

# Love to Code: Volume 1

**Written by Jie Qi**  
**Illustrations by K-Fai Steele**





# Contents



Prologue

Chapter 1: Light up an LED!

Chapter 2: Code a Blink!

Chapter 3: Add a Switch!

Chapter 4: Fade in and Out!

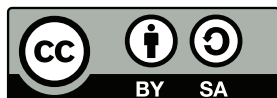
Debugging

Conclusion



**Love to Code: Volume 1** Copyright © 2017 by Jie Qi  
Some rights reserved.

This work is licensed under a Creative Commons  
Attribution-ShareAlike 4.0 International License:  
<https://creativecommons.org/licenses/by-sa/4.0/>



Publisher: Sutajio Ko-Usagi Pte Ltd  
dba Studio Kosagi, in Singapore  
[info@chibitronics.com](mailto:info@chibitronics.com)

Illustrator: K-Fai Steele  
Editor: Andrew “bunnie” Huang  
Technical Editor: Natalie Freed

ISBN: 978-981-11-4688-6



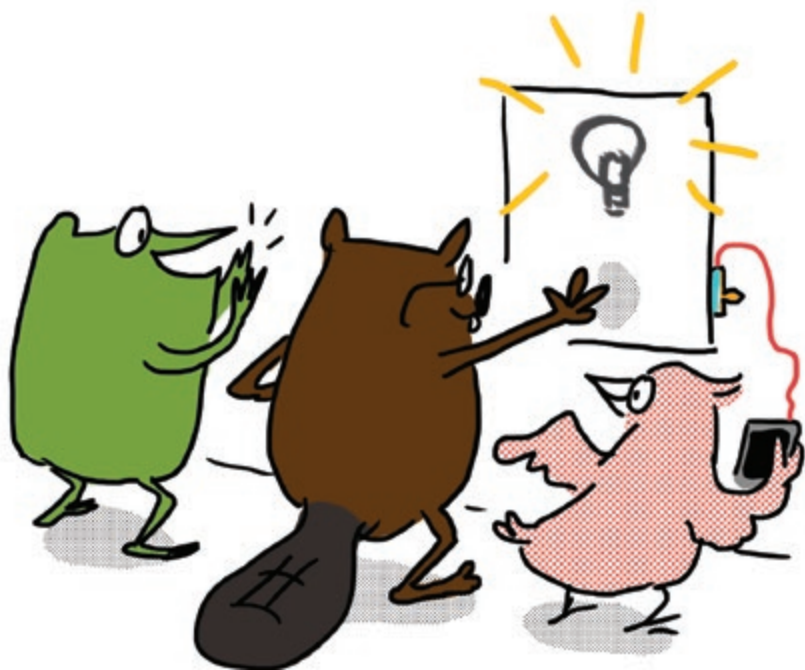


# Prologue

Fern is a really creative frog.



Recently some of Fern’s friends started to make drawings, paintings and costumes that would light up and were interactive...



“My friends are making drawings that **do** things!”



Fern wanted to make drawings that lit up too.



But everything was so complicated!

Her friend Sami, a seal, noticed that Fern was frustrated.

“Fern, do you want me to show you how to add lights to your drawings too?”

“Oh ... you don’t have to,” said Fern.

“But I can tell you want to do it. And look, you’ve already started!” Sami barked cheerfully.

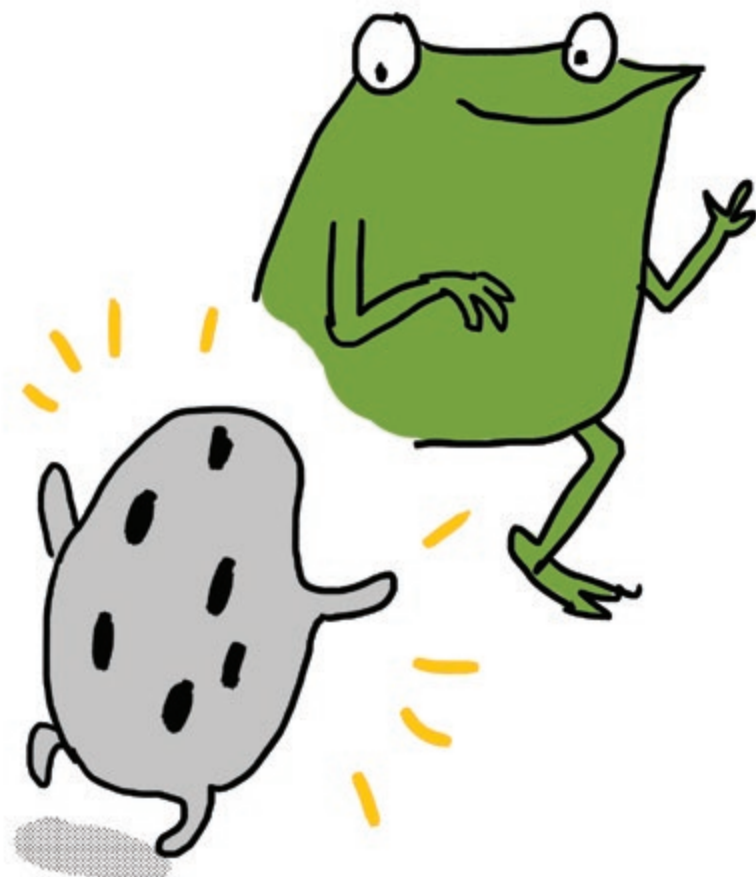


“Well, um,” Fern shuffled. “It’s just hard. There are just so many things to learn and sometimes it feels like I’ll never get there.”

Sami smiled. “Sometimes it’s hard. Learning any new thing can be challenging ... **but it can be fun too!** Let’s give it a try together!”



“I’ll be with you the whole time. What do you have to lose?”



Fern brightened at the thought and nodded with determination at Sami.

In this book, Fern gets help from not only Sami, but also from her whole gang of friends. Fern — and you — will learn how to add lights and sensors to drawings.

Fern promises that if you tag along and complete the activities in this book, your drawings will also be able to “do” something too!



Fern



Sami



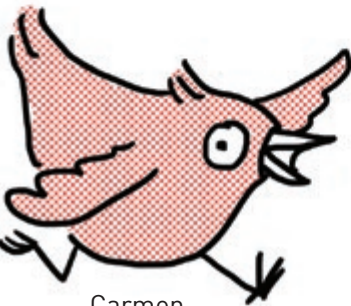
D. Bug



Rusty



Edith

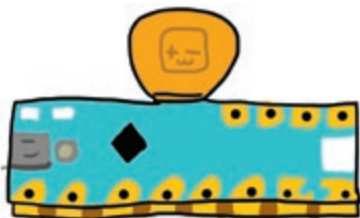


Carmen

The stuff in your Love to Code kit is what we'll use to make our drawings do something. Let's take a look at what's included in the kit!

## Chibi Chip and Cable

The Chibi Chip is the programmable brains behind our projects! It comes assembled with a clip so that we can quickly clip and unclip it to circuits. We use the special red cable to connect the Chibi Chip to our programming device — such as a tablet, phone or computer — and a USB power supply.



## Chibi Book

The Chibi Book is a binder that comes with a built-in USB battery pack capable of powering a Chibi Chip. **Note:** you will need three AA batteries, which are not included in the kit.



To turn on the Chibi Book, slide the switch on the top of the battery pack to the left.



Power is available when the power light is green.



The batteries are low when the low battery light turns red.



### LED stickers

These stickers are LED lights!  
We'll use them to make our projects glow.



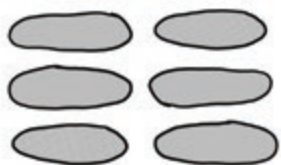
### Copper tape

We use copper tape to connect the different parts of our circuit.



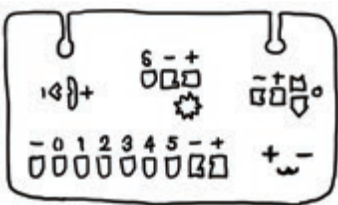
### Fabric tape patches

These conductive fabric patches help us fix broken circuit connections.



### Circuit sticker stencil

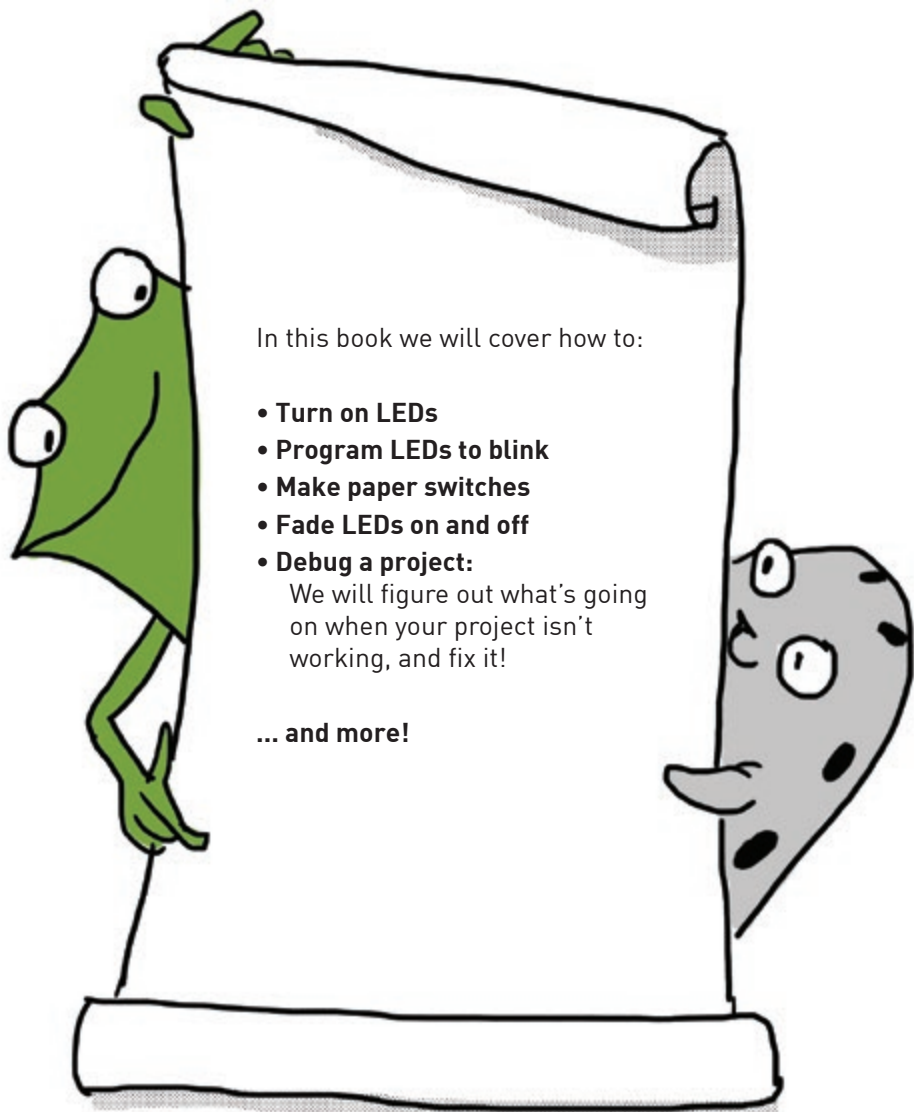
Use this stencil to help you sketch your own circuit designs!



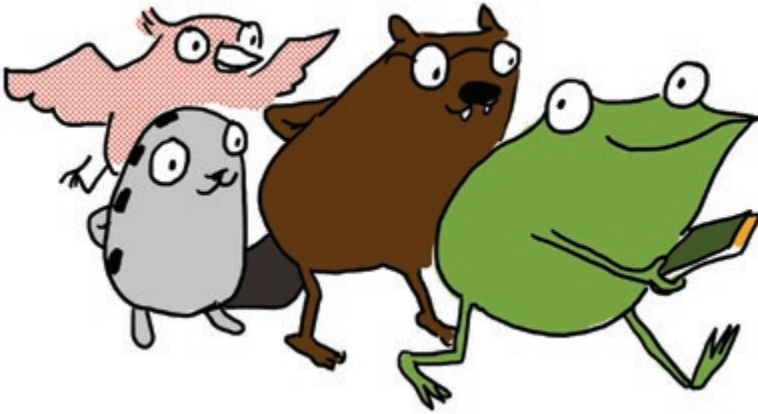
We'll explain how to use all of this stuff in the chapters to come!







