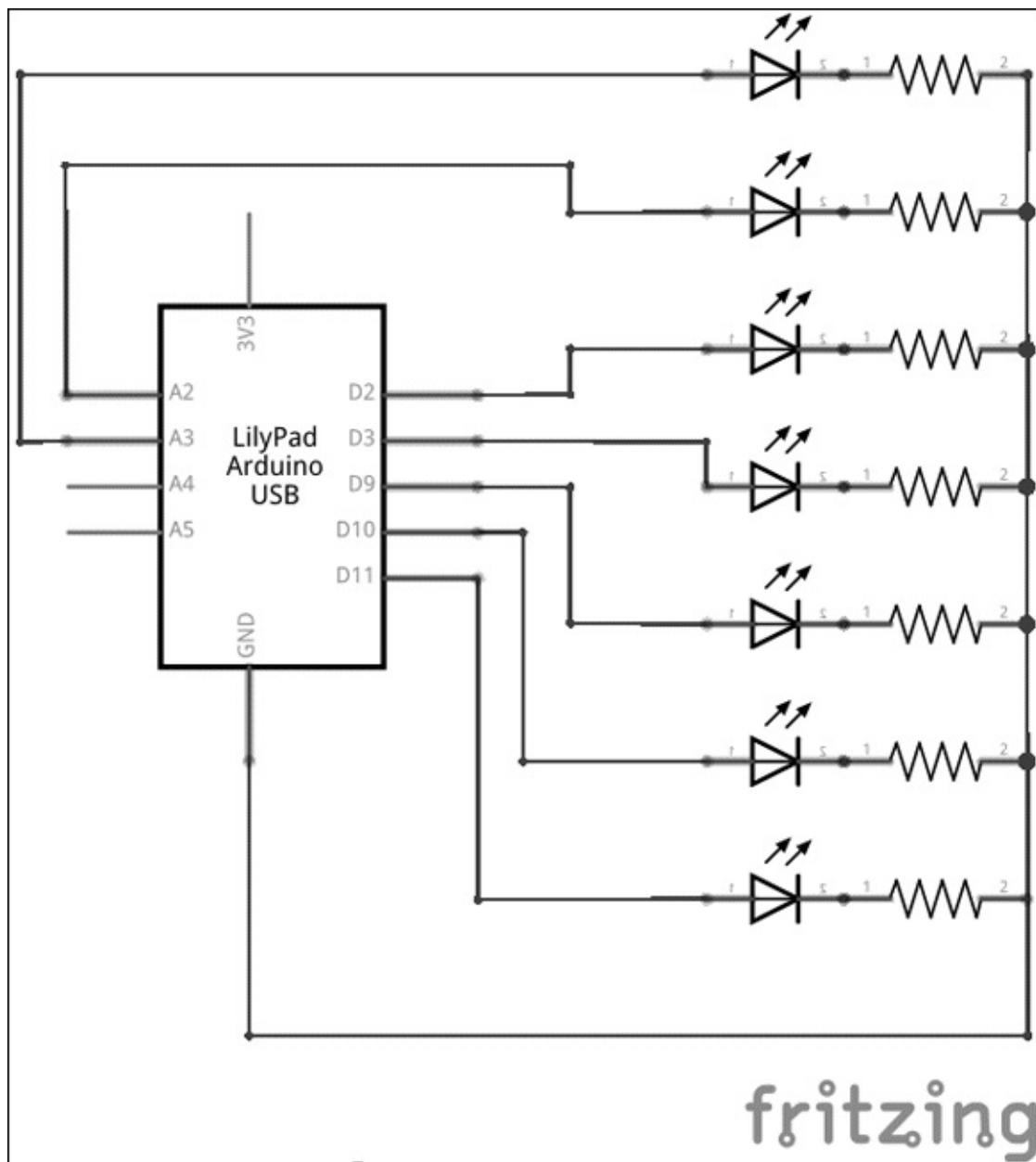
- A computer
- A Lilypad Arduino USB
- A USB micro cable
- A breadboard
- 14 alligator clips
- 7 Lilypad LEDs (or 7 LEDs and 7  $68\Omega$  or  $100\Omega$  resistors if not using Lilypad LEDs)
- A lithium ion polymer (LiPo) battery
- Some normal sewing thread
- Some conductive thread
- A hoodie
- A sewing needle
- Scissors
- Some white PVA glue
- Pliers (if not using Lilypad LEDs)



**Figure 8-13** The electronic components you need for the POV hoodie

## Understanding The Circuit

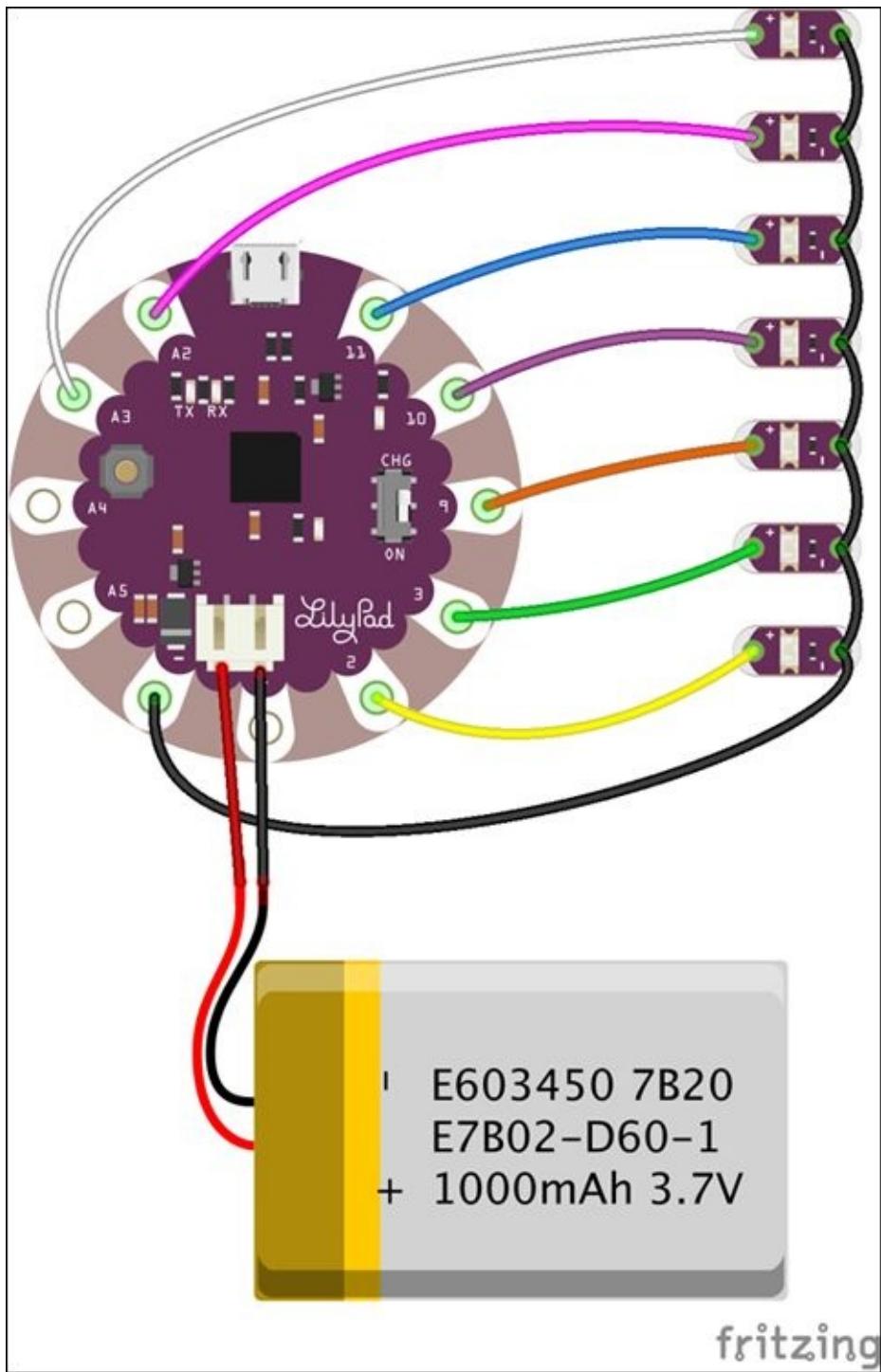
The circuit for the POV hoodie is similar to the LED animation circuit you made earlier in this adventure. The only change is that you use seven LEDs instead of five. [Figure 8-14](#) shows the circuit schematic for the POV hoodie.



[Figure 8-14](#) Circuit schematic for the POV hoodie

## Prototyping with Alligator Clips

The POV hoodie is a soft circuit, so you should prototype the circuit with alligator clips because the Lilypad Arduino USB and Lilypad LEDs don't fit into a breadboard. Build the circuit shown in [Figure 8-15](#).



**Figure 8-15** Prototyping the circuit with alligator clips

Start by connecting each pin to the positive side of one of the Lilypad LEDs. Then connect all the negative sides of the Lilypad LEDs to each other. Connect the negative side of the LED connected to Pin 2 to the GND pad of the Lilypad Arduino USB.



If you don't have Lilypad LEDs, replace them with current-limiting resistors and normal LEDs.

## Charging the Battery

The Lilypad Arduino USB includes a connector for a rechargeable lithium ion polymer

(LiPo) battery and even has a charger built in. LiPo batteries are flat, silver batteries. You don't need a big battery for this project. Batteries are measured in milliamp-hours (mAh), which is a measure of how much current they can output for how long. A 500 mAh battery provides a good balance—it's not too big and doesn't need to be charged too frequently—but you can use whatever size battery you can buy easily.

To charge the battery, connect it to the Lilypad Arduino USB and connect the Lilypad to a power source such as a computer with a USB cable. Make sure the switch on the Lilypad board is in the CHG position. The LED above CHG on the board lights up when the battery is charging; the battery stops charging automatically when it's fully charged.



LiPo batteries can be dangerous if you don't take care of them properly. Never leave them on their own while they're charging! It's also best to buy them from a trusted manufacturer like Adafruit or Sparkfun, or one of their distributors. Only buy LiPos that have built-in protection circuitry, and never use a battery that looks swollen or damaged. Adafruit has a nice guide on how to work with LiPo batteries at <https://learn.adafruit.com/li-ion-and-lipoly-batteries>.

## Writing the Code

The code for displaying the POV message is similar to the code for the LED animation earlier in this adventure. The frames for the LEDs are stored in two-dimensional arrays: one for each letter and one for a space. Because there are so many arrays, to help organise the code they are all stored in a header (.h) file. The code is available at [www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

Open the downloaded sketch in the Arduino IDE. The sketch should have two tabs: one named `pov_hoodie` and the other `alphabet.h`.

Click the `pov_hoodie` tab. Change the message in the bold line of code in `loop()` to what you want to be written on the hoodie, and upload it to your Lilypad Arduino USB. Your message can only consist of uppercase letters and spaces.

```
void loop()
{
    String message = "HELLO "; // put message in all caps here
    printText(message);
}
```

Wave your circuit in the air (a dark room and a friend to move with the circuit makes this easier), or take a long-exposure photo to see the message as shown in [Figure 8-16](#).



[Figure 8-16](#) Persistence of vision message captured with a long-exposure photograph

# DIGGING INTO THE CODE



Open the sketch you just downloaded in the Arduino IDE, and click the `alphabet.h` tab to see the code in the header file.

The following is the code for the letter A. If you turn your head sideways, you might be able to see the shape of the letter A written out in 1s:

```
int A[numFrames][numLEDs] = {  
    {1,1,1,1,1,1,0},  
    {0,0,1,0,0,0,1},  
    {0,0,1,0,0,0,1},  
    {0,1,1,0,0,1,1},  
    {1,1,1,1,1,1,0}};
```

In the main sketch, the header file describing all the letters is imported at the very top of the file:

```
#include "alphabet.h"
```

Then there are three variables describing how long the LEDs are turned off between letters, how long they are on when displaying a frame of a letter and what pins have LEDs. Because the Lilypad Arduino USB only has five digital pins, two analog pins are also used:

```
int letterSpace = 6; // time LEDs off between letters  
int dotTime = 3; // time LEDs are on  
  
int ledPins[] = {  
    2, 3, 9, 10, 11, A2, A3};
```

Inside `setup()`, each `pinMode()` is set to be `OUTPUT` and serial communication is started. The serial communication is used for debugging to see what is being output in the Serial Monitor:

```
// set pin modes to OUTPUT  
int i;  
for(int i=0; i<numLEDs; i++) {  
    pinMode( ledPins[i], OUTPUT);  
}  
  
Serial.begin(9600);
```

The `loop()` is kept quite simple as most of the work is done in other functions. The message to be displayed is stored in `message` and then sent to the function `printText()` as an argument. Because the header file only describes how to display capital letters, the message needs to be written only in capital letters or spaces—there should be no lowercase letters or punctuation:

```
String message = "A B C D "; // put message in all caps here  
printText(message);
```

The function `printText()` is a long one, but it is just doing a simple task many times. It reads in the message, letter by letter, and then tells the function `printLetter()` to display that letter in the message. The following code is for only `A`, `B` and `C`, but you can get the idea for how the rest of the alphabet works:

```
for (int i=0; i<text.length(); i++)  
{  
    switch(text[i])  
    {  
        case 'A':  
            printLetter(A);  
        case 'B':  
            printLetter(B);  
        case 'C':  
            printLetter(C);  
    }  
}
```

```
break;
case 'B':
  printLetter(B);
  break;
case 'C':
  printLetter(C);
  break;
```

The function `printLetter()` is just like the `displayAnimation()` function you wrote earlier. It goes frame by frame through the letter that is being displayed. It also prints what it is being sent to the LEDs to the Serial Monitor to help show what is going on. It has an extra `for` loop at the end of the function to pause between each letter; otherwise, it would be hard to read the individual letters and they would all blur together:

```
int frame;
int led;

// print letter
for( frame=0; frame<numFrames; frame++ ) {
  for( led=0; led<numLEDs; led++ ) {
    digitalWrite( ledPins[led], letter[frame][led] );
    Serial.print(letter[frame][led]);
  }
  Serial.println();
  // delay between each column displayed
  delay(dotTime);
}
Serial.println("-----");

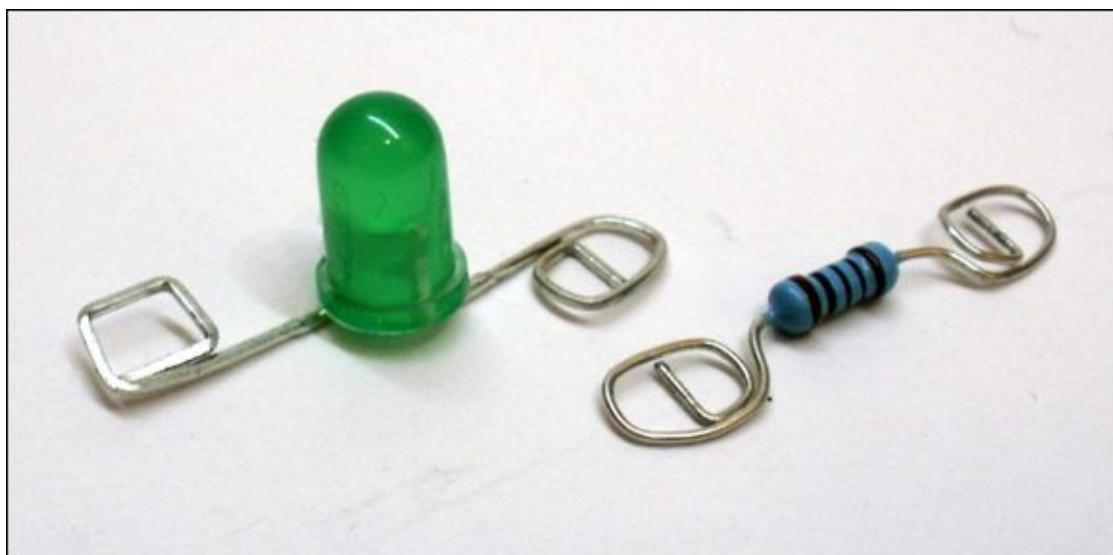
// print space after letter
for( led=0; led<numLEDs; led++ ) {
  digitalWrite( ledPins[led], 0 );
}
// delay for space between letters
delay(letterSpace);
```

## Making the POV Hoodie

After you have built the prototype circuit with your alligator clips and uploaded the code to test that it all works, you are ready to start sewing! Using conductive thread, you are going to stitch the Lilypad Arduino USB pads to each of the LEDs. Remember that conductive thread is just like wire, but it doesn't have the insulating plastic protecting it. That means if any pieces of thread touch each other, they will conduct electricity between them and may short your circuit. Use a separate piece of thread for each connection and make sure they don't accidentally touch another part of the circuit.