You can do the chapters in this book on your own, but they are also a lot of fun to do with a friend!



If you need help or get stuck, check out our web tutorials at:  
**[chibitronics.com/lovetocode](http://chibitronics.com/lovetocode)**

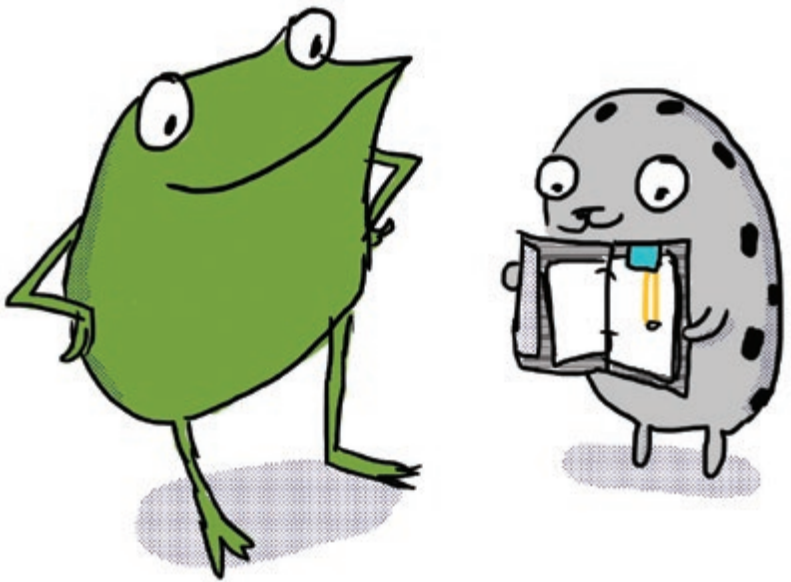
Or send a detailed description of your problem to  
**[help@chibitronics.com](mailto:help@chibitronics.com)**. We'll do our best to respond but please be patient, we're a small team of makers, artists, and dreamers — just like you!



Ready? Let's go!

# Chapter 1:

## Light Up an LED!



"Let's start with the basics," said Sami the seal to Fern the frog.  
"We can turn on a light by connecting it to a Chibi Chip!"

# TURN ON A LIGHT!

In this activity we will power up our Chibi Chip and use it to light up an LED sticker!

**We will need:**



Chibi Light LED sticker



copper tape



USB power  
(Chibi Book, laptop or wall plug)

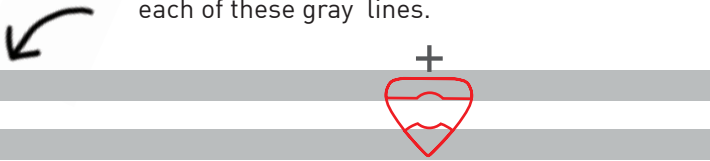


Chibi Chip



USB cable

1. Stick copper tape over each of these gray lines.



2. Stick an LED sticker over the triangle footprint.



3. Plug the Chibi Chip into a power source, so that the green PWR light comes on.



4. Align and clip the Chibi Chip to the page over this rectangle. Make sure the metal pads of the clip touch your circuit.



5. Bask in the soft white glow of the LED!

OH WAIT, THE LED ISN'T ON? TRY THESE DEBUGGING TIPS!



THIS IS D. BUG.  
HE HANGS AROUND  
AND HELPS OUT WHEN  
PROBLEMS SHOW UP!



MAKE A STRONG CONNECTION!  
TRY PRESSING HARD ON THE  
METAL PADS OF THE LED.



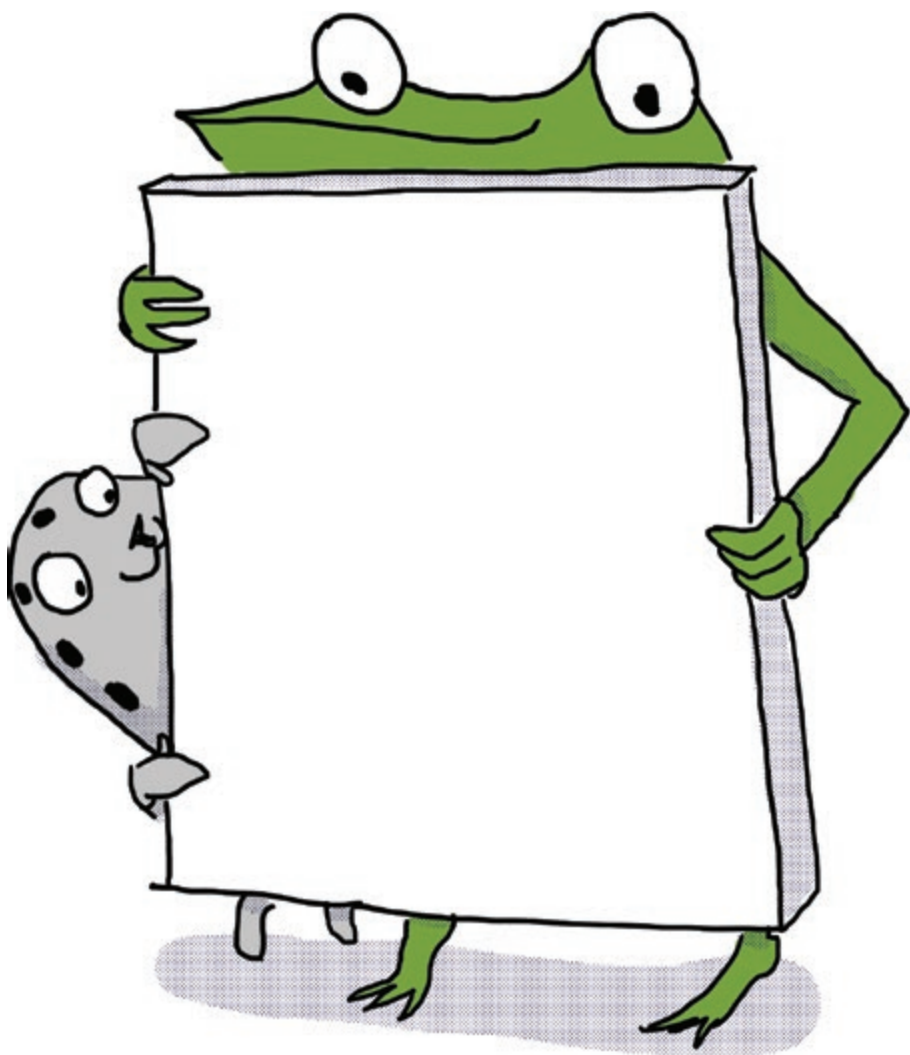
CHECK YOUR ALIGNMENT!  
MAKE SURE THE GOLDEN METALLIC  
STRIPES ON THE CHIBI CHIP ARE  
LINED UP AND MAKING SOLID  
CONTACT WITH THE COPPER TAPE.

STILL NOT LIGHTING UP? CHECK OUT THE DEBUGGING SECTION IN THE BACK OF THIS BOOK!



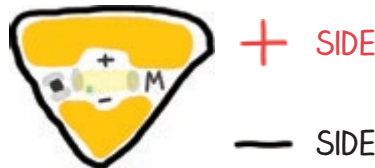
"You can see the LED's light through the paper!" exclaimed Sami. "Fern, let's draw something around the light."

What should Fern draw?

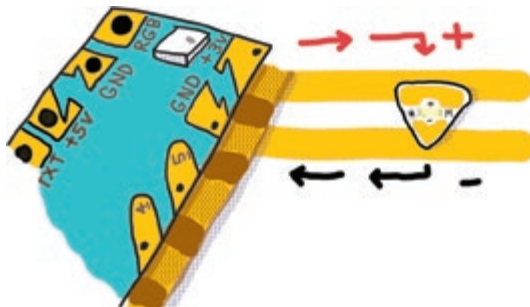


# HOW DOES IT WORK?

LED stickers have two sides: the **positive (+)** side is the wider, rounder part of the triangle, and the **negative (-)** side is the pointy tip. LEDs turn on when the (+) side is connected to the +3V pin and the (-) side is connected to the GND pin of a Chibi Chip. Copper tape is used to connect the LED sticker to the pins of the Chibi Chip.



The Chibi Chip provides power, which is conducted through copper tape. A **conductor**, such as copper, is a material that allows electricity to flow easily and with little resistance. Conductors can be used to connect different parts of a **circuit**. Individual pieces of copper tape within a circuit are called **traces**.



Clipping on the Chibi Chip makes a complete circuit! This means that everything got connected in an unbroken loop that allows power to flow, turning on the LED light.

We can turn on more lights by connecting them in **parallel**, like in this picture to the right.

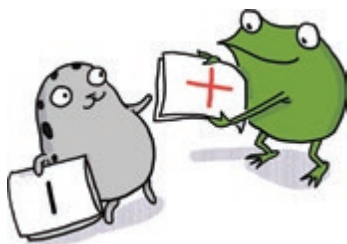




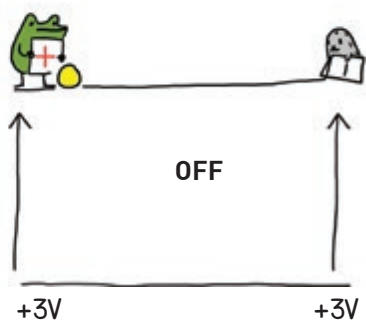
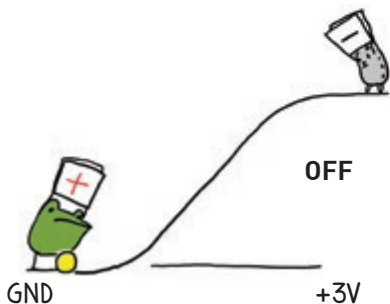
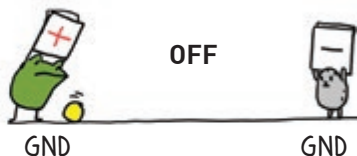
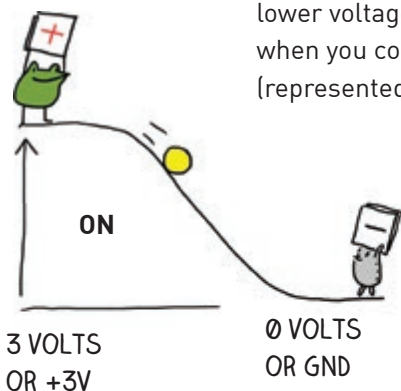
# DOWN THE RABBIT HOLE: LED CIRCUITS

Electricity is a form of **energy** that flows to turn on your light. This flow of energy is called a **current**. Like a ball rolling down a hill, a current will only flow when there is a difference in height, from high to low.

In electronics, the difference in height is called a **voltage**. Voltage is a measurement of potential energy, just like the height of a ball on a hill.

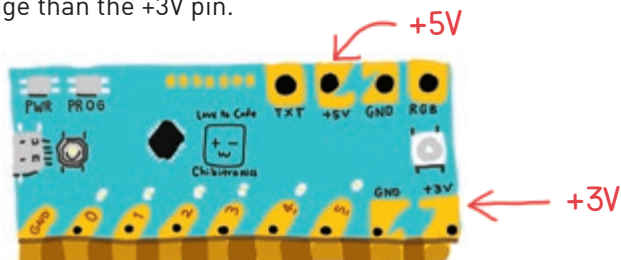


For an LED to turn on, current must flow through the LED from a higher voltage on the (+) side to a lower voltage on the (-) side. Here's what happens when you connect the (+) and (-) ends of your LED (represented by Fern and Sami) to different voltages:

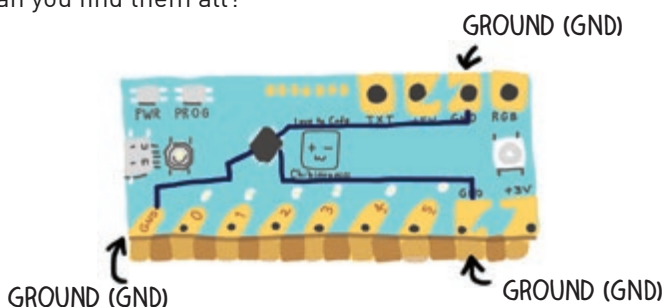




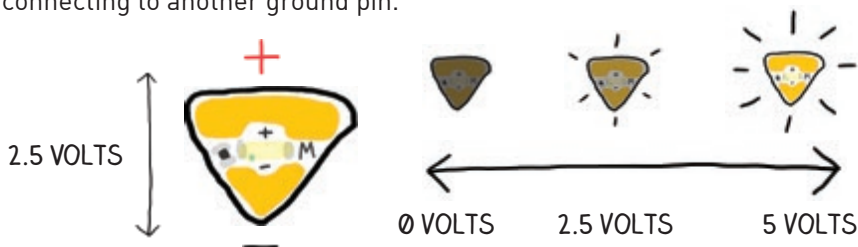
The +3V pin on your Chibi Chip sits at a height of 3 volts. There is another pin on the opposite side labeled +5V. This means it is at 5 volts. This pin provides even more voltage than the +3V pin.



- ✦ 0 volts is called **ground**, and we've labeled the pins with GND. This is where we connect the (-) side of our LED. There are actually three ground pins on a Chip. Can you find them all?

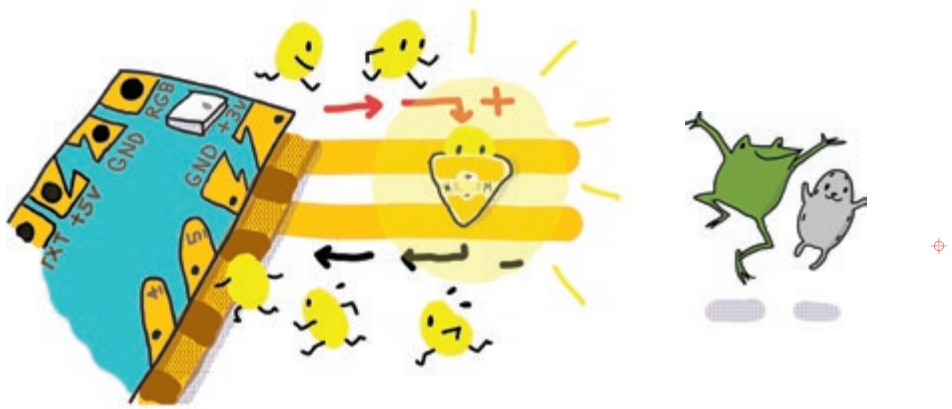


Even though they look like three separate pins, they are actually connected together by copper traces inside the Chibi Chip. This means that they are all at the same voltage (0 volts) so connecting to one ground pin is the same as connecting to another ground pin.



A typical LED sticker needs at least 2.5V to shine brightly, so when you connect it between the +3V pin and GND, there's more than enough voltage to turn the light on. The higher the voltage applied across an LED sticker, the brighter it will shine — up to a point. Be careful, if you give a single LED sticker too much voltage (more than 6V) it can burn out!

To summarize: when we connected the (+) side of the LED sticker to 3 volts and the (-) side to GND (or 0 volts), we provided enough voltage in the right direction for the LED to turn on!

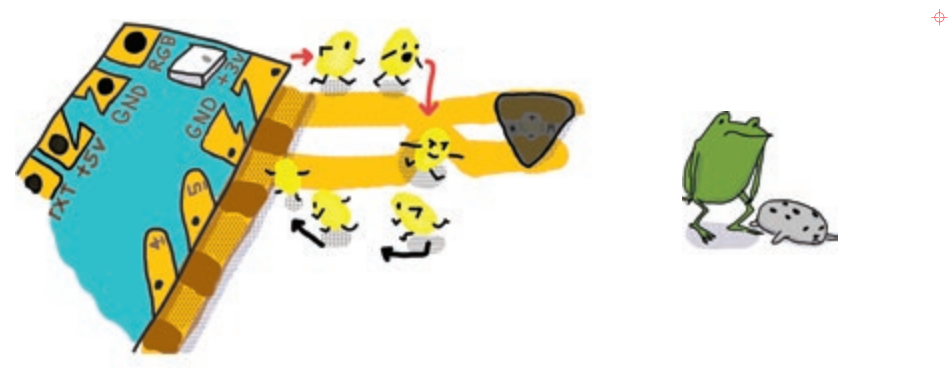


Don't worry if you accidentally connect the LED backwards. Nothing bad will happen, it will just not turn on.

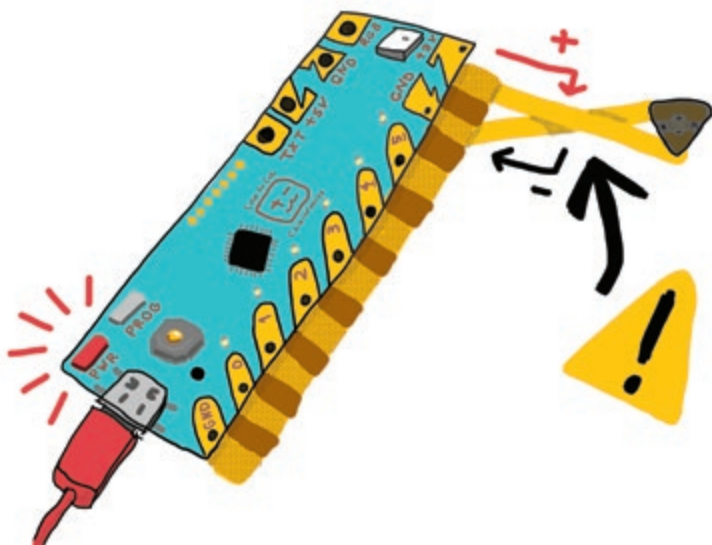
BE CAREFUL NOT TO CONNECT THE +3V PIN TO GND DIRECTLY BY CROSSING TRACES: THIS CREATES A SHORT CIRCUIT!



Electrons are lazy and will take any shortcut instead of going through your LED to turn it on, which is why this is called a "short" circuit.



In fact, short circuits are so appealing for electrons, they will stop flowing through the Chibi Chip and instead rush to flow through the short circuit. If there is a short circuit between power and ground, the Chibi Chip will stop working and it will drain a lot of power.



If this happens to a Chibi Chip, the Chip will shine a red warning light until the short circuit is disconnected.

If too much power flows through the short circuit during this burst, the circuit will get very warm, and the circuit might become permanently damaged if it gets too hot. So if you notice a short circuit, make sure to disconnect the Chibi Chip right away!



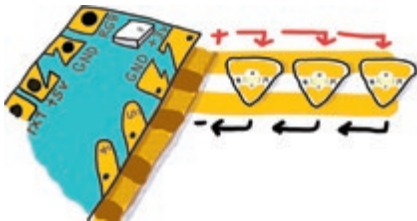


# HOW CAN WE TURN ON A BUNCH OF LIGHTS?

Here are a couple of ideas:

## Parallel Circuits

Parallel circuits are where each LED shares a trace between +3V and GND. To connect LEDs in parallel, stick the LEDs next to each other between a (+) and a (-) track, like rungs on a ladder. You can turn the circuit on page **1-2** into a parallel circuit by just adding LEDs!



As the illustration above shows, all of the (+)'s go to the +3V pin, and all of the (-)'s connect to ground. This way the power source provides enough voltage ("height") for each LED. Since each LED needs at least 2.5V, connecting them in parallel to the +3V pin will turn all the LEDs on.

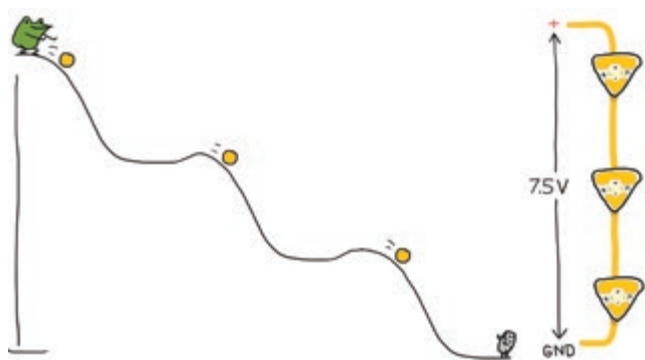


### Series Circuits

Series circuits are where LEDs are connected back to back, like beads in a chain. The (+) of one LED goes to the (-) point of the next.



Since the LEDs are in a line, the required voltage also adds up so we'll need a higher voltage power source. For example, since each LED sticker requires 2.5V to shine brightly, to turn on 3 LEDs in series we will want a 7.5V power supply. They may still glow with less voltage, but not as bright.



If you want to try out a series circuit, connect 3 LED stickers in series like this:



First, make a circuit with copper tape going the GND pin. Leave gaps for LEDs and stick LEDs so that they all point toward GND.

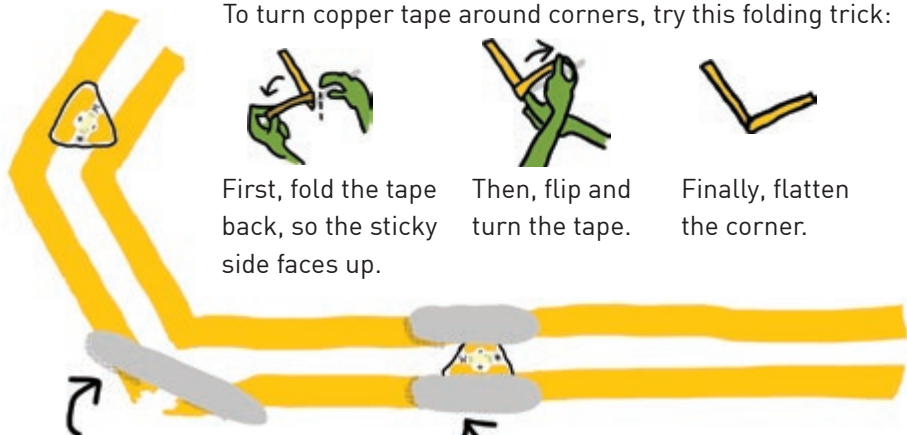
Then, clip a wire such as an alligator clip to the +5V pad of the Chibi Chip and connect the other end to the (+) side of different stickers in the series chain.

You'll see that the more LEDs there are between the +5V and GND, the dimmer the lights are, until they don't turn on at all!

# LET'S PLAY!

It's now time to design your own scene! Before we get started, here are some tips and tricks on how to use copper tape.

To turn copper tape around corners, try this folding trick:



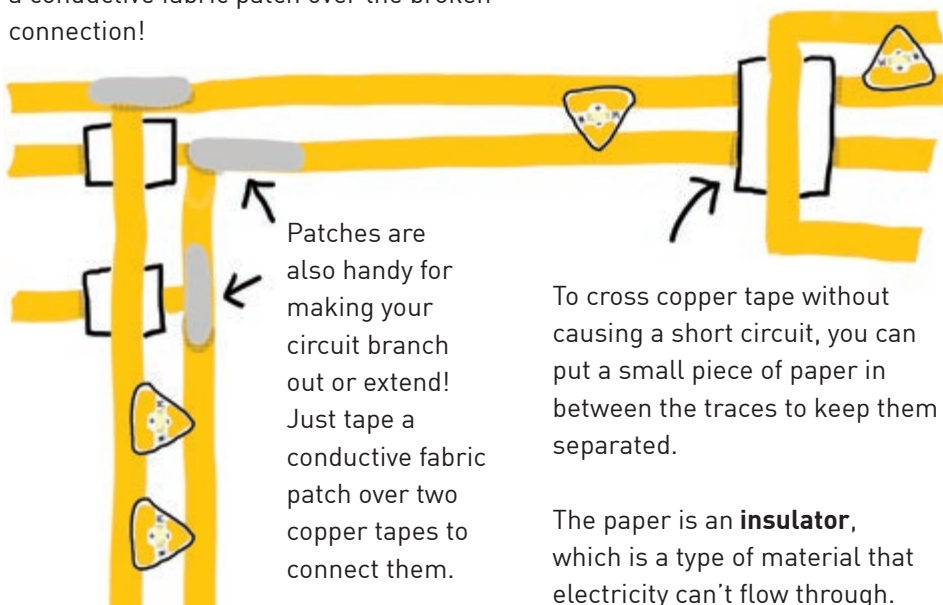
First, fold the tape back, so the sticky side faces up.