## Making Sewable LEDs

If you don't have Lilypad LEDs, you can still use the LEDs you would use in a breadboard. Using pliers, twist the legs of the LEDs and resistors into loops that you can sew through. You can twist the long and short legs of the LEDs into two different shapes to keep track of which is the positive and which is the negative. I like to twist the positive into a circle and the negative into a square (because it reminds me of a negative sign) as shown in [Figure 8-17](#), but you can make any shapes that help you keep track of the legs.



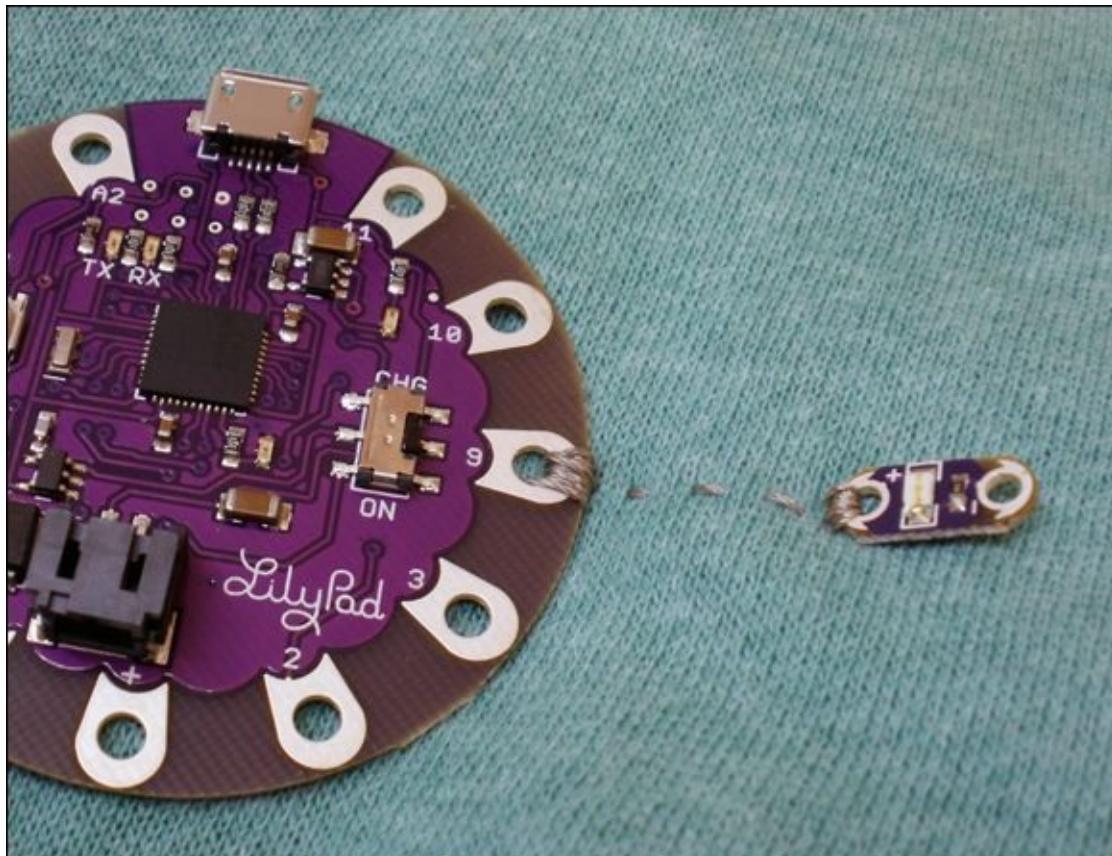
[Figure 8-17](#) Bending the legs of components to make them sewable

## Sewing the Electronics

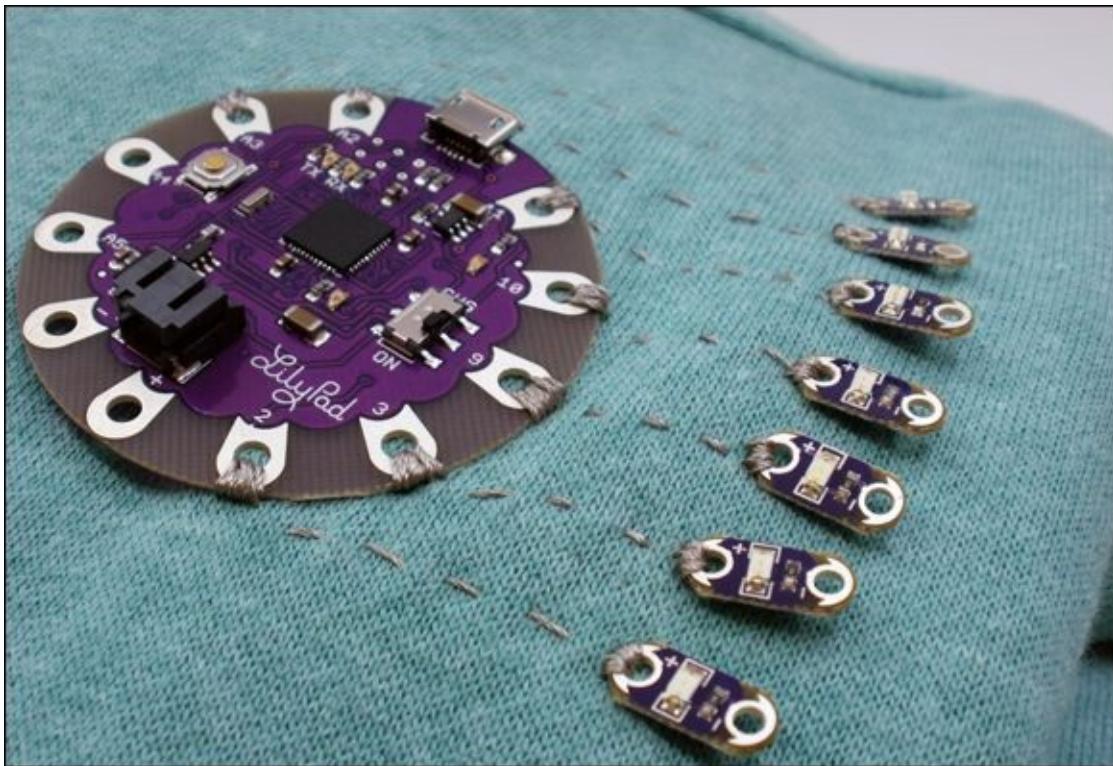
Sew the circuit by going through the following steps:

1. Start by deciding where you want to place your LEDs on the upper right sleeve of your hoodie. You may want to mark the locations of the LEDs with pins or tailor's chalk.
2. Start with the bottom LED. Using conductive thread, stitch Pin 9 to the positive side of the LED. Keep the stitches firm against the fabric of the hoodie and stitch the pads of the Lilypad Arduino USB and the Lilypad LED about five times to secure them to the fabric. Knot and cut the thread. See [Figure 8-18](#) for guidance.

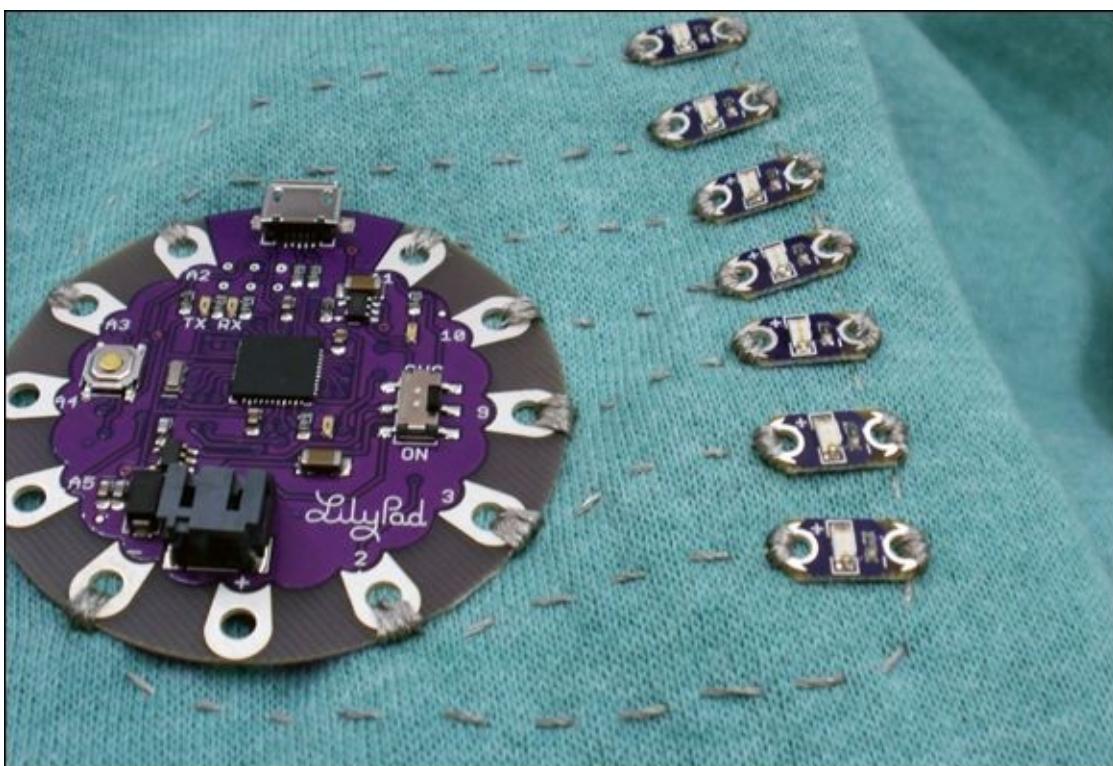
3. Alternatively, if you are using regular LEDs and resistors, stitch the Arduino pad to the LED and then use a separate piece of conductive thread to stitch the LED to the resistor.
4. With a separate piece of conductive thread, stitch Pin 10 to the positive side of the next LED above the one you just sewed. Again, firmly stitch the pads to the fabric and then knot and cut the thread.
5. Continue with each of the remaining LEDs. Take care that the stitched threads never touch each other (see [Figure 8-19](#)).
6. The negative sides of the LEDs (or resistors if you are not using Lilypad LEDs) can all touch each other, so they can be sewn with a single piece of thread. Start at the negative pad of the Lilypad Arduino USB and then stitch the bottom LED. After securing the negative pad of the LED, continue stitching up to the next LED.
7. Repeat step 6, sewing the negative pad of the next LED until you have sewn on all the LEDs as in [Figure 8-20](#).
8. Secure each of the knots with a little white PVA glue. The conductive thread can sometimes unknot itself over time, so the glue helps to prevent this.
9. Using normal thread, stitch around the battery to secure it to the hoodie.



**Figure 8-18** First connections for sewing the Lilypad circuit



**Figure 8-19** Continuing to sew the LEDs into the circuit



**Figure 8-20** The sewn POV circuit



You might not want to sew your Lilypad Arduino USB directly onto your hoodie, especially if you want to use the same board in multiple projects. The Lilypad Arduino SimpleSnap (<http://arduino.cc/en/Main/ArduinoLilyPadSimpleSnap>) comes with female snaps soldered to each of the pads. You can then sew the matching male side of the snaps into the hoodie using conductive thread. Because the snaps are metal, they conduct electricity, but the snaps let you remove the board from the hoodie without harming the hoodie.

You can modify your Lilypad Arduino USB to work the same way by soldering snaps onto the pads yourself. You can find metal snaps at any sewing or craft store.

# Further Adventures with the Lilypad

If you would like to learn more about the Lilypad Arduino USB, visit its page on the Arduino website at <http://arduino.cc/en/Guide/ArduinoLilyPadUSB>.

You can also learn more about using the Lilypad and find more projects at <http://lilypadarduino.org>.

There are a lot books about soft circuits for you to choose from. Here are just a few:

- *Fashioning Technology* by Syuzi Pakhchyan (Maker Media, 2008)
- *Make: Wearable Electronics* by Kate Hartman (Maker Media, 2014)
- *Switch Craft* by Alison Lewis and Fang-Yu Lin (Potter Craft, 2008)
- *Sew Electric* by Leah Buechley, Kanjun Qi and Sona de Boer (HLT Press, 2013)
- *Make: Wearable Electronics* by Kate Hartman (Maker Media, Inc., 2014)

## Arduino Command Quick Reference Table

Command	Description
int[][]	Indicates that the variable will be a two-dimensional array of variables stored in rows and columns.
const	Indicates that the variable will not change its value.



**Achievement Unlocked:** Bright light of Arduino fashion and manipulator of multiple dimensions!

## In the Next Adventure

In the next adventure, you put all your new skills together to create a pinball-inspired game!



## Adventure 9

### The Big Adventure: Building a Marble Maze Game

YOU'VE COME A long way, and now you've reached your final adventure—the biggest yet! In this adventure, you are going to combine digital input, digital output, analogue input and analogue output to create a marble maze game like the one in [Figure 9-1](#). Inspired by retro pinball machines, your game knows when you've scored points and tells you if you've achieved a new high score. It counts down to when the next game begins and keeps track of the time remaining in the game. It even has its own sound effects!



[Figure 9-1](#) A completed big adventure marble maze game

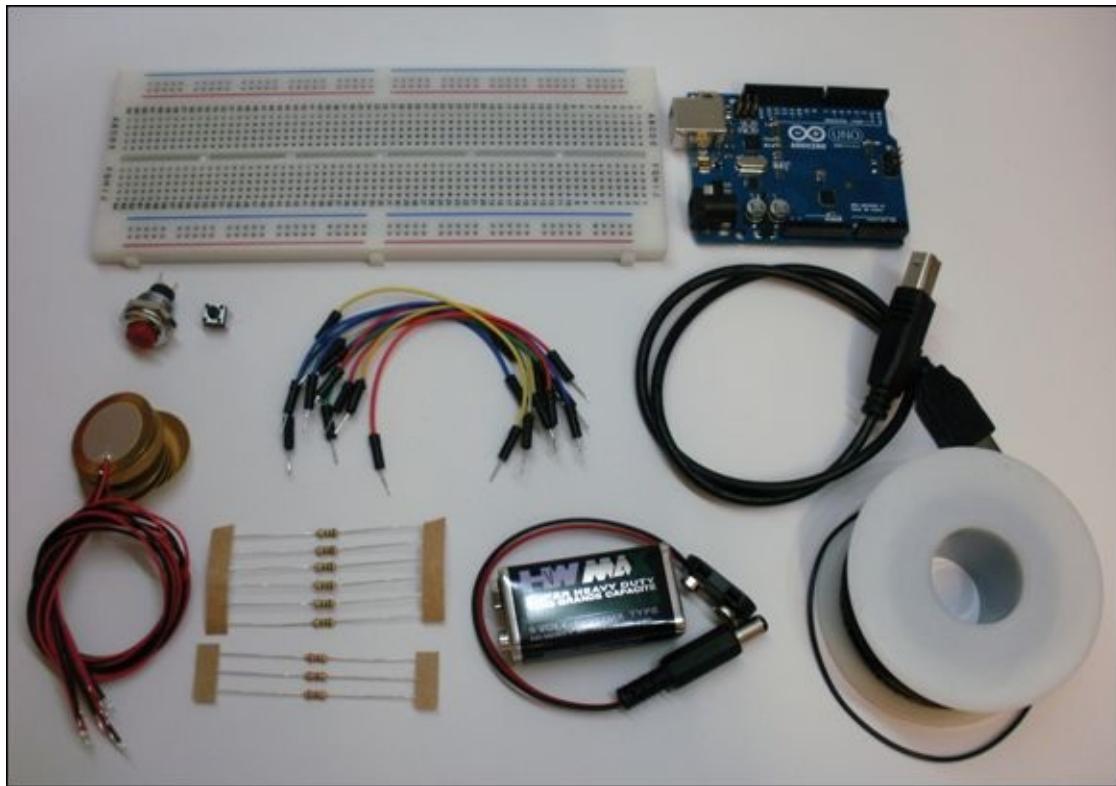
The game brings together a lot of the different skills you've learned from the earlier

adventures, but there are a couple of other things for you to learn before you can build your game. The first is how to use a piezo as a sensor. You've already used a piezo as a speaker but it is a multitalented component and you can also use it to sense vibrations, which is a vital skill for your marble maze game!

# What You Need

You need the following items for your marble maze game. The electronic components that you need are shown in [Figure 9-2](#):

- A computer
- An Arduino Uno
- A USB cable
- A breadboard
- Some jumper wires
- 6 piezos
- 6  $1M\Omega$  resistor
- 1 red LED
- 1 yellow LED
- 1 green LED
- 3  $220\Omega$  resistors
- 1 panel mount push-to-make button
- 1 tactile pushbutton
- 1 9V battery
- 1 9V battery connector
- Some wire
- A marble
- A cardboard box
- Some thick paper or card
- White PVA glue
- Masking tape or duct tape
- Paint or markers
- Scissors or a utility knife
- A soldering iron
- Solder



**Figure 9-2** The electronic components you need to build your maze game

# **Part One: Scoring Points**

In [Adventure 5](#) you were introduced to piezos as actuators that translate a varying voltage into a vibration that you can then hear as a sound wave. But that's not all they can do! Piezos are clever little components that can also translate vibrations into varying voltage.

In this adventure, you use piezos to detect which hole in the maze your marble has fallen through. Each hole is worth a different number of points, so part of your game design is to make the holes that are harder to reach worth more points.

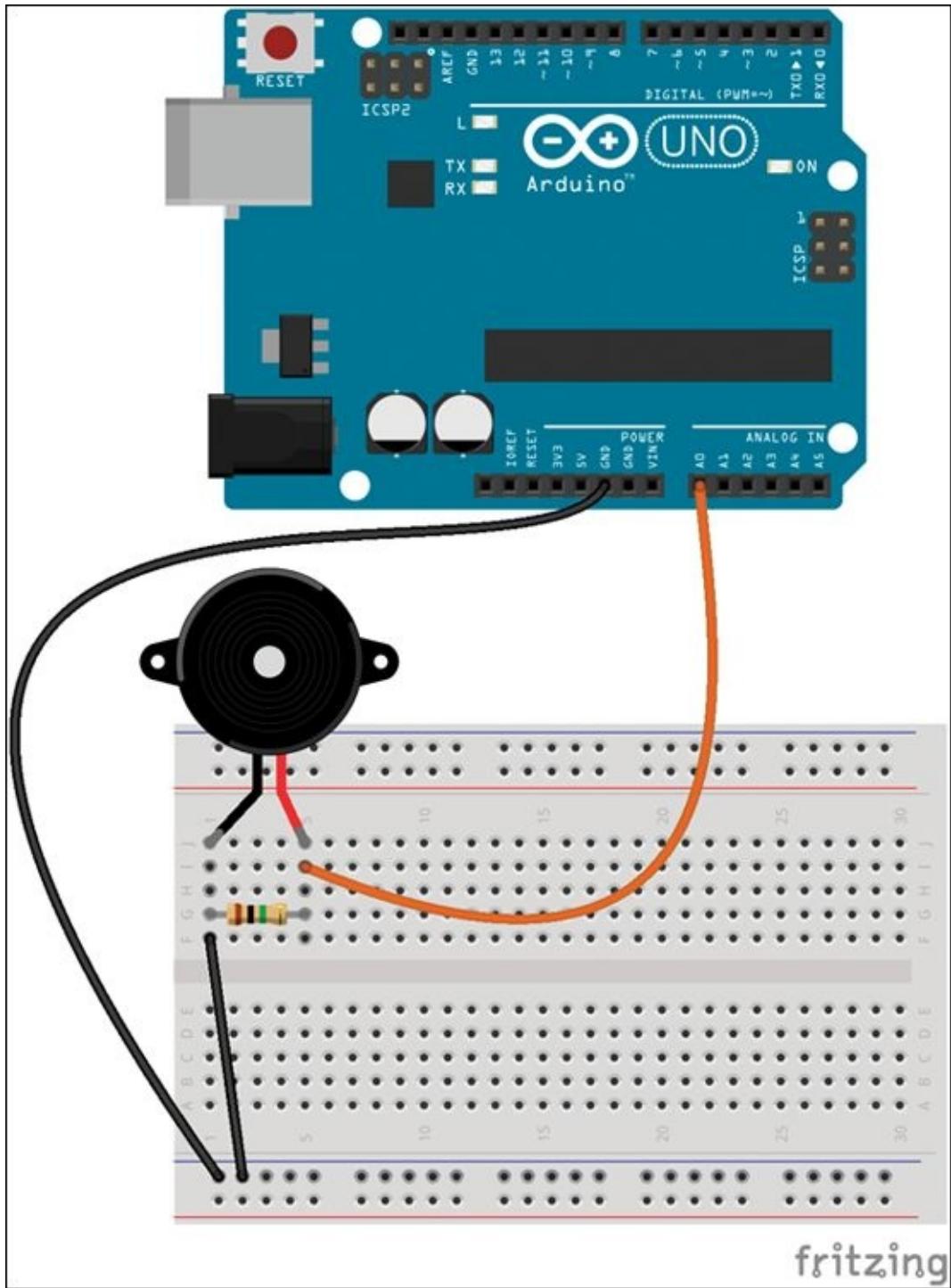
## Sensing Vibrations with Piezos

Piezos have the ability to produce voltage spikes that could damage your Arduino, so you need a really big resistor to help protect your Arduino. A resistor value of  $100M\Omega$  (that's 100 million Ohms) is typically used.

As piezos output a varying voltage, you will want to measure not just **LOW** or **HIGH** voltages but also voltages in between. You will therefore want to use an Analog Pin and **analogRead()** to measure the voltage.

Build the circuit shown in [Figure 9-3](#):

1. Place the red wire of the piezo in any of the short, middle rows of the breadboard and place the black wire in one of the long outside rows.
2. Put one leg of a  $1M\Omega$  resistor in the same short row as the red wire and the other leg in the same long row as the black wire.
3. Use a jumper wire to connect the row with the red wire and resistor leg to Pin A0.
4. Use another jumper wire to connect the row with the black wire and the other resistor leg to **GND**.



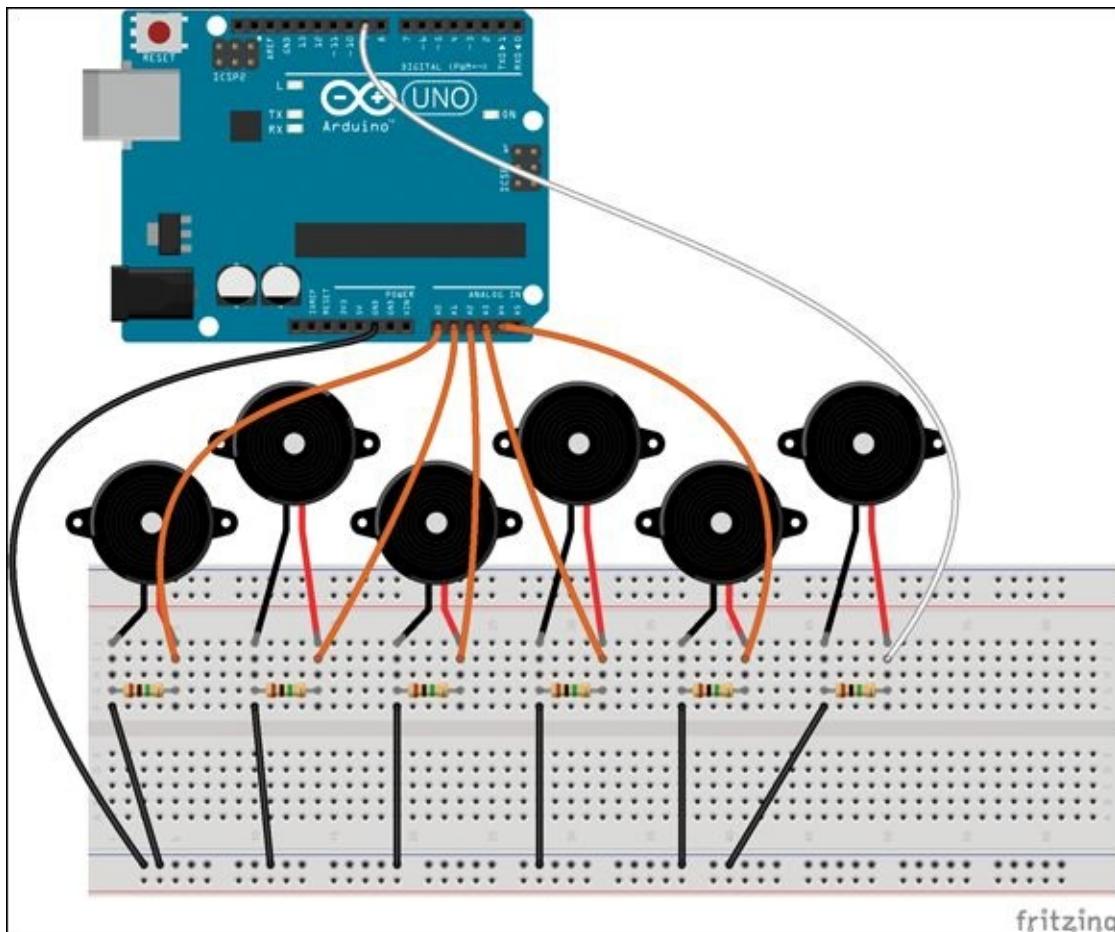
**Figure 9-3** Circuit to use a piezo as a sensor

Launch the Arduino IDE. Go to File  $\Rightarrow$  Examples  $\Rightarrow$  06.Sensor and open the sketch Knock. Upload it to your Arduino and open the Serial Monitor. Try out the sketch by tapping on the piezo. Watch what happens to the LED next to Pin 13 and see what is printed in the Serial Monitor. You should see the LED turn on and off with your knocks on the piezo.

## Setting a Points Threshold

You now have the basics of how to read in a value from a piezo—but you need more than one piezo to make your game challenging! The next step is to add four more piezos so that you have five in total.

On the same breadboard where you have one piezo connected to Pin A0, set up four more piezos in the same way. Connect them to Pins A1, A2, A3 and A4 as shown in [Figure 9-4](#). Ignore the last piezo for now (the one connected to Pin 9).



[Figure 9-4](#) Circuit for five piezos as sensors and one piezo as a speaker

Launch the Arduino IDE and start a new sketch. Begin by creating an empty `setup()` and `loop()`:

```
void setup() {  
}  
void loop() {  
}
```

Because you have five piezos to keep track of, it's best to store them all in an array. At the top of the sketch, add an array that stores all the pins with the piezos as sensors and a variable that stores how many piezos you are using:

```
int pointsPins[] = {  
    A0, A1, A2, A3, A4};
```

```
// number of piezos for points  
int numPointsPins = 5;
```

You don't need to set the pin mode of Analog Pins when you are using them as inputs, so the only line of code to add to your `setup()` is for beginning serial communication:

```
Serial.begin(9600);
```

Inside `loop()`, read in the value of each of the piezos. Because the pins are all stored in an array, you can use a `for` loop. For now you just print those values to the Serial Monitor.

Add the following lines of code to your loop, then upload it to your Arduino Uno and open Serial Monitor in the Arduino IDE:

```
// variables for pins  
int i;  
for( i=0; i<numPointsPins; i++ ) {  
    int currPinValue = analogRead(pointsPins[i]);  
  
    // print which pin in array and value  
    Serial.print("Piezo: ");  
    Serial.print(i);  
    Serial.println(" Value: ");  
    Serial.println(currPinValue);  
}
```

Tap on the piezos and note what values are being generated. Drop the marble you will use in your maze game onto a piezo and watch the values.