Then, flip and turn the tape.

Finally, flatten the corner.

**Conductive fabric patches** are like bandages for circuits. Accidentally torn copper tape can be rejoined by just sticking a conductive fabric patch over the broken connection!

If an LED is flickering, the connection can be improved by sticking a fabric patch over the LED and copper tape!



Patches are also handy for making your circuit branch out or extend! Just tape a conductive fabric patch over two copper tapes to connect them.

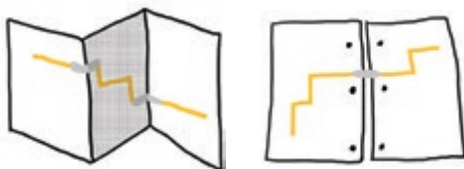
To cross copper tape without causing a short circuit, you can put a small piece of paper in between the traces to keep them separated.

The paper is an **insulator**, which is a type of material that electricity can't flow through.



COPPER TAPE WILL EVENTUALLY CRACK IF CREASED OR FOLDED TOO MANY TIMES!

Use conductive fabric patches to connect across creases or multiple sheets of paper! Unlike copper tape, fabric patches can be folded over and over again.



LED STICKERS WON'T WORK AS WELL IF THE UNDERLYING COPPER TAPE IS DIRTY OR WRINKLED!



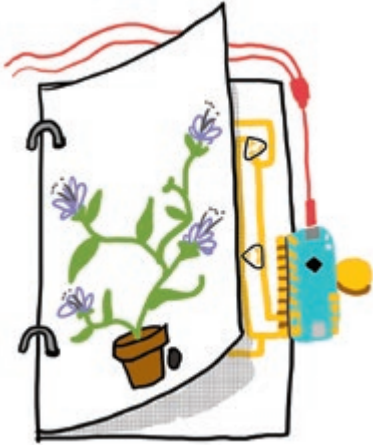
Hands should be free of oil and dirt before crafting with copper tape. Also, before applying LED stickers, wrinkled copper tape can be smoothed out by rubbing over it with the barrel of a pen or pencil!





Look on the facing page! It's a magical flower pot! What's growing? What's glowing?

Draw your own creature on page **1-15** and then design a circuit on page **1-17** to light it up!



REMEMBER TO WATCH  
OUT FOR SHORT  
CIRCUITS!



YAY!



NOPE.



YAY!









Design your own circuit on this page!



Start your circuit here, so you can clip on a Chibi Chip after you're finished!



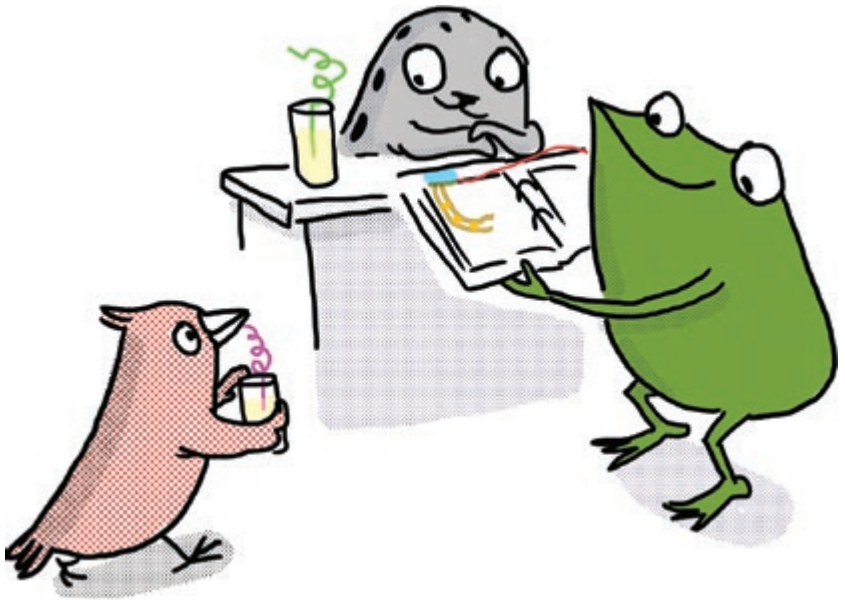
“Yay! This feels like magic!” Fern exclaimed.

Fern’s friend Carmen the bird walked past and admired the new creation.  
“Adding light to your art really does make it look magical,” she chirped.

Carmen, a clever programmer, thinks for a moment and asks, “Hey, do you want me to show you both how to make those lights blink?”

# Chapter 2:

## Code a Blink!



"Wow, if those lights could blink, then my drawings would really come alive!" said Fern enthusiastically. "Can I really do that?"

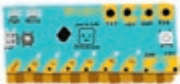
"We can control lights by writing programs, through a process called coding," said Carmen the bird. "Let's get started!"

# LET'S GET CODING!

In this activity we will upload a program to the Chibi Chip to make a light blink. The code starts as text in your browser, which then gets translated by a **compiler** and finally converted into a song that we play to the Chibi Chip. When the Chibi Chip hears the song, it decodes the song back into instructions on how to turn your LED light on and off.



## We will need:



Chibi Chip,  
mounted in a Clip



Programming  
and power cable



USB power  
(Chibi Book, laptop or  
wall plug)



Internet  
connection



A device with a web  
browser (phone,  
computer, or tablet):  
this is the programming  
device

1. Make sure we've got an Internet connection. Open a web browser and go to: **ltc.chibitronics.com**

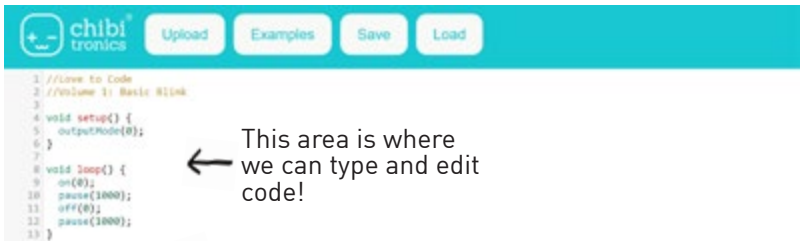
This brings us to the Love to Code (LTC) text programming editor. This is where we type the code that will be sent to the Chibi Chip.

The LTC code editor has a basic layout like this:

The **Upload** button sends code to our Chibi Chip.

**Examples** has code samples to help us get started!

Use **Save** and **Load** to stash code in the programming device, or download to your device for safekeeping & sharing.



This area is where we can type and edit code!

A sound banner will appear across the bottom of the screen when our code is uploading.

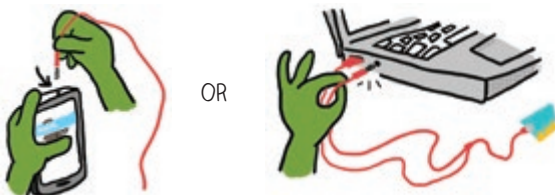
2. We start by opening **Examples → Love to Code Vol 1 → Basic Blink**



3. Plug the small end of the programming and power cable into the Chibi Chip. Then plug the big end into a USB power supply, such as a computer, wall power or the Chibi Book's built-in battery pack.



4. Plug the audio end of the programming and power cable into the audio jack of your programming device.



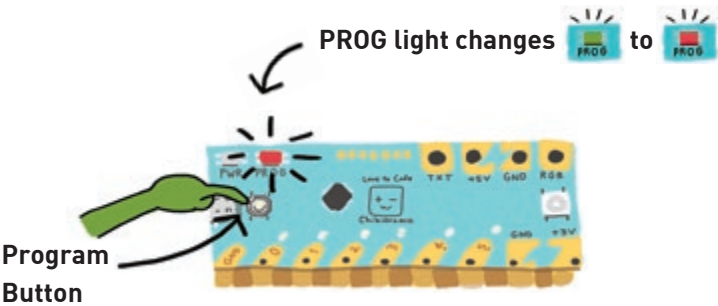
*Code a Blink!*

**2-3**

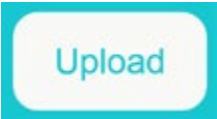
4. Turn the **volume up to the max** on the programming device and make sure it's not muted. Since code is uploaded to the Chibi Chip through the audio cable, we need to turn the volume up to 100% so the Chibi Chip can hear the code loud and clear!



5. Press and hold the program button on the Chibi Chip until the PROG light stays red. This lets the Chibi Chip know to listen for new code. Let go of the button after the light turns red.



6. Click **Upload** on the browser to upload the code and program the Chibi Chip.



A sound waveform should appear at the bottom of the page. Its presence means the code is currently being played to the Chibi Chip. If this bar does not appear, try refreshing the page and uploading again.





7. When the sound bar is done flashing, the red PROG light should turn back to green. This means the code is done uploading. If everything worked, the LED over pin 0 should now be blinking!



The uploaded code is saved to the Chibi Chip, and it will automatically reload itself and run whenever the Chibi Chip is powered on. This means once the Chibi Chip has been programmed, it no longer needs to be connected to a programming device.

HOW DID IT GO? DID THE LIGHT BLINK? IF NOT, TRY CHECKING THE FOLLOWING:



DID THE UPLOAD SOUND WAVEFORM ANIMATION APPEAR? IF NOT, TRY REFRESHING THE PAGE AND CLICKING UPLOAD AGAIN.



DID THE PROG LIGHT TURN STEADY RED BEFORE PROGRAMMING? IF NOT PRESS AND HOLD THE PROG BUTTON UNTIL IT TURNS RED AND TRY UPLOADING AGAIN.



CHECK OUT THE DEBUGGING SECTION AT THE BACK OF THIS BOOK FOR MORE TIPS!



*The original code that came with the Chibi Chip can be restored by uploading the example at:*

**Examples → Effects →  
Default Code - Light Cascade**

# DECODE THE CODE

How does the Chibi Chip blink the LED? It's actually following a set of instructions generated from our code! Let's take a look:

```
//Love to Code
//Volume 1: Basic Blink
```

The **comments** are notes to us about what the code does.

```
void setup() {
  outputMode(0);
}
```

The **setup** function happens once at the beginning, when you first turn on the chip.

```
void loop() {
  on(0);
  pause(1000);
  off(0);
  pause(1000);
}
```

The **loop** happens over and over after setup is done.

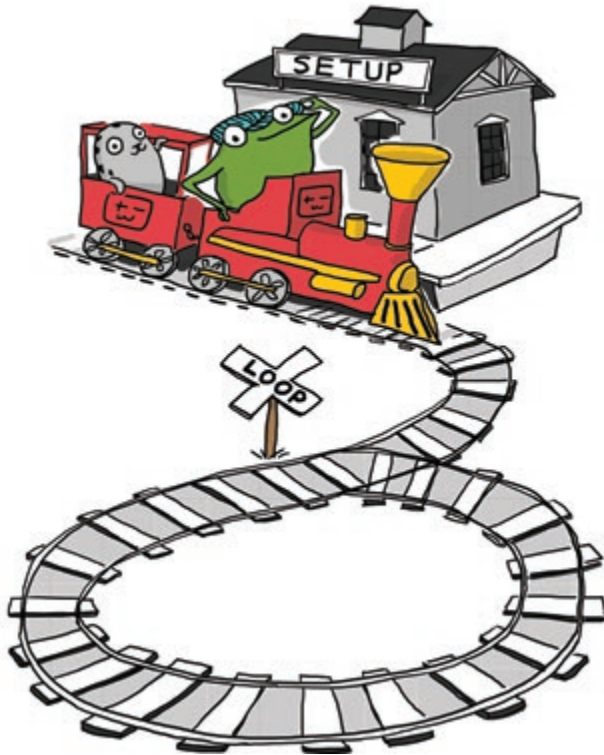
The **comments** describe what the code does. The Chibi Chip will ignore any notes that we leave in the code after a `//` (double slash). Adding comments is a good way to record and remember things.