Now you need to change your sketch so that a message is only printed when a piezo goes above a threshold value. In other words, a message is only printed when the piezos are triggered with a strong enough force.

At the top of your sketch before `setup()`, with your other variables, add a variable to set the threshold of when a piezo should trigger scoring points. Set it to a value that makes sense from your tests with your marble:

```
int piezoThreshold = 800;
```

Inside of `loop()`, add an `if` statement. The `if` statement checks whether the current piezo being read has gone above the threshold that you set. When the current goes above the threshold, the piezo number and value are printed to Serial Monitor.

```
int i;  
for( i=0; i<numPointsPins; i++ ) {  
    if( currPinValue > piezoThreshold ){  
        // print which pin in array and value  
        Serial.print("Piezo: ");  
        Serial.print(i);  
        Serial.println(" Value: ");  
        Serial.println(currPinValue);  
  
        // pause so that same marble doesn't score twice  
        delay(300);  
    }
```

```
}
```

Upload the changes to your sketch and check that each piezo only prints a message when you drop the marble on it. Here is the sketch as it stands:

```
// variables for pins
int pointsPins[] = {
  A0, A1, A2, A3, A4};

// number of piezos for points
int numPointsPins = 5;

//---setup-----
// runs once when board first powered
// or reset
void setup() {
  // start serial communication
  Serial.begin(9600);
}

//---loop-----
// runs continuously after setup()
void loop() {
  int i;
  for( i=0; i<numPointsPins; i++ ) {
    int currPinValue = analogRead(pointsPins[i]);

    // if above the threshold
    if( currPinValue > piezoThreshold ){
      // print points and new score
      Serial.print("Piezo: ");
      Serial.print(i);
      Serial.println(" Value:");
      Serial.print(currPinValue);

      // pause so that same marble doesn't score twice
      delay(300);
    }
  }
}
```

## Adding Sound Effects

Now it's time to see what that sixth piezo is supposed to do! In the previous section you worked with the five piezos that are acting as sensors; the sixth piezo acts as a speaker. In [Figure 9-4](#), the circuit for the sixth piezo looks just like the circuits for the other five piezos, which might be confusing. The only difference is that it is connected to Pin 9 instead of an Analog Pin.

You use Pin 9 so that you can call `tone()` to make sound through the piezo. However, just because the piezo is being used as a speaker, it doesn't prevent it from potentially producing a voltage spike as a sensor. If the marble hits the speaker piezo, it could produce a voltage spike that could damage your Arduino Uno. So you use the same protection circuit on the speaker piezo.

Go ahead and add the last piezo on Pin 9 as shown in [Figure 9-3](#).

In your code, add a variable at the top before `setup()` to keep track of the speaker:

```
int speakerPin = 9;
```

Right after you have printed your messages in the `for` loop and before the `delay()`, add the following lines of code in bold to play a sound whenever a piezo is triggered above the threshold:

```
// if above the threshold
if( currPinValue > piezoThreshold ){
    // print points and new score
    Serial.print("Piezo: ");
    Serial.print(i);
    Serial.println(" Value:");
    Serial.print(currPinValue);

    // play scoring music
    tone(speakerPin, 659, 300);
    delay(300);

    // pause so that same marble doesn't score twice
    delay(300);
}
```

Upload your sketch to your Arduino Uno and test that you hear a tone whenever a piezo is triggered.

## Keeping Score

Now that you have five piezos that sense when the marble drops on them and another that acts as a speaker, you can start assigning point values to each of the sensors and keep track of your score.

First you need to create a new variable to store your score. At the top of your sketch, with your other variables, add the following line:

```
int currentScore = 0;
```

Inside the `if` statement in the `for` loop in `loop()`, you are going to do three things. First you create a variable that calculates how many points have just been scored and then adds it to current score. Then you print the number of points that have just been scored and the new total current score to the Serial Monitor:

```
if( currPinValue > piezoThreshold ){
    // add points
    int newPoints = (i+1)*10;
    currentScore += newPoints;

    // print points and new score
    Serial.print(" Score! ");
    Serial.print(newPoints);
    Serial.println(" points");
    Serial.print(" Current score: ");
    Serial.print(currentScore);
    Serial.println(" points");

    // play scoring music
    tone(speakerPin, 659, 300);
    delay(300);

    // pause so that same marble doesn't score twice
    delay(300);
}
```

The number of points is determined by adding 1 to the pin in the array that was triggered (the variable `i`) and then multiplying that number by 10. For example, the piezo on Pin A3 is item 3 in the array, so the score for triggering that piezo is  $(3+1)*10$ , or 40 points.

The following code is the full sketch with the changes in bold. Upload it to your Arduino Uno and open the Serial Monitor. Check that the sound effects and points messages all work as you would expect:

```
// variables for pins
int pointsPins[] = {
    A0, A1, A2, A3, A4};

int speakerPin = 9;

// number of piezos for points
int numPointsPins = 5;

// when points triggered
```

```

int piezoThreshold = 800;

int currentScore = 0;

void setup() {
  // start serial communication
  Serial.begin(9600);
}

void loop() {
  int i;
  for( i=0; i<numPointsPins; i++ ) {
    int currPinValue = analogRead(pointsPins[i]);

    // if above the threshold
    if( currPinValue > piezoThreshold ){
      // add points
      int newPoints = (i+1)*10;
      currentScore += newPoints;

      // print points and new score
      Serial.print(" Score! ");
      Serial.print(newPoints);
      Serial.println(" points");
      Serial.print(" Current score: ");
      Serial.print(currentScore);
      Serial.println(" points");

      // play scoring music
      tone(speakerPin, 659, 300);
      delay(300);

      // pause so that same marble doesn't score twice
      delay(300);
    }
  }
}

```



You can download all the sketches from the companion site at  
[www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

## Part Two: Designing Your Maze Game

At last you're ready to start working on your maze! First, you need to find a box that you can turn into your game. The exact size isn't important—it just needs to be big enough to hold all the sensors and the Arduino Uno, but it's best not to have a box that's so big that you can't easily hold it in your hands. A box that is approximately  $10 \times 13 \times 3$  inches ( $26 \times 33 \times 8$  cm) works well.

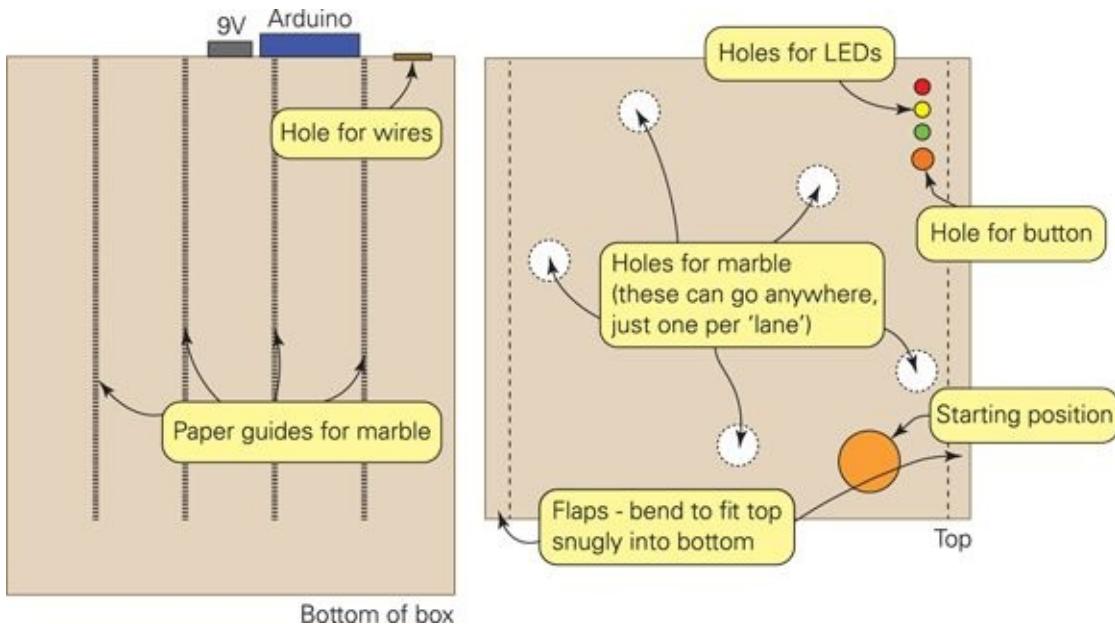


If you can only find a box that is the right length and width but is too tall, use scissors or a utility knife to make it shorter. You may need help from an adult if it's tough to cut!

## Drawing the Maze

When you have your box ready, make a sketch of what you want your maze to look like. Here are a few rules to keep in mind when you do this (see [Figure 9-5](#)):

- Make your game into five columns and decide where in each column you will place the holes. There will only be one piezo for each column, so you can only have one hole per column. Decide how many points you want each hole to be worth, with one worth 10 points, one worth 20, one worth 30, one worth 40 and one worth 50. Make the most difficult hole worth the most and the easiest one worth the least.
- It's important to leave an empty space at the bottom of the maze. This is where the marble rolls out when you have scored. You need to leave enough space that you can easily retrieve the marble and play again.
- Reserve a section in the upper-right corner for three LEDs and the button. These tell you when to start, when your time is up and whether you have achieved a new high score. Leave enough room for the LEDs, button and labels. Approximately  $2 \times 3$  inches ( $5 \times 8$  cm) should be enough.
- Decide where you will place the marble at the start of each game.



[Figure 9-5](#) Guidelines for designing your maze

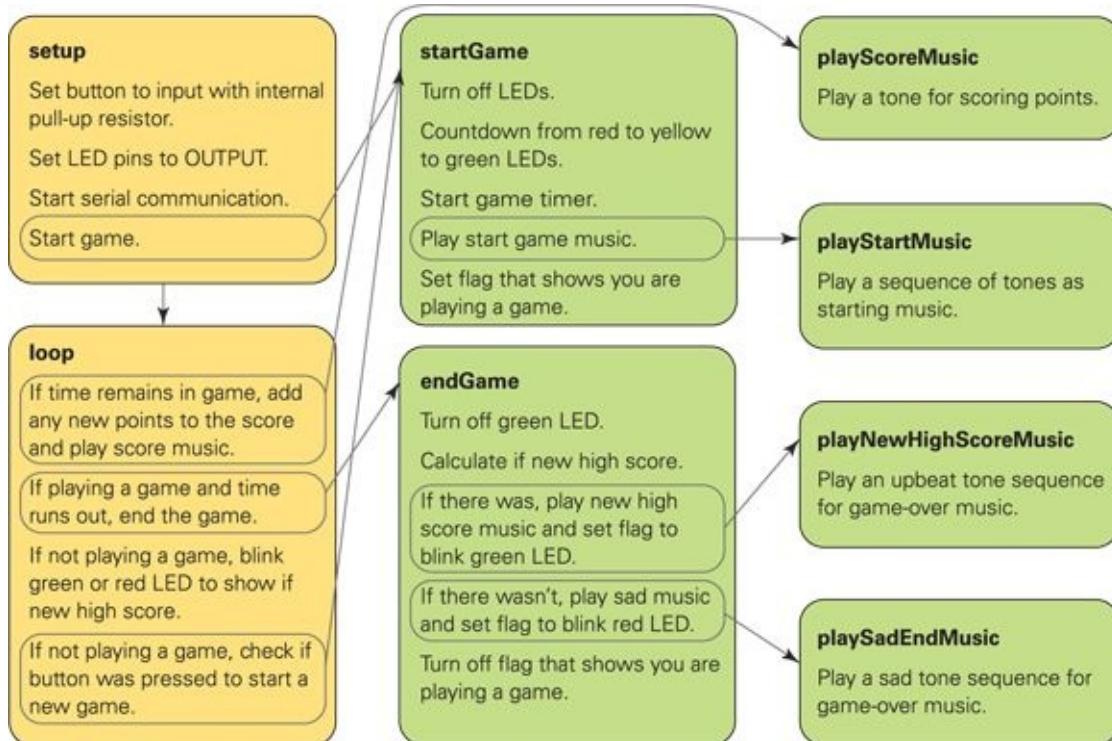
After you have laid out where the holes and other components will be located, you are ready to design your maze! Draw the path of your maze on a piece of paper. Don't lose that piece of paper—you'll need it later when you actually build the maze.

# Designing the Game Code

Here's how the marble maze game is played:

1. The player presses the start button. The red LED lights up, then the yellow LED and finally the green LED. The starting music plays and the game starts. The player places the marble in the starting position on the maze.
2. The player tries to roll the marble along the maze and into the different holes to score points.
3. A tone plays whenever points are scored.
4. Each time the player scores, the marble rolls to the bottom of the box. The player places the marble back at the starting position and tries to score again.
5. The steps are repeated until the time runs out.
6. If the player has just achieved a new high score, a tune plays and the green LED blinks. If they didn't achieve a new high score, a different tune plays and the red LED blinks.
7. If the Arduino Uno is reset or turned off, the high score is cleared. Otherwise players can play more rounds and try to beat the previous high score.

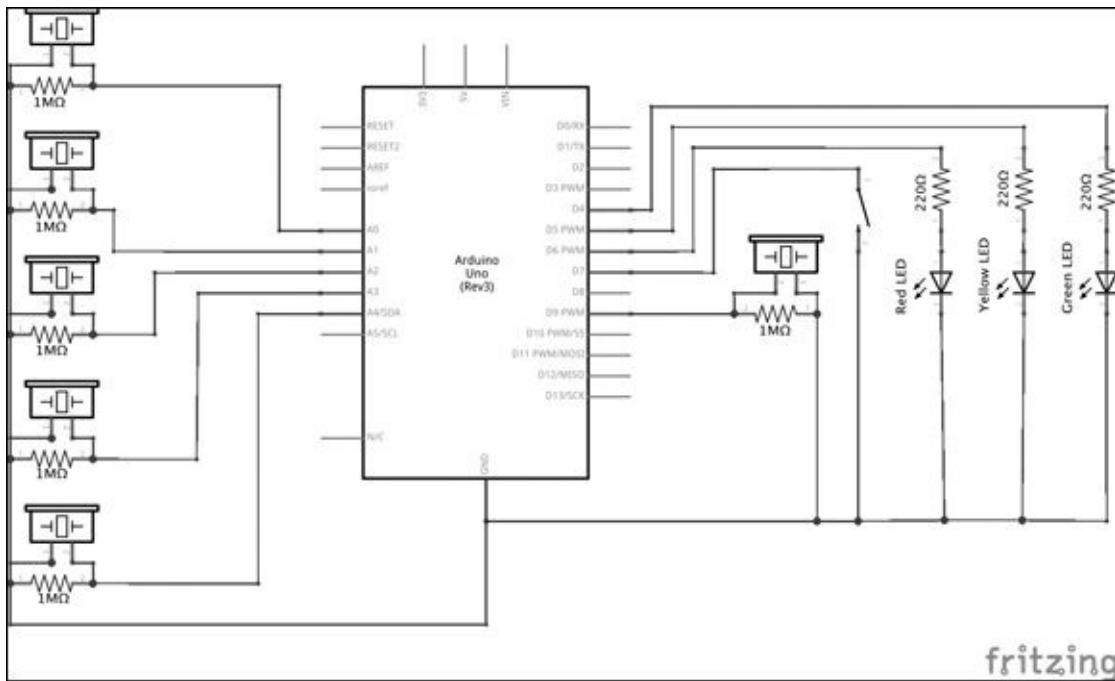
All of these steps can be broken down into functions for the code. Then those functions can be called at the right time in the game. [Figure 9-6](#) shows how all the functions interact in the Arduino sketch that you will write later.



[Figure 9-6](#) How the code works when a game is played

## Prototyping the Circuit

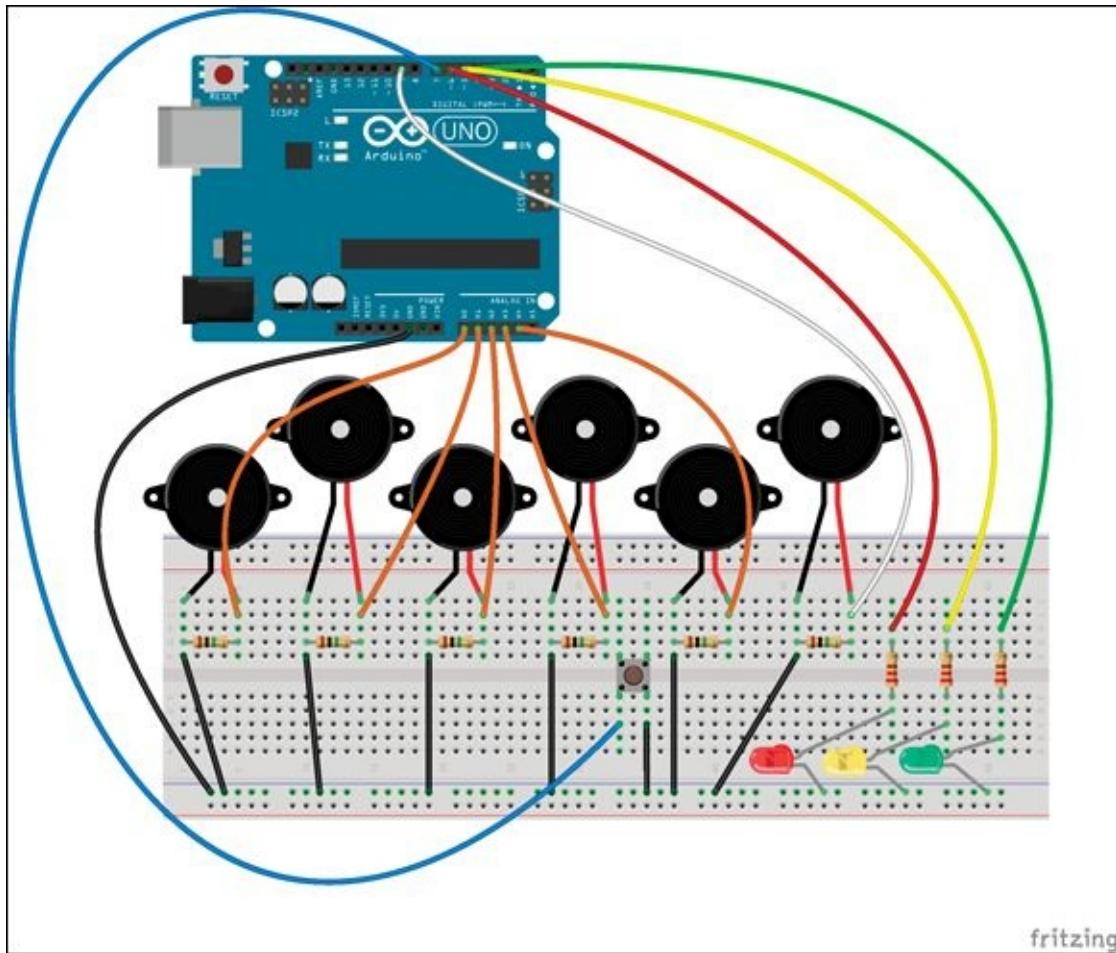
You have already built most of the circuit for the maze game—you have built the circuit for the five sensor piezos and the speaker piezo. The remaining components are the three LEDs and button. [Figure 9-7](#) shows the circuit schematic for the full maze game.



[Figure 9-7](#) Circuit schematic of the maze game

You should test your circuit on a breadboard before building your full maze game—but you know that by now! Build the circuit now (shown in [Figure 9-8](#)):

1. Start with the piezo circuits that you have already built.
2. Add three LEDs to the breadboard—one red, one yellow and one green. Connect the negative legs to the long row connected to **GND**.
3. Place a  $220\Omega$  current-limiting resistor before each LED.
4. Use a jumper wire to connect the resistor before the red LED to Pin 6, the resistor before the yellow LED to Pin 5 and the resistor before the green LED to Pin 4.
5. Place your tactile pushbutton across the gap in the middle of the board. Use a jumper wire to connect one side of your button to the long row connected to GND. Connect the other side to Pin 7.



**Figure 9-8** Maze game prototype circuit on a breadboard

## **Part Three: Writing the Code**

You're about to write the longest Arduino sketch of all the adventures. Don't worry—you are ready! You have seen all the different parts of the code before. You're about to put it together into a more complex sketch, but the sketch is just made up of smaller chunks of code. You start with the piezo-scoring sketch that you wrote earlier in this adventure and add new features, little by little. You should test what you have written after you add each new feature; I'll tell you when you should do that.

## Starting the Game

The LEDs need to show the countdown to begin whenever a new game is started. You can also print serial messages that show extra information if your Arduino Uno is connected to your computer. The first lines of code are the variables for the LED pins. At the top of your sketch add the following lines:

```
int greenLED = 4;  
int yellowLED = 5;  
int redLED = 6;
```

To help organise your sketch, you create a function that performs all the tasks that need to be done when a new game begins. Add the following code to the end of your sketch `afterloop()`:

```
void startGame() {  
    // make sure all LEDs start off  
    digitalWrite(redLED, LOW);  
    digitalWrite(yellowLED, LOW);  
    digitalWrite(greenLED, LOW);  
  
    Serial.println("****NEW GAME****");  
    Serial.print("Starting game in...");  
    // turn on red LED  
    digitalWrite(redLED, HIGH);  
    Serial.print("ready...");  
    delay(1000);  
    // turn off red LED  
    digitalWrite(redLED, LOW);  
  
    // turn on yellow LED  
    digitalWrite(yellowLED, HIGH);  
    Serial.print("set...");  
    delay(1000);  
    // turn off yellow LED  
    digitalWrite(yellowLED, LOW);  
  
    Serial.println("go!");  
    // turn on green LED  
    digitalWrite(greenLED, HIGH);  
  
    // reset score  
    currentScore = 0;  
}
```

The `startGame()` function makes sure all the LEDs are off at the start of the game and then turns them on one by one. It also prints countdown messages. At the end of the function the variable holding the current score, `currentScore`, is reset to `0`.

In `setup()`, set the `pinMode()` to `OUTPUT` for each of the LEDs and call the function you just wrote so that a new game starts automatically when you turn on the Arduino Uno. The new code you should add is in bold:

```
void setup() {
```

```
// set up pin mode for button  
pinMode(buttonPin, INPUT_PULLUP);  
  
// set up pin modes for LEDs  
pinMode(greenLED, OUTPUT);  
pinMode(yellowLED, OUTPUT);  
pinMode(redLED, OUTPUT);  
  
// start serial communication  
Serial.begin(9600);  
  
// start countdown to start  
startGame();  
}
```

Upload your sketch, open Serial Monitor and test that it works. Your LEDs should light up in the right order and countdown messages should appear in Serial Monitor.

## Ending the Game

To turn your marble maze into a game, you need a timer so the player can score as many points as possible within a given time.

To make a timer in your code, begin by adding the following variables to the top of your sketch. You don't use all of them right away but they will all be used in your completed game:

```
int buttonPin = 7;  
int maximumTime = 10000;  
long gameStartTime;  
boolean playingGame = false;
```

The first variable you use is `gameStartTime`. You might notice that it isn't an `int`—it's a `long`. You first encountered `long` in [Adventure 6](#). A `long` can store a bigger number than an `int`, which was needed for the capacitive sensing library that you used to make your crystal ball. Here the `gameStartTime` variable stores the time counted in the number of milliseconds since the Arduino sketch began. That could be a very big number, so the variable should be a `long` instead of an `int`.

In order to store the time that a game starts, you use a new function: `millis()`. This is a built-in function for Arduino so you don't need to import a library to use it. It returns the current number of milliseconds that have passed since the sketch started. You want to save this number so that you can check and see how much time has gone by since a game was started.

Add the following line of code to the very end of your `startGame()` function:

```
gameStartTime = millis();
```

In `loop()`, you then check and see if time has run out. The variable `maximumTime` stores how long a game can run. It's currently set to 10,000 milliseconds (10 seconds), but you can make that shorter or longer.

Add the following `if` statement to the top of `loop()`. All the code you have written so far in your loop that reads in from the piezos and keeps track of the score should go inside the `if` statement. That way, new points can only be scored if the time hasn't run out:

```
if( (millis() - gameStartTime) < maximumTime ){  
    // code you have already written that keeps track of points being  
    // scored  
}
```

The `if` statement checks that the time since the game was started is still less than `maximumTime`.

The variable `playingGame` is one of the variables you just added to the top of your sketch. It has a Boolean data type. That means the variable can only be equal to `true` or `false`. You are using the variable as a **flag**. Whenever a game is being played, the variable is set to `true`, and when a game isn't being played the variable is set to `false`. You can then make decisions in code based on whether a game is being played.



A **flag** is a variable in a program that keeps track of the state of some other part of the code. It is usually a Boolean.

In `startGame()`, add a line at the end of the function that sets the flag `playingGame` to `true`:

```
playingGame = true;
```

Now create a new function at the end of your sketch called `endGame()`. This function is called when the time has run out. It then sets the `playingGame` flag to `false`:

```
void endGame() {  
    Serial.println("Game Over!");  
    Serial.print("Score: ");  
    Serial.println(currentScore);  
  
    // set flag that not currently playing a game  
    playingGame = false;  
}
```

The `if` statement checks if a game is within the time limit. If that isn't true, you want something other than scoring points to occur. If the game is being played and the time has run out, you need to end the game. If the time has run out and the game isn't being played, then you need to display whether a new high score was achieved. You code the new high score part in the next section. For now, focus on ending the game.

In the code that runs only when the game isn't being played or the time has run out, you use an `else` statement. An `else` statement contains the code that should be run only when the conditions in the `if` statement are `false`. It won't run that code at any other time, and it has to be paired with an `if`.

Inside your loop, after the closing `}` of your `if` statement, add the following lines:

```
else{  
    // else if playing a game but time has run out  
    if( playingGame ){  
        // end the game  
        endGame();  
    }  
}
```

The code in the `else` statement is run only if the time has run out. The `if` statement inside the `else` statement checks if the game flag is still set to `true`. If it is, then the game is ended by calling `endGame`.

Upload the sketch to your Arduino Uno and test that it works. It should do everything that it has been doing so far—starting a game and keeping track of the score. Now it should also end the game after `maximumTime`, which is set to 10 seconds.

## Starting a New Game

Now that your game ends after a time limit, you need a way to start a new game to try and beat your score! The next step is to add a button that starts a new game. You already have the button in the circuit on your breadboard, so you only need to add the code.

Inside the `else` statement you just added in the previous section, add another `else` statement to the `if` statement that is checking whether the `playingGame` flag is `true`:

```
else{
    int buttonValue = digitalRead( buttonPin );
    if( buttonValue == 0) {
        // button is pressed, start new game
        startGame();
    }
}
```

Now if the `playGame` flag is set to `false`, the button is checked to see if it is being pressed. If it is, then a new game is started by calling the `startGame()` function.

The `loop()` now has the following code. Upload your sketch to your Arduino Uno and test that everything works. A new game should start with its countdown LEDs when you first start the Arduino Uno, and it should end after 10 seconds. If you push the button, a new round of the game begins:

```
void loop() {
    // if playing a game and still within time
    if( (millis() - gameStartTime) < maximumTime ){
        // read in each points pin
        int i;
        for( i=0; i<numPointsPins; i++ ) {
            int currPinValue = analogRead(pointsPins[i]);

            // if above the threshold
            if( currPinValue > piezoThreshold ){
                // add points
                int newPoints = (i+1)*10;
                currentScore += newPoints;

                // print points and new score
                Serial.print(" Score! ");
                Serial.print(newPoints);
                Serial.println(" points");
                Serial.print(" Current score: ");
                Serial.print(currentScore);
                Serial.println(" points");

                // pause so that same marble doesn't score twice
                delay(300);
            }
        }
    }
    else{
        // else if playing a game but time has run out
        if( playingGame ){

    
```

```
// end the game
endGame();
}
// else if not playing a game
else{
    // check if button has been pressed to start new game
    int buttonValue = digitalRead( buttonPin );
    if( buttonValue == 0) {
        // button is pressed, start new game
        startGame();
    }
}
}
```

By now the `loop()` is getting a little complicated! [Figure 9-9](#) illustrates how it all works. It also shows what you will be adding next—keeping track of the high score.

**if** It has been less then 10 seconds since the game started...  
then go through the piezos and read in their values.

**if** A piezo sensor is above the threshold...  
then add points to the score, play the score music and wait 300 ms.

**else**  
Read in the value from the button.

**if** The button was pressed...  
then start new game.

**if** A game is still being played...  
then end the game.

**else**  
**if** There was a new high score...  
then flash the green LED  
**else**  
then flash the red LED.

**Figure 9-9** How a `loop()` works

# Keeping Track of the High Score

At the top of your sketch, add three more variables:

```
boolean newHighScore = false;  
int currentScore;  
int highScore = 0;
```

You have added one more flag: `newHighScore`. This flag keeps track of whether the last game played resulted in a new high score. If it did, then the green LED flashes when a game isn't being played. If it didn't, then the red LED flashes.