The main code is split into two parts called **functions**: the “setup” and the “loop” functions. We label the functions first, using **void setup()** and **void loop()**, and then we put the function's code inside squiggly brackets `{` and `}`, after each label. The coded **statements** inside each function are step-by-step instructions for a Chibi Chip. Each statement must end with a “`;`” semicolon.

```
void setup() {
  // code here is used to setup the Chibi Chip
  // it's run only once at the beginning
}
```

```
void loop() {
  // code here is the loop or 'body'
  // and gets run over and over
}
```

The program runs like a train on a track. First it leaves the station, which is the **setup()**, and then runs around and around on the track, which is your **loop()**.



In other words, when first turned on, the Chibi Chip runs through the **setup()** function once. Then it will run the instructions coded in the **loop()** function over and over, until powered off or reset. The next time it turns on, the Chibi Chip will start with **setup()** again.



*The complete set of coded instructions is called a program. The Chibi Chip expects every program to have at least the **setup()** and **loop()** functions. If either is missing, the Chibi Chip won't know how to run the program!*

In the loop function we have three different kinds of instructions: **on()**, **pause()**, and **off()**. These three kinds of instructions allow us to blink a light by turning it on, pausing, then turning it off and pausing again.

The numbers (0 through 5) on the bottom of a Chibi Chip label the **pins**. Our code statements can control these pins.

For example, **on()** turns a pin on, and **off()** turns a pin off. The number inside the **()** parenthesis is called an **argument**, and specifies which pin to turn on or off. So, in our blink example, **on(0)** turns on pin 0 and **off(0)** turns off pin 0. The indicator LED above pin 0 also turns on and off, helping us confirm that the Chibi Chip understood our code statements.

**on(0)**

**Pin 0 ON**



**off(0)**

**Pin 0 OFF**



**pause()** tells the Chibi Chip to wait and not do anything. The number inside the **()** parenthesis tells how long to wait in milliseconds. In our blink code, **pause(1000)** instructs the Chibi Chip to wait 1000 milliseconds, or 1 second.

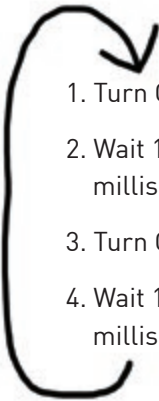


*Remember:  
there are 1000  
milliseconds in  
1 second!*

The Chibi Chip goes so quickly from one instruction to the next that if we don't tell it to pause, everything blurs together! The smaller the delay, the faster things will go.

Looking at the loop function again, this is what's happening:

```
void loop() {  
  on(0);  
  
  pause(1000);  
  
  off(0);  
  
  pause(1000);  
}
```

- 
1. Turn ON pin 0
  2. Wait 1000 milliseconds
  3. Turn OFF pin 0
  4. Wait 1000 milliseconds



...



...

After going through all four steps one-by-one, the Chibi Chip will go back to step 1 and repeat. And this is what makes the LED light blink!



# PLAY WITH THE CODE

Let's try writing our own code! Start by changing the pause times:

```
void loop() {  
  on(0);  
  pause(1000);  
  off(0);  
  pause(1000);  
}
```



Change this number to **500**  
in both pauses

Upload this new code to the Chibi Chip using the procedure starting at step 5 on page **2-4** and see what happens.



REMEMBER TO PRESS THE  
PROG BUTTON ON YOUR CHIBI  
CHIP BEFORE CLICKING THE  
UPLOAD BUTTON, SO IT KNOWS  
TO LISTEN FOR CODE!

The light over pin 0 should now be flashing faster! Try some other numbers for the pause between on and off statements. Can you make it blink super slow?



We can also create more complex patterns by adding more `on()` and `off()` statements too. For example, try this:

```
void loop() {  
  on(0);  
  pause(1000);  
  off(0);  
  pause(500);  
  on(0);  
  pause(500);  
  off(0);  
  pause(500);  
  on(0);  
  pause(500);  
  off(0);  
  pause(1000);  
}
```

This makes a long-short-short blink pattern. Try making your own rhythms! How about a heartbeat?



THE CHIBI CHIP'S COMPILER IS SUPER PICKY ABOUT PUNCTUATION:

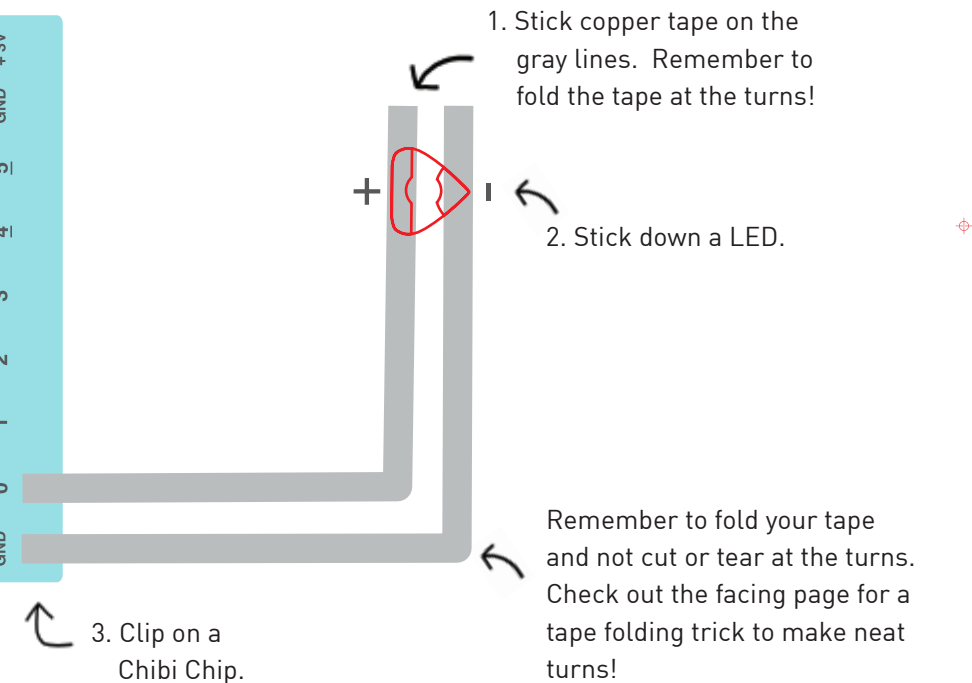


- 1) IT IS CASE-SENSITIVE (SO "On" IS NOT THE SAME AS "on")
- 2) DON'T FORGET THE SEMICOLONS (;) AT THE END OF EACH STATEMENT.

WRITE THE CODE EXACTLY AS SEEN ABOVE, OTHERWISE THE CHIBI CHIP WON'T UNDERSTAND OUR CODE AND IT WON'T UPLOAD!

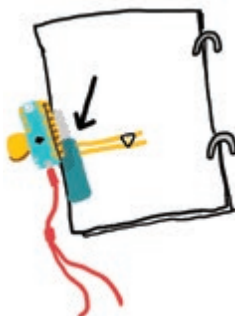
# PROGRAM THIS CIRCUIT

We can use a program to control a circuit of our own design. Try it with this template!



See how when you connect the LED circuit to pin 0, the LED also blinks just like the indicator light on the Chibi Chip? That's because the pin sends power to both the light on the Chip as well as to any LEDs in a circuit connected to that pin.

Try clipping GND and pin 0 to the "Turn on a Light" circuit on page 1-2, and see that blink too!





# DRAW HERE!



To turn copper tape around corners without tearing, try this folding trick:



1. Fold tape back, so the sticky side faces up



2. Flip and turn your tape



3. Flatten the corner!

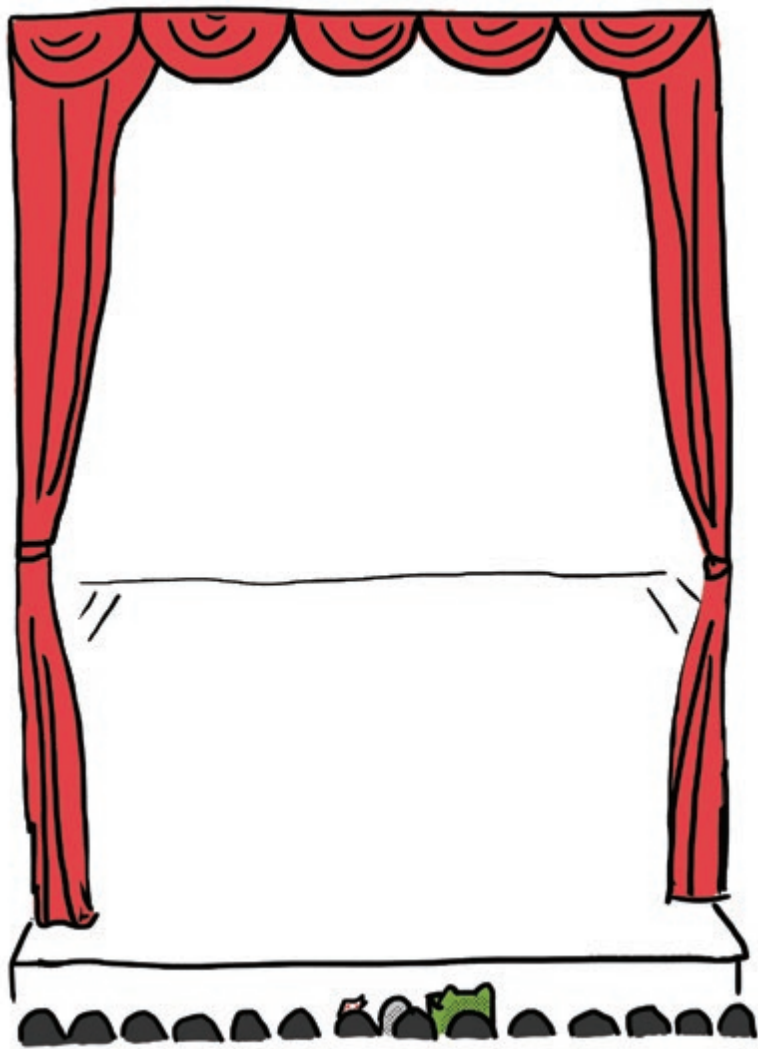


If the tape tears, don't worry! Fix it with a conductive fabric patch!

REMEMBER TO SAVE YOUR CODE AS YOU GO!



Draw something on page **2-13**, to see the shadow of your drawing appear when the light blinks on!



# DECODE THE CODE

Now we're ready to program more pins! To do that we have to start with our setup code.

Recall that the setup code runs exactly one time after being powered on to make sure the Chibi Chip has everything it needs before trying to run our program. It's a bit like gathering up ingredients before cooking. That's why it's called "setup."



Let's take a closer look at the **setup()** in our blink example:

```
void setup() {  
    pinMode(0);  
}
```



**pinMode()** sets up the pin to be in **OUTPUT** mode, which means it will turn things on and off, like a LED light or a motor. The number inside the "()" parenthesis is an argument that tells the Chibi Chip which pin to set.

In our example, **pinMode(0)** sets pin 0 to be an **OUTPUT** so that pin 0 can turn things like our LEDs on and off.

A pin can also be an **INPUT**, which is something that takes in information from the circuit, like a switch or sensor. We will learn more about inputs in the next chapter!