In `endGame()`, add the following code. It compares the latest score with the saved high score. If it's a new high score, then the flag is set to `true`. Otherwise it is set to `false`:

```
if( currentScore > highScore ) {  
    // if a new high score  
    highScore = currentScore;  
    Serial.println("New High Score!");  
    newHighScore = true;  
}  
else{  
    // else new no high score  
    newHighScore = false;  
}
```

Now you need to add the code to make the LEDs flash. In `loop()`, inside the `else` statement where you check the button, add the following code:

```
if( newHighScore ) {  
    digitalWrite(greenLED, HIGH);  
    delay(200);  
    digitalWrite(greenLED, LOW);  
    delay(200);  
}  
// blink red if flag is false  
else {  
    digitalWrite(redLED, HIGH);  
    delay(200);  
    digitalWrite(redLED, LOW);  
    delay(200);  
}
```

Upload the sketch and check that it works. You should be able to start a game when the Arduino Uno is turned on or reset, then when the game ends the green LED flashes if you get a new high score. If you didn't achieve a new high score, the red LED flashes until you start a new game.



If you are having problems getting your sketch to compile because of a typo, try downloading the sketch from the companion site ([www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino)). Reading through how the code works before trying to type it all will help you better understand what is going on.

## Adding Sounds

You already have written the code that plays sounds when points are scored in Part One of this adventure. The only thing left to do is to put it in its own function. That makes the code easier to read and makes it easier to understand what is going on.

Cut the two lines of code in `loop()` that play a sound when a piezo is triggered and put it in its own function called `playScoreMusic` at the end of the sketch:

```
void playScoreMusic() {  
    tone(speakerPin, 659, 300);  
    delay(300);  
}
```

Now you need to call the function you just wrote. In `loop()`, call the function right after you have printed the current score:

```
playScoreMusic();
```

There are just three more tone sequences to code. The first is the music that will be played when a new game starts. At the end of the sketch, add a new function called `playStartMusic()` with the following lines of code:

```
void playStartMusic() {  
    tone(speakerPin, 523, 300);  
    delay(300);  
    tone(speakerPin, 659, 300);  
    delay(300);  
    tone(speakerPin, 784, 300);  
    delay(300);  
    tone(speakerPin, 1047, 500);  
    delay(600);  
}
```

At the end of `startGame`, call the function you just wrote:

```
playStartMusic();
```

One of two different tone sequences are played at the end of the game. Which one is played depends on whether a new high score was just achieved. The sequence for new high score sounds happier than the tones played when you didn't get a new high score.

At the end of your sketch, add the `playNewHighScoreMusic()` and `playSadEndMusic()` functions:

```
void playNewHighScoreMusic() {  
    tone(speakerPin, 880, 300);  
    delay(200);  
    tone(speakerPin, 440, 500);  
    delay(200);  
    tone(speakerPin, 880, 300);  
    delay(200);  
    tone(speakerPin, 440, 500);  
    delay(200);  
    tone(speakerPin, 880, 300);  
    delay(200);  
    tone(speakerPin, 440, 500);
```

```
delay(200);
tone(speakerPin, 880, 300);
delay(500);
}
void playSadEndMusic() {
  tone(speakerPin, 698, 300);
  delay(300);
  tone(speakerPin, 622, 300);
  delay(300);
  tone(speakerPin, 587, 300);
  delay(300);
  tone(speakerPin, 523, 500);
  delay(600);
}
```

Both functions are called within the `endGame` function. `playNewHighScoreMusic` is called inside the `if` statement that checks whether a new high score was achieved, and `playSadEndMusic` is called inside the `else`:

```
if( currentScore > highScore ) {
  // if a new high score
  highScore = currentScore;
  Serial.println("New High Score!");
  newHighScore = true;
  // play new high score music
  playNewHighScoreMusic();
}
else{
  // else no new high score
  newHighScore = false;
  // play end music
  playSadEndMusic();
}
```

# CHALLENGE



Try changing the start and end `tone()` sequences to customise your game!

And that's it! You now have the full sketch. The full code is shown here so that you can check what you've written against it. Upload the sketch to your Arduino Uno and test it out with your breadboard circuit. After you are happy with how your game works, you are ready to finish building your maze game!

```
// variables for pins
int pointsPins[] = {
    A0, A1, A2, A3, A4};
int buttonPin = 7;
int greenLED = 4;
int yellowLED = 5;
int redLED = 6;
int speakerPin = 9;

// number of piezos for points
int numPointsPins = 5;

// when points triggered
int piezoThreshold = 800;

//game timer variables
int maximumTime = 10000;
long gameStartTime;
boolean playingGame = false;

// high score variables
boolean newHighScore = false;
int currentScore;
int highScore = 0;

-----setup-----
// runs once when board first powered
// or reset
void setup() {
    // set up pin mode for button
    pinMode(buttonPin, INPUT_PULLUP);

    // set up pin modes for LEDs
    pinMode(greenLED, OUTPUT);
    pinMode(yellowLED, OUTPUT);
    pinMode(redLED, OUTPUT);

    // start serial communication
    Serial.begin(9600);

    // start countdown to start
```

```

    startGame();
}

//---loop-----
// runs continuously after setup()
void loop() {
    // if playing a game and still within time
    if( (millis() - gameStartTime) < maximumTime ){
        // read in each points pin
        int i;
        for( i=0; i<numPointsPins; i++ ) {
            int currPinValue = analogRead(pointsPins[i]);

            // if above the threshold
            if( currPinValue > piezoThreshold ){
                // add points
                int newPoints = (i+1)*10;
                currentScore += newPoints;

                // print points and new score
                Serial.print(" Score! ");
                Serial.print(newPoints);
                Serial.println(" points");
                Serial.print(" Current score: ");
                Serial.print(currentScore);
                Serial.println(" points");

                // play scoring music
                playScoreMusic();

                // pause so that same marble doesn't score twice
                delay(300);
            }
        }
    }
    else{
        // else if playing a game but time has run out
        if( playingGame ){
            // end the game
            endGame();
        }
        // else if not playing a game
        else{
            // check if button has been pressed to start new game
            int buttonValue = digitalRead( buttonPin );
            if( buttonValue == 0) {
                // button is pressed, start new game
                startGame();
            }

            // blink green if newHighScore flag is true
            if( newHighScore ) {
                digitalWrite(greenLED, HIGH);
                delay(200);
                digitalWrite(greenLED, LOW);
                delay(200);
            }
        }
    }
}

```

```
        }
        // blink red if flag is false
        else {
            digitalWrite(redLED, HIGH);
            delay(200);
            digitalWrite(redLED, LOW);
            delay(200);
        }
    }
}

//---startGame-----
// sets up variables for a new game and starts
// countdown
void startGame() {
    // make sure all LEDs start off
    digitalWrite(redLED, LOW);
    digitalWrite(yellowLED, LOW);
    digitalWrite(greenLED, LOW);

    Serial.println("*****NEW GAME*****");
    Serial.print("Starting game in...");
    // turn on red LED
    digitalWrite(redLED, HIGH);
    Serial.print("ready...");
    delay(1000);
    // turn off red LED
    digitalWrite(redLED, LOW);

    // turn on yellow LED
    digitalWrite(yellowLED, HIGH);
    Serial.print("set...");
    delay(1000);
    // turn off yellow LED
    digitalWrite(yellowLED, LOW);

    Serial.println("go!");
    // turn on green LED
    digitalWrite(greenLED, HIGH);

    //play start music
    playStartMusic();

    // start game timer
    gameStartTime = millis();
    // set flag that currently playing a game
    playingGame = true;
    // reset score
    currentScore = 0;
}

//---endGame-----
// sets up variables for a new game and starts
// countdown
```

```
void endGame() {
    Serial.println("Game Over!");
    Serial.print("Score: ");
    Serial.println(currentScore);

    // turn off green LED
    digitalWrite(greenLED, LOW);

    // calculate high score
    if( currentScore > highScore ) {
        // if a new high score
        highScore = currentScore;
        Serial.println("New High Score!");
        newHighScore = true;
        // play new high score music
        playNewHighScoreMusic();
    }
    else{
        // else new no high score
        newHighScore = false;
        // play end music
        playSadEndMusic();
    }

    Serial.print("High Score is: ");
    Serial.println( highScore );
    Serial.println();

    // set flag that not currently playing a game
    playingGame = false;
}

//---playStartMusic-----
// plays starting tone sequence
void playStartMusic() {
    tone(speakerPin, 523, 300);
    delay(300);
    tone(speakerPin, 659, 300);
    delay(300);
    tone(speakerPin, 784, 300);
    delay(300);
    tone(speakerPin, 1047, 500);
    delay(600);
}

//---playScoreMusic-----
// plays scoring tone
void playScoreMusic() {
    tone(speakerPin, 659, 300);
    delay(300);
}

//---playSadEndMusic-----
// plays sad tone sequence
void playSadEndMusic() {
```

```
tone(speakerPin, 698, 300);
delay(300);
tone(speakerPin, 622, 300);
delay(300);
tone(speakerPin, 587, 300);
delay(300);
tone(speakerPin, 523, 500);
delay(600);
}

//---playNewHighScoreMusic-----
// plays happy new high score tone sequence
void playNewHighScoreMusic() {
  tone(speakerPin, 880, 300);
  delay(200);
  tone(speakerPin, 440, 500);
  delay(200);
  tone(speakerPin, 880, 300);
  delay(200);
  tone(speakerPin, 440, 500);
  delay(200);
  tone(speakerPin, 880, 300);
  delay(200);
  tone(speakerPin, 440, 500);
  delay(200);
  tone(speakerPin, 880, 300);
  delay(500);
}
```

## Part Four: Building the Maze Game

Now that you have your code and circuit tested and working, it's time to finish building your maze. You are going to use a box with no lid—so three sides and bottom, but no top. You will be making a partial top for the box that contain the maze and holes for the marble to drop through. If you want to cover the box with paint or paper to decorate it, you might find it easier to do before cutting holes and assembling the electronics inside it.



You can watch a video of how to build your maze game on the companion site at  
[www.wiley.com/go/adventuresinarduino](http://www.wiley.com/go/adventuresinarduino).

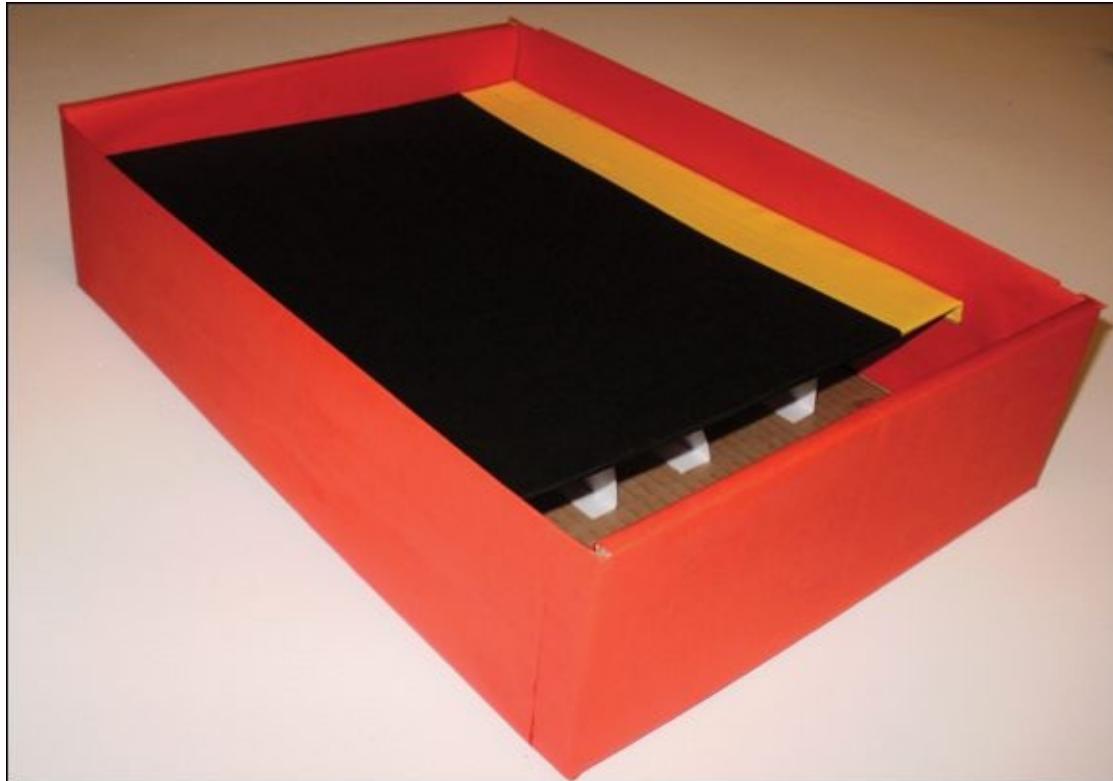
## Making the Maze

Use the following steps to create the maze:

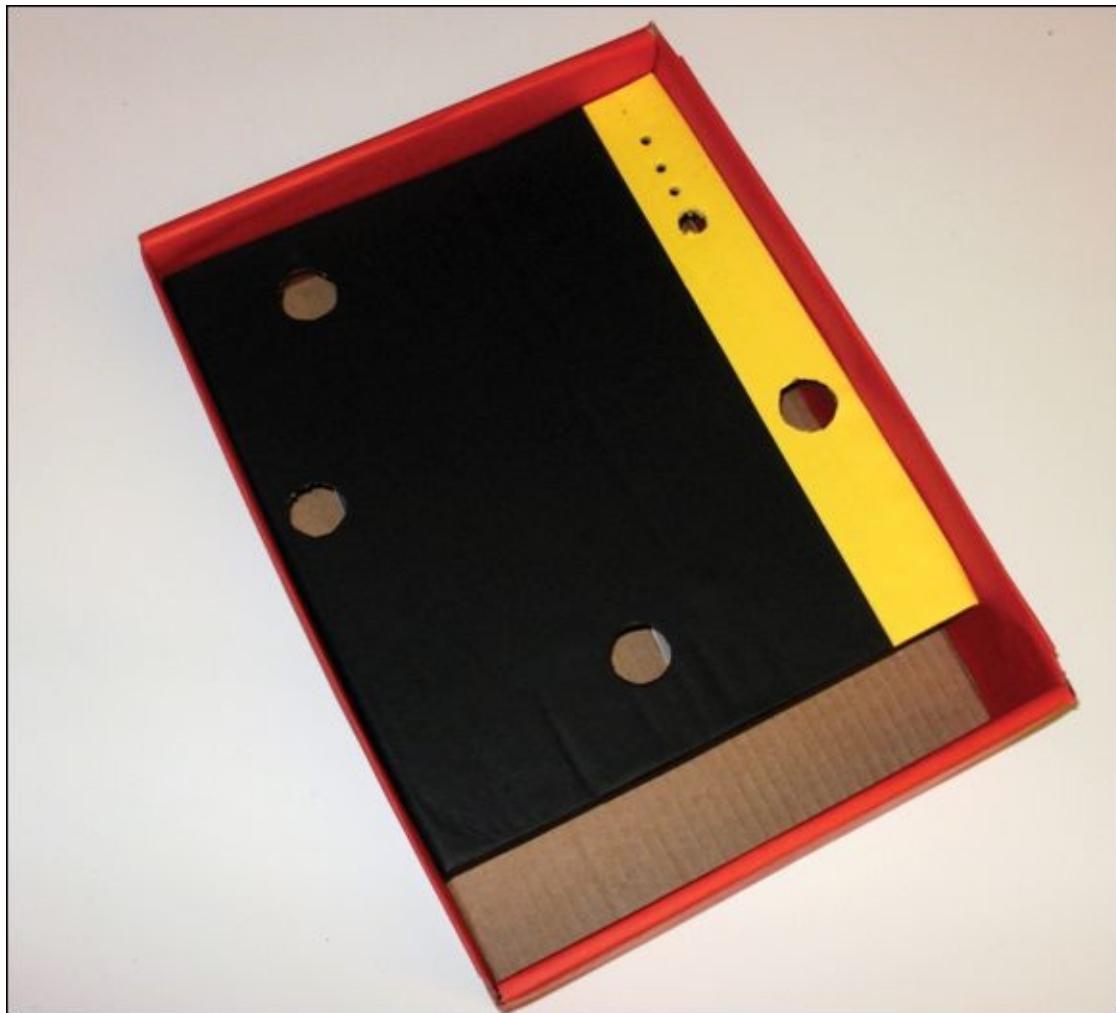
1. Cut four strips of very thick paper or card that are the length of your maze and approximately 3" (8 cm) high. Fold over about half an inch (1 cm) of the card so that you can glue the strips of paper to the inside of the box. These strips stop the marble from rolling onto another piezo and accidentally triggering more points after it has dropped from the maze. Glue the strips of paper into place as shown in [Figure 9-10](#).
2. Cut another piece of cardboard about 2" (5 cm) wider than the bottom of your box width and about 2" (5 cm) shorter than the length of your maze. You can fold the extra inch on each side into flaps to help keep the maze firmly in place in the box. See [Figure 9-11](#) as a guide.
3. Now's the time to find the piece of paper on which you created your design for the maze earlier in this adventure. Using this as your guide, mark the maze and hole locations on the piece of cardboard that you just cut. Mark the holes for the LEDs and button.
4. Make the holes in the cardboard for the marble, LEDs and button, as shown in [Figure 9-12](#).
5. Create "walls" from paper to form your maze. Cut strips of paper and fold an edge to glue to the cardboard along your maze lines.
6. Use paint or markers to decorate the cardboard in any way you like—here's where you can let your imagination run wild! You probably want to include some indication of the number of points that can be scored at each hole.



**Figure 9-10** Glue strips of card to guide the marble after it drops through a hole.



**Figure 9-11** Lid of the maze game fitted to bottom



**Figure 9-12** Maze game before electronics

## Assembling the Piezos

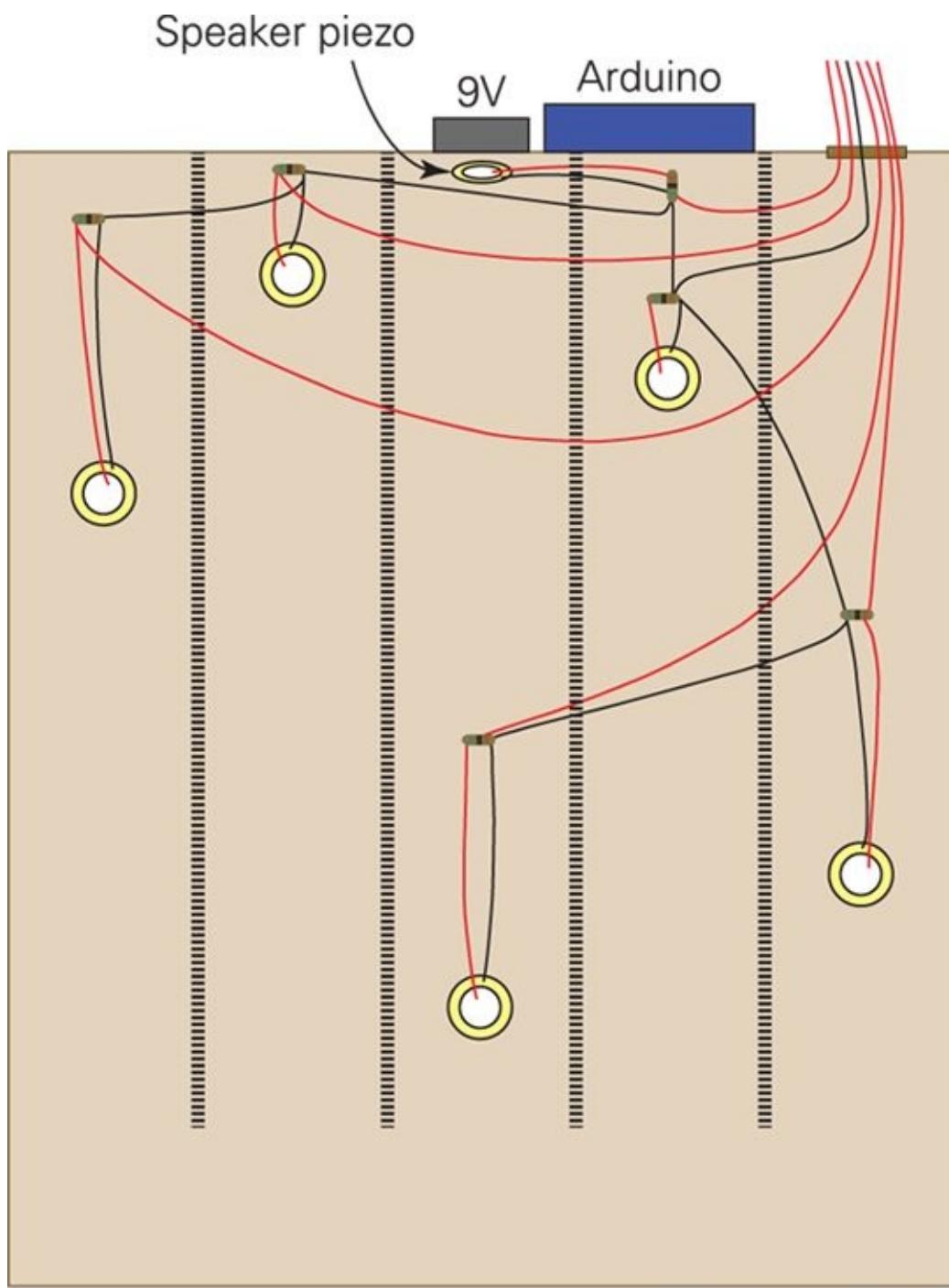
You are now going to start assembling your circuit. Use the following steps to assemble the piezos:

1. Mark on your box where your piezos, LEDs and button will be located.
2. In the upper-right corner of your box, poke a hole through the cardboard so that all the wires from inside the box can pass through and eventually reach the Arduino Uno on the outside of the box.
3. Cut wires for each of the piezos that reach from the red wire of each piezo to the Arduino Uno. Cut another set of wires that connect all the black wires from the piezos to each other (see [Figure 9-13](#)).
4. Solder a  $1M\Omega$  resistor between the red and black wires of the piezos. Solder the wires onto the resistor on the same side as the red wire from the piezos. Solder the wires connecting each of the black wires from the piezos. See [Figure 9-13](#) for guidance. Label the wires so you know which wire goes to which pin.



Only start the soldering steps when there is an adult nearby to help you. Soldering can be dangerous, so be careful!

5. Insert the six wires soldered to the red wires of the piezos into their input pins on the Arduino Uno. Insert the wire soldered to the piezos' black wires into GND. Upload the maze game sketch and test that the piezos all work. You should be able to score points with five of them, and the sixth should play back the sound effects and music.



Bottom of box

[\*\*Figure 9-13\*\*](#) Wiring layout for piezos

## Assembling the LEDs and Button

When you're sure that any wet paint or glue that you used in your decorations have dried, use the following steps to assemble the LEDs and button:

1. Solder a current-limiting resistor onto each of the positive legs of the LEDs. Cut and solder on wires that reach from each of the resistors to the Arduino Uno.
2. Cut and solder two wires from each of the contacts on the button that reaches to the Arduino Uno.
3. Place the LEDs and button in their holes on the cardboard top to the game. Solder the negative legs of the LEDs and one contact of the button together. [Figure 9-14](#) shows what you should have in front of you once you've finished soldering.



[Figure 9-14](#) Solder the negative legs of the LEDs and one contact of the button together.

# Completing the Finishing Touches

You just need a few finishing touches to complete this adventure! Use the following steps to bring it all together:

1. Pass the all the wires from the piezos, LEDs and pushbutton through the hole in back of the bottom box. Attach the Arduino Uno to the outside of the box near the hole for the wires using masking tape.
2. Connect the wires to the Arduino Uno with the sensor piezos connecting to analog inputs, and connect the speaker piezo, button and LEDs to digital pins. One wire connecting all the negative wires of the piezos should connect to one GND pin, and another wire connecting the negative legs of the LEDs and button should connect to another GND pin.
3. Test that your lights, button and piezos are all acting as you expect by playing a round of your new game.

Finished! Congratulations on completing your big adventure! That was a lot of work but I hope you found it rewarding. You can now relax by playing a few rounds of your new marble maze game!

**Arduino Command Quick Reference Table**

Command	Description
<code>boolean</code>	Data type for a variable. Can be either <code>true</code> or <code>false</code> . See also <a href="http://arduino.cc/en/Reference/BooleanVariables">http://arduino.cc/en/Reference/BooleanVariables</a> .
<code>else</code>	Code that is executed only if the preceding <code>if</code> statement was <code>false</code> . See also <a href="http://arduino.cc/en/Reference/Else">http://arduino.cc/en/Reference/Else</a> .
<code>millis()</code>	Function that returns how long the Arduino sketch has been running in milliseconds. See also <a href="http://arduino.cc/en/Reference/Millis">http://arduino.cc/en/Reference/Millis</a> .

# Further Adventures: Continuing Your Adventures with Arduino

The Arduino is a great way to learn about electronics and coding but of course that is only half the fun. As you've seen, projects really come alive when you start embedding your Arduino into physical objects. You have used a lot of different hand tools and techniques, but why not start exploring digital tools like laser cutters and 3D printers? Check out Make Magazine's site (<http://makezine.com/3d-printing/>) to get started.

You could also take the electronics and coding skills you've learned with the Arduino and use them with the Raspberry Pi! Check out *Adventures in Raspberry Pi* by Carrie Anne Philbin (Wiley, 2014).

Most importantly, remember that you are now a member of a worldwide Arduino community. You can always find more resources and tutorials on the Arduino website (<http://arduino.cc>) along with a forum full of nice people ready to answer your questions.



**Achievement Unlocked:** Arduino mastermind!



**Appendix A**  
**Where to Go  
From Here**

**JUST BECAUSE YOU'VE** reached the end of your Arduino adventures here, it doesn't mean your adventures are finished! It's time for you to venture out on your own. There are lots of things to explore and many resources to help you explore them.

# More Boards, Shields, Sensors and Actuators

You have already looked at two Arduino boards besides the Arduino Uno: the Lilypad and Leonardo. Arduinos come many shapes and sizes, so if you have a project in mind, explore what might be best for it. Does it need to be small? The Sparkfun Arduino Pro Mini ([www.sparkfun.com/products/11113](http://www.sparkfun.com/products/11113)) or Arduino Micro (<http://arduino.cc/en/Main/ArduinoBoardMicro>) might be perfect. Do you want to add lots of touch sensors and play audio files? The Bare Conductive Touch Board ([www.bareconductive.com/shop/touch-board/](http://www.bareconductive.com/shop/touch-board/)) does that all on a single board!

## Shields

A shield is a soldered circuit that fits perfectly on top of your Arduino Uno board. It can hold some complicated circuitry like an Arduino Ethernet Shield (<http://arduino.cc/en/Main/ArduinoEthernetShield>) that lets you connect your Arduino to the Internet with an Ethernet cable. There are shields that add touch screens, MP3 players, motor control and much more. Visit your favourite store—whether in person or online—and see what's available.

## Sensors and Actuators

There are also many more sensors available than you've used for the adventures in this book. Just about anything you can sense, you can get your Arduino to sense as well. Want to detect the barometric pressure? Try the BMP180 ([www.adafruit.com/products/1603](http://www.adafruit.com/products/1603)). How about sound? Try an electret microphone from SparkFun (<https://www.sparkfun.com/products/9964>).

The same goes for actuators. There are many types of motor and speaker and, of course, there is a huge selection of LEDs and screens. The Adafruit NeoPixel is a great RGB LED that you can control with an Arduino ([www.adafruit.com/category/168](http://www.adafruit.com/category/168)).

Of course, an alternative to buying more sensors is to make your own! Kobakant's How to Get What You Want (<http://kobakant.at/DIY>) is a collection of guides and documentation on DIY sensors, often using materials like conductive thread and fabric.

## On the Web

The web is full of projects and resources. Only a few of the most popular ones are listed here, so search around for more!

## The Arduino Site

The first stop for any aspiring Arduino engineer is the Arduino website (<http://arduino.cc>). You will find everything you need to know about every official Arduino board that is made—and there are a lot of them. It's also the home of the Arduino playground (<http://playground.arduino.cc>) where Arduino users can upload their own projects and tutorials.

If you have any questions or problems trying to bring a project to life, you can ask a question on the Arduino Forum (<http://forum.arduino.cc>). There are always friendly folk willing to help, but it's good to do a couple of things before asking a question. First, make sure it's not a question that has already been asked. Search around the forums—maybe someone else has already provided all the information you need. Second, give as many details as you can. Describe what you have tried to do, what you want to happen and exactly what is actually happening. This makes it much easier for others to help you and more likely that they will reply.

## Manufacturers

Other than the Arduino website, the best resources are two companies that build their own sensors, actuators and kits for the Arduino: SparkFun and Adafruit. Both have excellent sites full of tutorials and guides. Adafruit is at <https://learn.adafruit.com> and SparkFun is at <https://learn.sparkfun.com>.

## Blogs

If you are in need of some inspiration for your next project, try out Adafruit's blog (<http://adafruit.com/blog>), Make Magazine (<http://makezine.com>) or Hackaday (<http://hackaday.com>). There are lots of specialist blogs as well. For example, if you are into wearable technology, keep your eye on Fashioning Technology (<http://fashioningtech.com>). Want to take your Arduino to the skies? DIY Drones (<http://diydrones.com/>) can help you with that.

## Videos

Sometimes nothing beats seeing a video demonstration of a new skill. YouTube channels are a great way to learn about a new sensor or how to make a new project. You won't be surprised to learn that both Adafruit (<https://www.youtube.com/user/adafruit>) and SparkFun (<https://www.youtube.com/user/sparkfun>) have YouTube channels overflowing with information. Make Magazine has one as well (<https://www.youtube.com/user/makemagazine>).