# PLAY WITH THE CODE

Ready to add more pins to our program? Let's try to make pin 3 blink! First, let's add one more `outputMode()` statement to the setup, so the Chip knows how to configure pin 3 before we use it:

```
void setup() {  
  outputMode(0);  
  outputMode(3);  
}
```

Now, add more `on()` and `off()` statements to the loop to make our new pin do stuff, using "3" as the argument inside the parenthesis:

```
void loop() {  
  on(0);  
  pause(1000);  
  off(0);  
  pause(1000);  
  on(3);  
  pause(1000);  
  off(3);  
  pause(1000);  
}
```

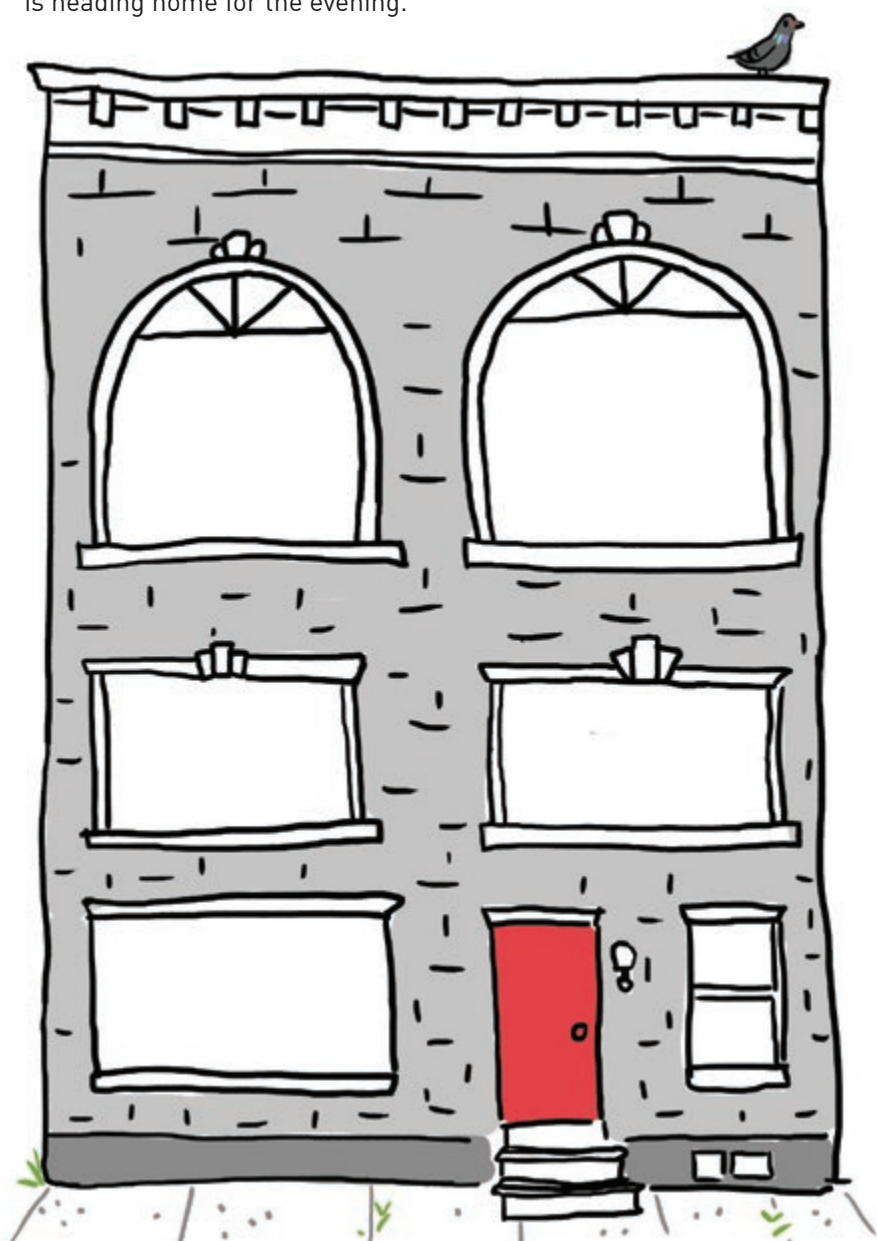


Upload this new code and you will see pin 0 and pin 3 both blinking on and off! To make the lights blink together, try this code:

```
void loop() {  
  on(0);  
  on(3);  
  pause(1000);  
  off(0);  
  off(3);  
  pause(1000);  
}
```

Even though pin 3 and pin 0 are controlled with statements on different lines, the Chibi Chip goes from one step to the next so fast that they seem to happen at the same time! Try adding or modifying `pause()` statements to make your own patterns.

It's starting to get dark in Fern, Sami, and Carmen's town, so everyone is heading home for the evening.

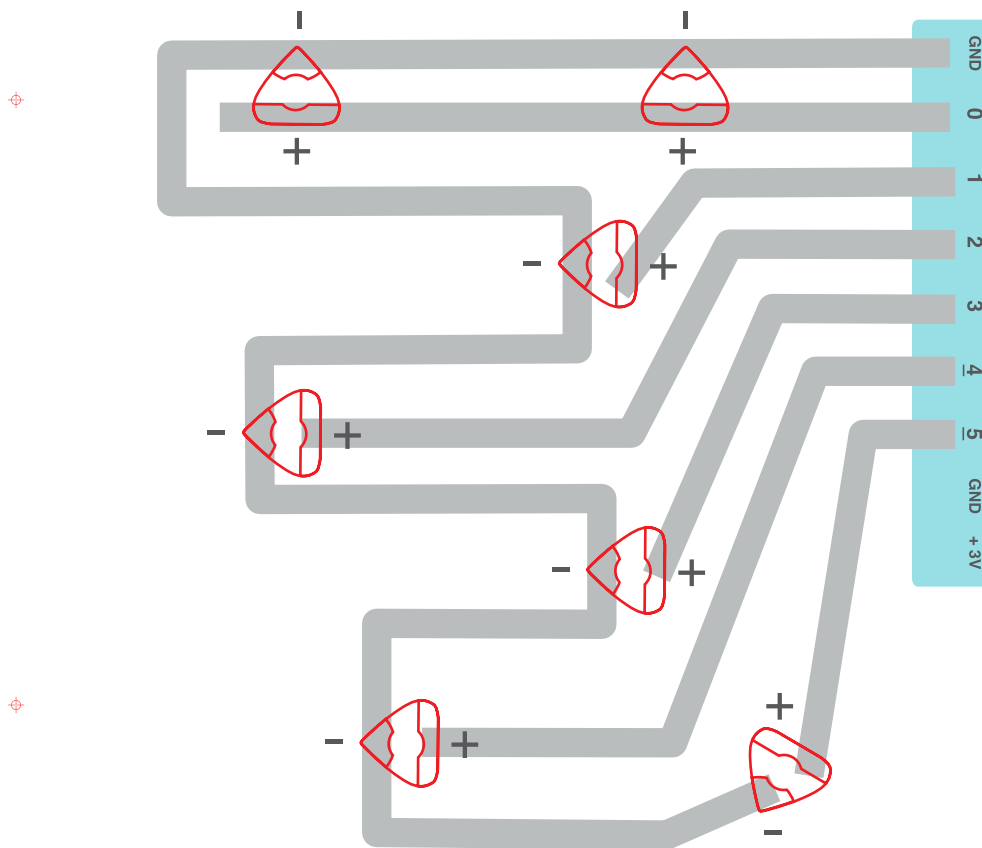


Draw what everyone in the apartment building is doing when they come home. Their shadows will show up in the windows when the LED stickers on the facing page turn on!



# PROGRAM THIS CIRCUIT

Just like before, we can clip the Chibi Chip to a circuit and control the lights using a program. Here's a circuit where all of the pins are connected to LEDs, and the LEDs are placed in spots that will light up the windows on the page **2-17**. Now we're ready to play with all the pins!

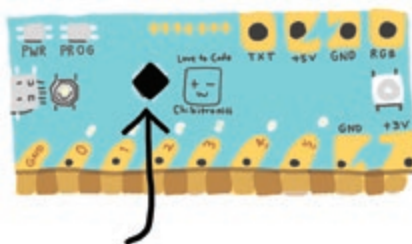


Can you get all 6 of the pins to blink in a pattern? If you get stuck while coding, check out the Six Pin Blink in the Examples menu to help you get started!



# DOWN THE RABBIT HOLE: MICROCONTROLLERS

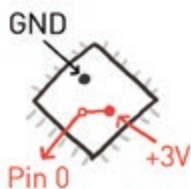
So how does a program actually make lights blink? There's a little black diamond on the Chibi Chip called a **microcontroller**: this is a tiny computer.



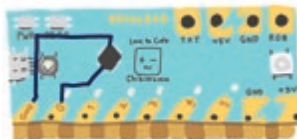
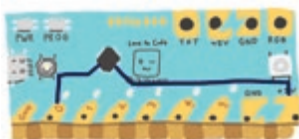
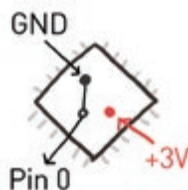
The microcontroller runs instructions coded by your program. Some of these instructions change how the pins behave.

Unlike the +3V and GND pads, which are permanently wired to +3V and GND, all the numbered pins can be programmed by statements in our code to connect to +3V, GND, or neither. All of this happens inside the tiny microcontroller!

`on(0);`

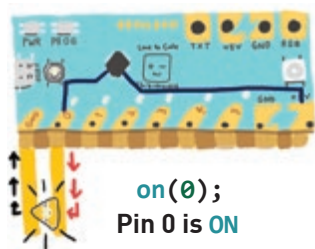


`off(0);`

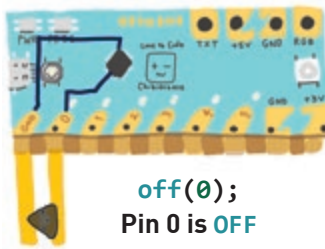




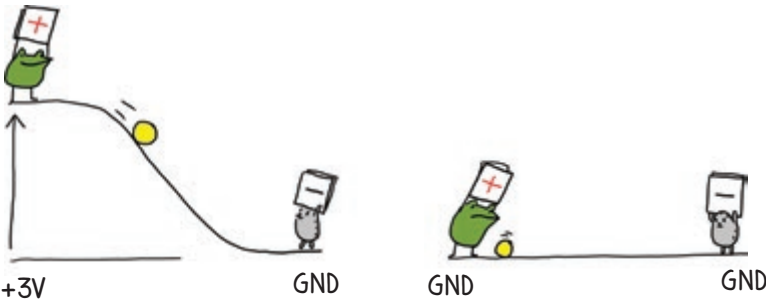
When we connect an LED to the Chibi Chip, we connect the (-) side to GND, and the (+) side to a pin (in this example, pin 0). Our code can configure the (+) side of the sticker to be either +3V or GND by using statements that reference pin 0.



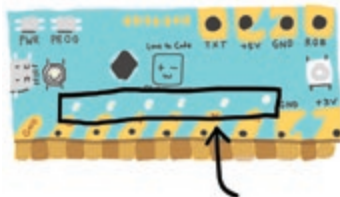
When pin 0 is **ON**, it's connected to +3V by the microcontroller. The LED turns on because the (+) end is connected to +3V through pin 0, and there is a voltage difference, allowing current to flow.



When pin 0 is **OFF**, it's connected to GND by the microcontroller. The LED turns off because both the (+) and (-) ends are connected to GND, and there is no voltage difference.



There's an indicator light built into the Chibi Chip for each numbered pin.



These indicator lights let us know what a program is doing without having to first build a circuit. They're really handy for testing programs and debugging circuits.

# LET'S PLAY!

Now it's your turn to design your own circuit and light up the night sky (turn to page **2-24** for a preview).

+3V  
GND  
5  
4  
3  
2  
1  
0  
GND



Above is a Chibi Clip template to help you get started. Use as many pins, and whatever circuit shape you like! If you're not sure how to start, take a look back at the circuit on page **2-19** for reference.