For an excellent series of electronics videos, search YouTube for "Collin's Lab". Collin Cunningham teaches basic electronics in a way that's easy to follow. Some of the videos are made with Make Magazine and others are from Adafruit, so just search for "Collin's Lab".

Massimo Banzi, one of the founders of Arduino, made a series of videos that accompanies the official Arduino Starter Kit. The videos have a project book, but you can only get it by buying the kit. The first video is at [https://www.youtube.com/watch?v=2X8d\\_r0p92U](https://www.youtube.com/watch?v=2X8d_r0p92U).

# **Books**

Physical books are great to have on hand for reference. There are a lot of them out there and more are being written about the latest technology all the time, but here are a few to get you started.

## Getting Started with Arduino and General Projects

There are far more projects you can make with an Arduino than you have made with this book. Go out and make even more projects! Build more things!

- *Getting Started with Arduino* by Massimo Banzi (Maker Media, Inc., 2011)
- *Arduino For Dummies* by John Nussey (Wiley, 2013)
- *Arduino Projects For Dummies* by Brock Craft (Wiley, 2013)
- *Exploring Arduino: Tools and Techniques for Engineering Wizardry* by Jeremy Blum (Wiley, 2013)

## General Electronics

Electrical circuits include much more than just Arduinos. If you would like to start making really advanced projects, it's a good idea to learn more about circuits. These books can get you going:

- *Make: Electronics* by Charles Platt (Make, 2009)
- *Practical Electronics for Inventors* by Paul Scherz and Simon Monk (Tab Books, 2013)

## Soft Circuits and Wearables

Electronics and crafting techniques like sewing actually go together really well. If you're interested in exploring this perfect match, here are some great books to get you started:

- *Fashioning Technology* by Syuzi Pakhchyan (Maker Media, Inc., 2008)
- *Switch Craft: Battery-Powered Crafts to Make and Sew* by Alison Lewis and Fang-Yu Lin (Potter Craft, 2008)
- *Sew Electric* by Leah Buechley, Kanjun Qi, and Sonja de Boer (HLT Press, 2013)
- *Make: Wearable Electronics* by Kate Hartman (Maker Media, Inc., 2014)

## Other Specialised Topics

Of course, there are so many more things that Arduinos can do. How about investigating Arduino robots or Arduinos that can talk to the Internet?

- *Making Things Move: DIY Mechanisms for Inventors, Hobbyists, and Artists* by Dustyn Roberts (McGraw-Hill, 2011)
- *Making Things Talk: Using Sensors, Networks, and Arduino to See, Hear, and Feel Your World* by Tom Igoe (Maker Media, Inc., 2011)

**Appendix B**

**Where to Get Tools  
and Components**

NAVIGATING YOUR WAY through the world of tools and electrical components can be difficult, but luckily the growth of do-it-yourself electronics and coding projects has made it easier than ever before to find what you need for your Arduino project.

# Starter Kits

A really easy way to get going with the adventures in this book is to buy a starter kit. A starter kit will include an Arduino Uno and almost all the components you need like LEDs, resistors and a servo. You can compare the list of components you need in the Introduction with what is available in a starter kit and then buy whatever isn't included. You will still need to buy an Arduino Leonardo and Lilypad Arduino USB separately to complete [Adventures 7](#) and [8](#).

Almost all the stores listed in this appendix sell their own starter kits, so there are lots of options. The Arduino company makes its own starter kit that comes with a whole book of projects. You can buy it from a reseller or directly from the Arduino shop online (<http://store.arduino.cc/product/K000007>).

# **Brick-and-Mortar Stores**

A brick-and-mortar store simply means a shop! Being able to walk into a store to find the components you need has some benefits. For example, when working with physical components, it's always useful to be able to pick them up and see them for yourself. You can also ask questions from the helpful staff in the store. Plus, you don't have to wait for your package to be delivered; you can go home and get making right away!

## In the UK

Maplin ([www.maplin.co.uk](http://www.maplin.co.uk)) sells all sorts of things and has a wide range of stock; you can now buy Arduinos there, along with electrical components. Most of the smaller components like resistors and LEDs are kept behind the counter, so you can look up the product code in a catalogue or in-store computer and a member of staff will get it for you. They also stock useful tools like soldering irons.

## In the US

RadioShack ([www.radioshack.com](http://www.radioshack.com)) had long been the place for hobbyist electronics. The chain filed for bankruptcy in 2015, leaving those in the United States without a physical store to visit. There isn't yet a chain as large as RadioShack to take its place, but spaces like TechShop (<http://techshop.ws/>) are becoming more widespread and contain shops selling electronics.

# **Online Stores**

There are broadly two different kinds of online store for electrical components: friendly hobbyist or specialist sites; and vast catalogue sites. If you are just starting out, it's better to stick with a friendly site that doesn't stock so many different options but does stock what you will most likely need.

The bigger sites have thousands, if not millions, of components so can be difficult to navigate if you don't know exactly what you are looking for. However, they tend to be cheaper than other online stores and also stock less popular items that are harder to find.

## Online Stores Shipping from the EU

Adafruit and Sparkfun are the best sites for reference, but you might not want to deal with international shipping if you don't live in North America. Luckily there are a large number of EU distributors that import Adafruit and Sparkfun products. That means you can get making faster! Try some of these sites to see if they have what you need:

- Arduino Store: <http://store.arduino.cc>
- Cool Components: [www.coolcomponents.co.uk](http://www.coolcomponents.co.uk)
- Maplin: [www.maplin.co.uk](http://www.maplin.co.uk)
- Oomlout: <http://oomlout.com>
- Pimoroni: <http://shop.pimoroni.com>
- Proto-Pic: <http://proto-pic.co.uk>
- RobotShop: [www.robotshop.com/eu/en](http://www.robotshop.com/eu/en)
- SK Pang: <http://skpang.co.uk>

Some big catalogue sites that serve the EU include the following:

- Digi-Key: [www.digikey.co.uk](http://www.digikey.co.uk)
- Farnell: [www.farnell.com](http://www.farnell.com)
- Mouser: <http://uk.mouser.com>
- Rapid: [www.rapidonline.com](http://www.rapidonline.com)
- RS Components: [www.rs-components.com/index.html](http://www.rs-components.com/index.html)

## Online Stores Shipping from the US or Canada

The two maker-focused sites that all other maker companies aspire to be are Adafruit ([www.adafruit.com](http://www.adafruit.com)) and Sparkfun (<https://www.sparkfun.com>). Both have excellent guides and tutorials for how to use everything they sell. A lot of their stock overlaps, but each company also makes their own products. These sites should always be your first stop online. They are both located in the US, so read on to the next section if you are not in North America and don't want to wait (or pay) for your order to arrive from far away.

You can also check out these other smaller online sites that are also in the US:

- Maker Shed: [www.makershed.com](http://www.makershed.com)
- RobotShop: [www.robotshop.com](http://www.robotshop.com)
- Spinkenzie Labs: [www.spikenzielabs.com](http://www.spikenzielabs.com)

If you want to try one of the large catalogue sites based in the US, check out some of these:

- Allied Electronics: <http://ex-en.alliedelec.com>
- Digi-Key: [www.digikey.com](http://www.digikey.com)
- Jameco: [www.jameco.com](http://www.jameco.com)
- Mouser: [www.mouser.com](http://www.mouser.com)
- Newark: [www.newark.com](http://www.newark.com)

## Glossary

### actuator

A device that translates an electrical signal into a real-world action such as light, sound or movement. Examples include motors, lights and speakers.

### alligator clips

Wires with spring-loaded clips that resemble the jaws of an alligator. They are useful for prototyping soft circuits or connecting components that don't use jumper wires.

### analogue

A signal that varies between **LOW** and **HIGH**, as opposed to a digital signal. On the Arduino Uno, an analogue signal can be measured as a number between 0 for ground and 1023 for 5V. An analogue signal can be output as a value between 0 for 0V and 255 for 5V.

### anode

The positive leg of a directional component, such as the long leg of an LED.

### argument

A piece of information given to a function, which the function then uses to perform its task. The argument goes inside the brackets that follow the function name. For example, the function **delay(1000)** has the argument **1000**, which is the number of milliseconds you want the Arduino to wait before executing the next line.

### array

A list of the same type of thing in code. For example, an array can hold a list of **ints**.

### binary

A number that uses only the digits 0 and 1, as opposed to decimal, which uses the digits 0 to 9. Binary is also known as base-2. Decimal is referred to as base-10.

### breadboard

A reusable device that allows you to create circuits without needing to solder all the components. Breadboards have a number of holes into which you push wires and components to create circuits.

## capacitance

The ability to store an electrical charge. Electrical components built especially to hold a charge are called capacitors, but other objects—even people—also have capacitance.

## cathode

The negative leg of a directional component, such as the short leg of an LED.

## comments

Notes within your code that explain what a line or section of code is intended to do. Each comment line begins with `//` or, if you want to write a comment that spans multiple lines, it is placed between `/*` and `*/`. These special characters tell the computer running the program to ignore that line or lines.

## compiling

The process of taking code written by a human and turning it into instructions that can be understood by a machine.

## current

The rate at which electrical energy flows past a point in a circuit. It is the electrical equivalent of the flow rate of water in pipes. Current is measured in amperes (A). Smaller currents are measured in milliamperes (mA).

## debugging

The process of locating the cause of any errors in your computer program code and fixing them.

## declaring

Where a new variable is created by giving it a name and a data type such as `int`. The variable does not hold a value until it is given its first value.

## digital

A signal that is only either on or off, or `HIGH` or `LOW`. On the Arduino Uno, a `HIGH` signal is 5V and a `LOW` signal is ground.

## direct current (DC)

The type of electricity used in Arduino circuits. It's the same kind that is

generated by a battery and is the opposite of alternating current (AC), which is what comes out of mains plugs in the wall.

## driver

A piece of software that lets your computer communicate with an external device, such as a printer or a keyboard.

## dual in-line package (DIP or DIL)

One possible shape of an IC chip. It has two rows of legs that can fit into a breadboard.

## duty cycle

The ratio of time a signal is **HIGH** versus **LOW** in a given cycle. In PWM, the higher the duty cycle, the higher the output voltage.

## float

A data type for numbers that aren't whole numbers, but include a decimal place such as 1.3 or -54.089.

## floating input

A pin that is not connected to anything. The pin reads in random values if it is not connected to a voltage source such as ground, 5V or a sensor.

## for loop

A programming device that repeats a block of code for a predetermined number of times.

## function

A set of lines of code that have a name. A function can be used over and over again. It may take some information as an input and output more information when it is finished, but not all functions need to do that.

## instantiation

Giving a variable a value for the first time. Instantiation can happen at the same time you declare the variable or you can do it later, but the declaration always needs to come first.

## integrated circuit (IC)

Circuits contained within a single chip. The same circuit can be put into different shaped chips, called packages. When working with a breadboard, you need what is known a DIP or DIL package. That's the shape that has legs that fit into a breadboard.

## integrated development environment (IDE)

A software application that is used to write computer code in a particular

language; it's also referred to as a programming environment. The application can create and edit code, as well as run (or execute) the code. Many IDEs also provide features to help programmers debug their programs—in other words, check their programs for errors.

## light-emitting diode (LED)

An electrical component that lights up when electrical current flows through it. A diode only lets electricity flow in one direction, so an LED lights up only when the long leg is connected to the positive side of a power source and the short leg is connected to the negative side. If the legs are switched, the LED won't light up.

## library

A collection of reusable functions in code that can be imported and used in multiple sketches.

## light-dependent resistor

A resistor that changes its resistance according to how much light it is exposed to. It is also sometimes called a photoresistor.

## long

A data type that can hold whole integer numbers from  $-2,147,483,648$  to  $2,147,483,647$ .

## newline character

A character that represents what happens when you press the Enter or Return key on your keyboard.

## Ohm's Law

The mathematical relationship between voltage, current and resistance. Voltage equals current multiplied by the resistance—or, put another way,  $V=IR$ .

## panel mount push button

A push button that is designed to be mounted inside a case. It comes with a nut and washer to secure it to a panel.

## piezo

A crystal that expands and shrinks when electricity is run through it. It also generates electricity when it is squeezed or bent.

## potentiometer

A type of resistor with an adjustable knob to vary the resistance of current.

## pull-up resistor

A resistor that is connected to the high voltage in a circuit, which sets the default state of the pin on that circuit to **HIGH**. The resistor is usually  $10\text{k}\Omega$ .

## pulse width modulation (PWM)

How the Arduino board generates an output signal between 0V and 5V. The signal switches quickly between LOW and HIGH and the resulting output voltage is between the two voltages.

## red-green-blue light-emitting diode (RGB LED)

A single LED with four legs that contains three lights: one red, one green and one blue. The three lights share either a common anode or a common cathode.

## resistor

An electrical component that resists current in a circuit. For example, LEDs can be damaged by too much current, but if you add a resistor with the correct value to the LED circuit to limit the amount of current, the LED is protected. Resistance is measured in ohms or  $\Omega$ . You need to pick a resistor with the correct value to limit the current through a circuit; the value of a resistor is shown by coloured bands that are read from left to right.

## sensor

A device that detects something in the real world such as light, sound or movement and translates it into an electrical signal. Examples include potentiometers and light-dependent resistors.

## serial communication

A way that two devices, such as a computer and an Arduino board, can send and receive data to each other. One piece of data is sent at a time.

## servo

A motor that can be controlled to rotate to a specific position. It usually can't rotate more than 180 degrees.

## shift register

A device that can control multiple outputs with relatively few inputs. It is commonly used to control a large number of LEDs.

## sketches

Arduino programs. The name comes from the quick drawings artists make.

## **soft circuit:**

Circuit built with flexible materials such as conductive thread and fabric. Soft circuits are often used in projects that are going to be worn.

## **surface-mount device (SMD)**

One possible shape of an IC chip or other component such as a resistor. It is made for soldering onto a flat surface without any legs being inserted into holes on a circuit board.

## **switch**

A component that either disrupts or redirects the flow of current in a circuit.

## **tactile pushbutton**

A type of switch. A push-to-break pushbutton interrupts the flow of current in a circuit when it is pressed. A push-to-make pushbutton does the opposite and interrupts current only when it is not pressed.

## **two-dimensional array**

Data stored in rows and columns, like in a spreadsheet.

## **variable**

A code construct that holds a value that can be changed. For example, the variable `greenLED` stores the number `5`.

## **voltage**

The difference in electrical energy between two points in a circuit. It is the electrical equivalent of water pressure in pipes, and it is this pressure that causes a current to flow through a circuit. Voltage is measured in volts (V).

## **voltage divider**

A circuit that outputs a fraction of the input voltage. It is a useful circuit for translating a change in resistance into a change in voltage.



# **WILEY END USER LICENSE AGREEMENT**

Go to [www.wiley.com/go/eula](http://www.wiley.com/go/eula) to access Wiley's ebook EULA.