Remember: LEDs that are connected to the same pin will come on at the same time. Since the Chibi Chip has 6 programmable pins, you can control up to 6 separate groups of lights with one Chibi Chip!



One method for designing a circuit is to:



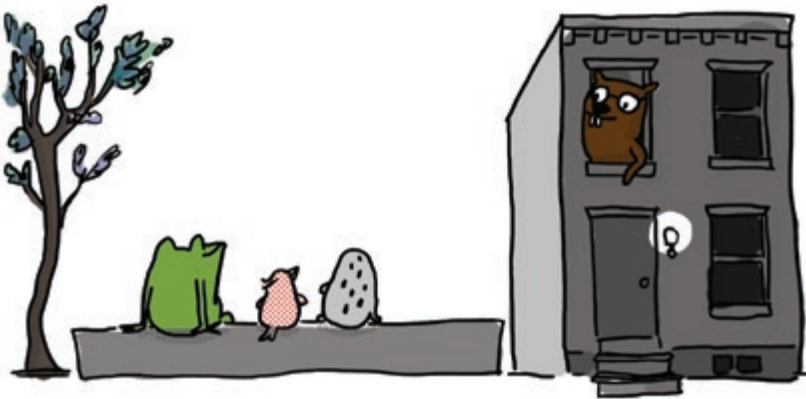
1. Draw dots where you want the lights to go.



2. Lay copper tape going from GND to each of these dots.



3. Tape copper traces from the programmable pins to the dots, and add your LED stickers!



Edith the beaver overheard her three friends enjoying the light show and looked out her window.

“How am I missing out on this party?” cried Edith, and she quickly rushed out to join the fun.

# Chapter 3:

## Add a Switch!



"I love building things! I can build switches out of paper and copper tape," Edith the beaver quipped. "If we add a switch to our circuit, then we can interact with the lights!"

Thwacking her tail enthusiastically, as if smacking a giant switch, Edith continued, "When someone presses the switch, we can make something happen, like blinking the lights."

"Amazing!" exclaimed Fern. "Show us how!"

# PROGRAM A SWITCH

In this activity we'll add interactivity to our projects by using a switch to turn on a light. We can create different types of switches out of paper and use our switch to trigger light patterns.

## We will need:



Chibi Chip



cable



USB Power



Internet connection



copper tape



Chibi Light  
LED Sticker



Web browser  
(phone, computer, or  
tablet): this is your  
**programming device**



scissors

1. Turn the page and make the switch circuit template on page **3-4**.
2. Upload the **Basic Switch** example code to the Chibi Chip. Go to **Examples → Love to Code Vol 1 → Basic Switch**

```
1 // Love to Code
2 // Volume 1: Basic Switch
3
4 int pressed = 0;
5
6 void setup(){
7   pinMode(0);
8   pinMode(5);
9 }
10
11 void loop(){
12   pressed = read(5);
13   if (pressed == 1){
14     on(0);
15   } else {
16     off(0);
17   }
18 }
19 }
```

3. The flap at the bottom of page **3-4** is the switch. Press it and see the LEDs on pin 0 and pin 5 turn on.



SWITCH NOT WORKING?

FIRST, CHECK IF THE CODE WAS UPLOADED TO THE CHIP:



DID THE PROG LIGHT TURN STEADY RED BEFORE PROGRAMMING? IF NOT, PRESS AND HOLD THE PROG BUTTON UNTIL IT TURNS RED AND TRY UPLOADING AGAIN.



MAKE SURE THE VOLUME IS ALL THE WAY UP.



DID THE UPLOAD SOUND WAVEFORM ANIMATION APPEAR? IF NOT, TRY REFRESHING THE PAGE AND CLICKING UPLOAD AGAIN.

IF THE CODE LOOKS GOOD, LET'S TRY CHECKING OUR CIRCUIT:



CHECK YOUR ALIGNMENT. MAKE SURE THE GOLDEN METALLIC STRIPES ON THE CHIBI CHIP ARE LINED UP AND MAKING SOLID CONTACT WITH THE COPPER TAPE.



IS THE LED OVER PIN 0 TURNING ON, BUT NOT THE LED IN YOUR CIRCUIT? PRESS HARD ON THE LED STICKER'S METAL PADS FOR A STRONGER CONNECTION.

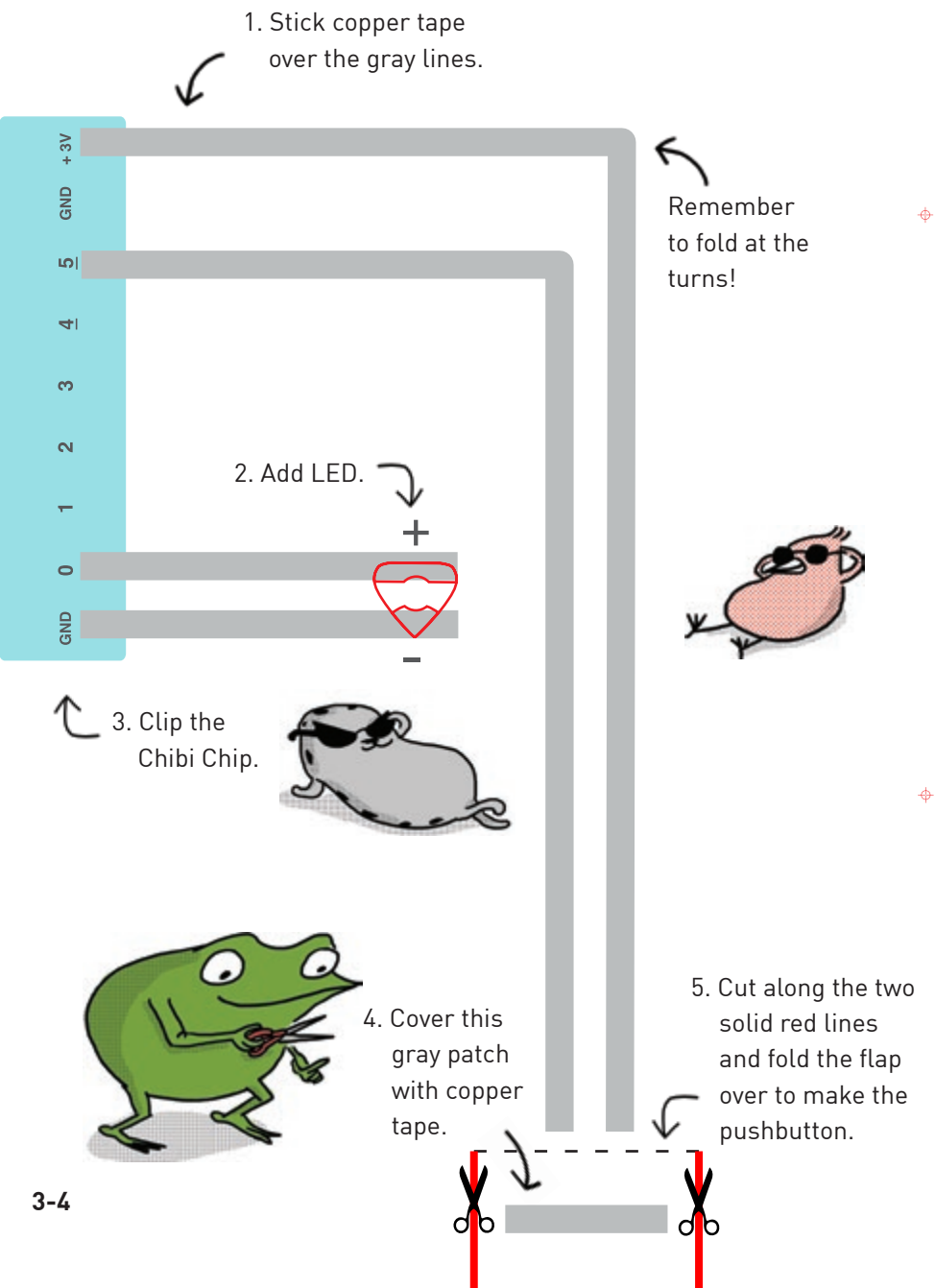


CHECK FOR TEARS. PATCH TEARS WITH CONDUCTIVE FABRIC TAPE.



STILL NOT WORKING? CHECK OUT THE DEBUGGING SECTION IN THE BACK OF THIS BOOK.

# PUSHBUTTON SWITCH TEMPLATE







Almost done!



*Add a Switch!*

**3-5**

What happens when you  
press on Edith's tail?



# DECODE THE CODE

Let's take a look at how the switch example program works. Here's our code:

```
// Love to Code
// Volume 1: Basic Switch
```

```
int pressed = 0;
```

Make a variable named **pressed** to store whether the switch is on or off.

```
void setup() {
  pinMode(0);
  pinMode(5);
}
```

Set pin 0 to output mode for turning on the LED.

Set pin 5 to input mode for reading the switch.

```
void loop() {
  pressed = read(5);
```

Read from the input pin (pin 5) to see if the switch is pressed.

```
  if(pressed == 1) {
    on(0);
  } else {
    off(0);
  }
}
```

Turn on or off the light depending on whether the switch is pressed.

```
}
```

"Don't worry, I'll explain this all," said Edith.



We start by creating a **variable** which is a text name that stores a number. This comes in handy for storing our switch status so we can use it for controlling our pins later. We create variables like this:

```
int pressed = 0;
```

This is the name of our variable      ↗      ↖      This is the initial value we set our variable to

The **int** right before our variable stands for **integer**. On a Chibi Chip, it means that the variable can be set to a whole number, like -42 or 18000, but not to a decimal like 3.14. The whole number must be no less than -2147483648 and no greater than 2147483647, and typed without commas.

We named our variable **pressed** because it tells us whether the switch is pressed or not, but we can name a variable any single word that helps us remember what it's for!

We set **pressed** equal to 0 at the beginning as a default, or initial, value. As long as we don't change **pressed** from 0, every time we write **pressed** in the code, it is the same as writing the number 0.



The cool thing about variables is that we can update the stored number. We do this by setting it to another value with the **=** equal sign. For example, **pressed = 1** changes **pressed** to equal 1 instead of 0. We use this technique to update our **pressed** variable based on if the button is pressed or not.



In our `loop()` we start by inspecting the voltage on pin 5 (the one connected to our switch) using the `read()` function, and storing the result in `pressed`:

sets the value

`pressed = read(5);`

variable that we save  
our voltage reading to

pin that we are  
reading from

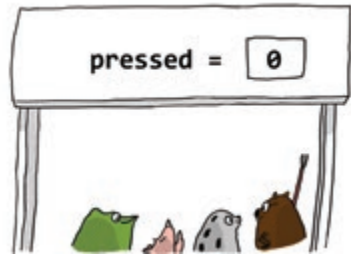
`read()` gives a `1` if the switch is pressed and a `0` if the switch is not pressed. Because our circuit has a switch connected to pin 5, we use `read(5)` to specify which pin to check.



pin 5 connected to +3V →  
`read(5)` is 1 →  
`pressed = 1`



pin 5 not connected to anything →  
`read(5)` is 0 →  
`pressed = 0`



We program our lights to turn on or off depending on the switch's value. We do this using an `if()` statement. Here is the general structure of the `if()` statement:

```
if(condition) {  
    // Option 1 code: runs when  
    // condition is TRUE  
} else {  
    // Option 2 code: runs when  
    // condition is FALSE  
}
```

An `if()` statement allows us to change the behavior of the circuit based on the answer to a question. This question is called a **condition**.

If the condition is **TRUE**, then the **Option 1 code** between the first set of curly braces `{}` runs. If the condition is not true, or **FALSE**, then the **Option 2 code** between the second set of curly braces `{}` runs.

It's like train tracks that split into two paths, and which path the train takes depends on whether the condition is true or false.



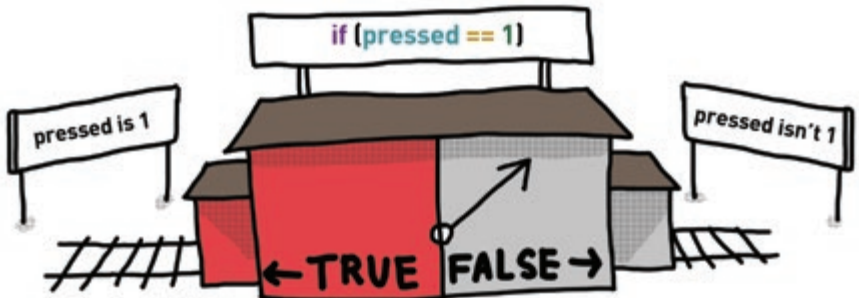
In our code we ask “is it true that **pressed** is **1**?” using the condition statement **pressed == 1**.

↙ Condition statement: is **pressed** equal to **1**?

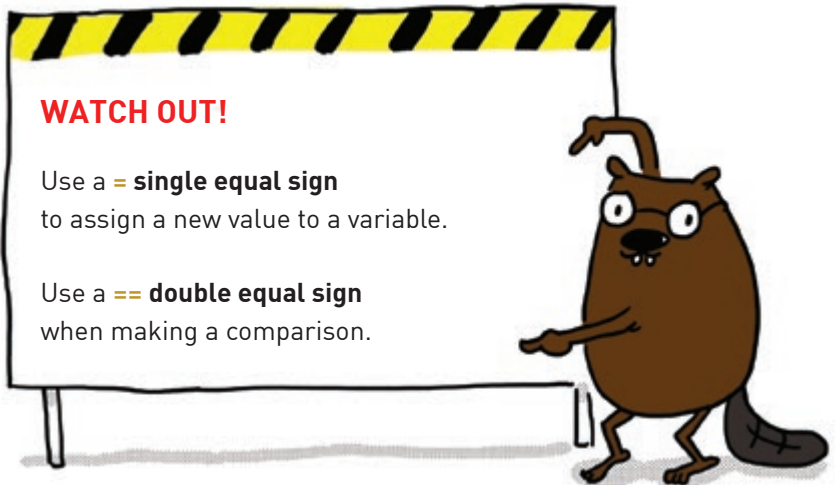
```
if(pressed == 1){  
  on(0);  
} else {  
  off(0);  
}
```

If yes, do this: turn **ON** pin 0

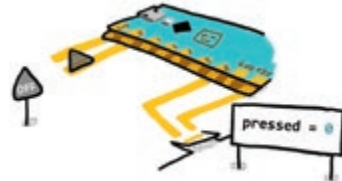
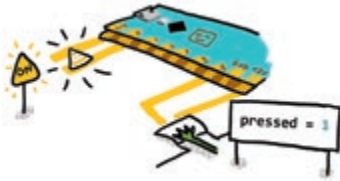
Otherwise, do this: turn **OFF** pin 0



Condition statements are written as comparisons. The **==** double equal sign asks, “Are the left and right sides are equal?”



To summarize: if our switch is pressed, then pressed is **1**, our condition is **TRUE** and we turn **ON** pin 0. If our switch is not pressed, then pressed is **0**, our condition is **FALSE**, and we turn **OFF** pin 0.



Switch is pressed →  
**pressed = 1** →  
condition is **TRUE** →  
turn **ON** pin 0

Switch is not pressed →  
**pressed = 0** →  
condition is **FALSE** →  
turn **OFF** pin 0





# PLAY WITH CODE

Whew, we've covered a lot! Let's try it out and play with some code! We can program the light to do other things when the switch is pressed. All we have to do is change what's inside the `if()` statement. Let's start by changing the code in the `loop()` function:

```
pressed = read(5);
```

```
if(pressed == 1) {
```

```
  off(0);
```

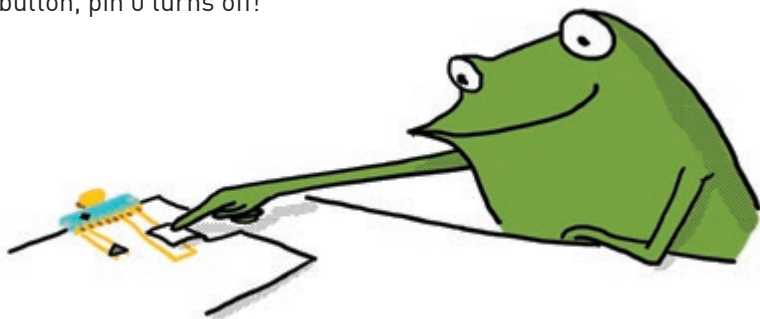
```
} else {
```

```
  on(0);
```

```
}
```

← `on(0);` and `off(0);`  
trade places

Upload this new code to a Chibi Chip and clip it to the example switch circuit on page 3-4. Notice that pin 0 is on now, and when we press the button, pin 0 turns off!



We can make the pushbutton activate any light sequence by just putting the code for the sequence between the `{` and `}` braces:

```
pressed = read(5);
```

```
if(pressed == 1) {
```

```
  // code here runs when switch is pressed
```

```
} else {
```

```
  // code here runs when switch is NOT pressed
```

```
}
```

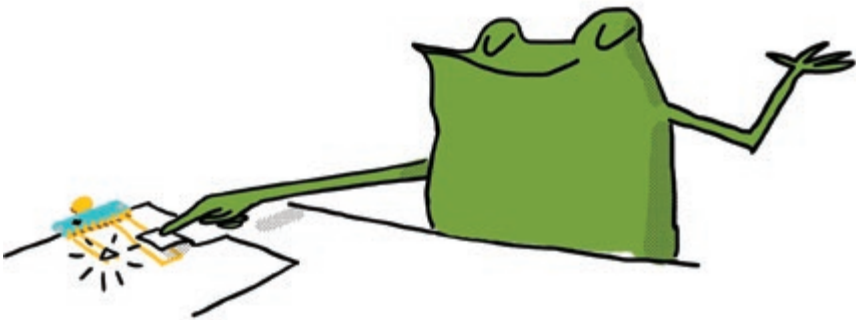
Note that the `else {}` statement is optional. The contents inside `{}` can be left blank, or the entire `else {}` clause can be deleted if it's not needed.

For example, try putting this blink sequence inside the `if()` statement and see what happens!

```
pressed = read(5);

if(pressed == 1) {
  on(0);
  pause(500);
  off(0);
  pause(500);
  on(0);
  pause(500);
  off(0);
  pause(500);
} else {
  on(0);
}
```

Voilà! We've made a trigger button! Now every time we press the button, it triggers the blink animation inside the `if()` statement. Try programming your own patterns for the trigger button!



# CODE CLEANUP

At this point our code is getting pretty complex. It may even be confusing to remember what we mean. Time to clean up our code and make it more readable with **comments** and **variables**!



**Comments** are human-readable text that explain what the code does without affecting how the code runs. Like labels or sticky notes, their purpose is to make the code much easier for us and our friends to read! Go to your programming device and add some comments to our code:

```
/* This code turns on a light connected  
   to pin 0 when a switch on pin 5  
   is pressed.  
*/
```

```
// make a variable for remembering switch values  
int pressed = 0;
```

```
void setup(){  
  pinMode(0); // make pin 0 output  
  pinMode(5); // make pin 5 input  
}
```



Anything after a **//** double slash on the same line becomes a comment and changes to **brown**. If we need to leave a long comment that takes up multiple lines, we can save time by putting the entire comment text between **/\*** and **\*/**.

Comments are also handy for keeping code snippets in a program that aren't actually run. This is useful when we're testing out different options and don't want to delete the original code. Just put the unneeded code between the `/*` and `*/`, like a regular comment. This is called **commenting out** code:

**Before:**

Original code makes the light blink when the switch is pressed.

```
if(pressed == 1) {  
  on(0);  
  pause(500);  
  off(0);  
  pause(500);  
} else {  
  off(0);  
}
```

**After:**

New code comments out blink code, so the light stays on when the switch pressed.

```
if(pressed == 1) {  
  on(0);  
  /* pause(500);  
    off(0);  
    pause(500);  
  */  
} else {  
  off(0);  
}
```

Another useful tool for cleaning up code is **variables**. So far we've used variables to save information from our switches. However, variables are useful for replacing numbers with more informative text names. Using variables instead of values help make code easier to understand and maintain:

**Before:**

What is connected to these pins? Nobody knows!

```
void setup(){  
  outputMode(3);  
  outputMode(4);  
}
```



**After:**

When the LED pins are named as variables, everyone knows what the pins are for!

```
int starLED = 3;  
int flowerLED = 4;  
  
void setup(){  
  outputMode(starLED);  
  outputMode(flowerLED);  
}
```

Using variables as labels also lets us quickly change a value in multiple places with a single edit. For example, it lets us change many pin numbers at once by editing just one line of code, instead of having to find and change every line that uses that pin number.

#### Before:

Change every line that uses pin 2

```
void setup(){  
  pinMode(2);  
}
```

```
void loop(){  
  on(2);  
  pause(1000);  
  off(2);  
  pause(500);  
  on(2);  
  pause(1000);  
  off(2);  
  pause(200);  
}
```

#### After:

Change only once at the variable declaration of **myLED**.

```
int myLED = 2;  
  
void setup(){  
  pinMode(myLED);  
}
```

```
void loop(){  
  on(myLED);  
  pause(1000);  
  off(myLED);  
  pause(500);  
  on(myLED);  
  pause(1000);  
  off(myLED);  
  pause(200);  
}
```



In the left-hand code, we've highlighted all the places we'd have to update if we changed the connection of the LED from pin 2 to something else. By using the **myLED** variable in the right-hand code, we can update the pin with a single edit to our code. Where else could we have used a variable?



These tips will help make your code more readable and easier to debug. We will begin to use more variables and comments in our example code too, so watch out for them!

# PAPERCRAFT SWITCHES

Now that we've cleaned up our code, it's time to get messy with craft materials! In the next several pages, we'll go through step-by-step instructions on how to build a few types of papercraft switches. But before we dig in, here is a brief introduction to the ideas we'll cover.



## Paper Clip Switch Holder

Use a paper clip to hold down a switch that's been built at the edge of a page. The switch will stay on even when it's not being pressed! The switch circuit we built on page 3-4 is perfect for use with a paper clip.

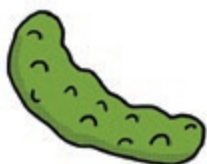
## Press-the-Flap Switch

Use a flap of paper with copper tape on it to close a circuit anywhere.

This way, we can make switches anywhere on the page, not just on the edge or corner of the paper!

To make sure the switch doesn't accidentally press itself and turn on, insert a bit of paper or foam tape as a spacer between the flap and the circuit.

We can also use fun shapes for the flap, like some of the ones below!





### Wind-Sensing Switch

Make a flap switch that has a thin stem, like a leaf shape, to make a wind-sensing switch!

This shape will catch the wind, like a sail, and the stem makes it flexible enough to press down and close the circuit when blown on.



### Pop-Up Switches

Circuits and switches don't have to be flat! We can put copper tape at the end of a long strip of paper, and curl or fold it so that it zigzags and pops off the page.

These flexible switches can stretch to close different circuits or they can be clipped in place.



## Pocket Character Switch

This switch comes in two parts: a character and a pocket. The character is a loose piece of paper that has copper tape pasted on a flap that's been rolled around to the back side. To hold the character, we place a pocket over the switch gap in our circuit.



Front



Back

When the character is placed inside the pocket, the copper tape on the character closes the switch gap and completes the circuit!



A character can work as an ON/OFF switch too! Just flip the character over so that the copper tape faces away from the circuit. That way, the switch is off even when the character is in the pocket!





# SWITCH CRAFT GALLERY

The next few pages are a gallery of switches for us to craft! The artworks in this gallery are all incomplete and need our help to come to life. Once we're done, you can refer back to these pages for inspiration in your own projects!



All of the circuits use the **Basic Switch** example code so make sure that's loaded onto the Chibi Chip before starting. Then, complete the circuit template in each frame and follow the craft directions for the cutouts.

## You'll need:



Chibi Chip



cable



copper tape



scissors



USB Power

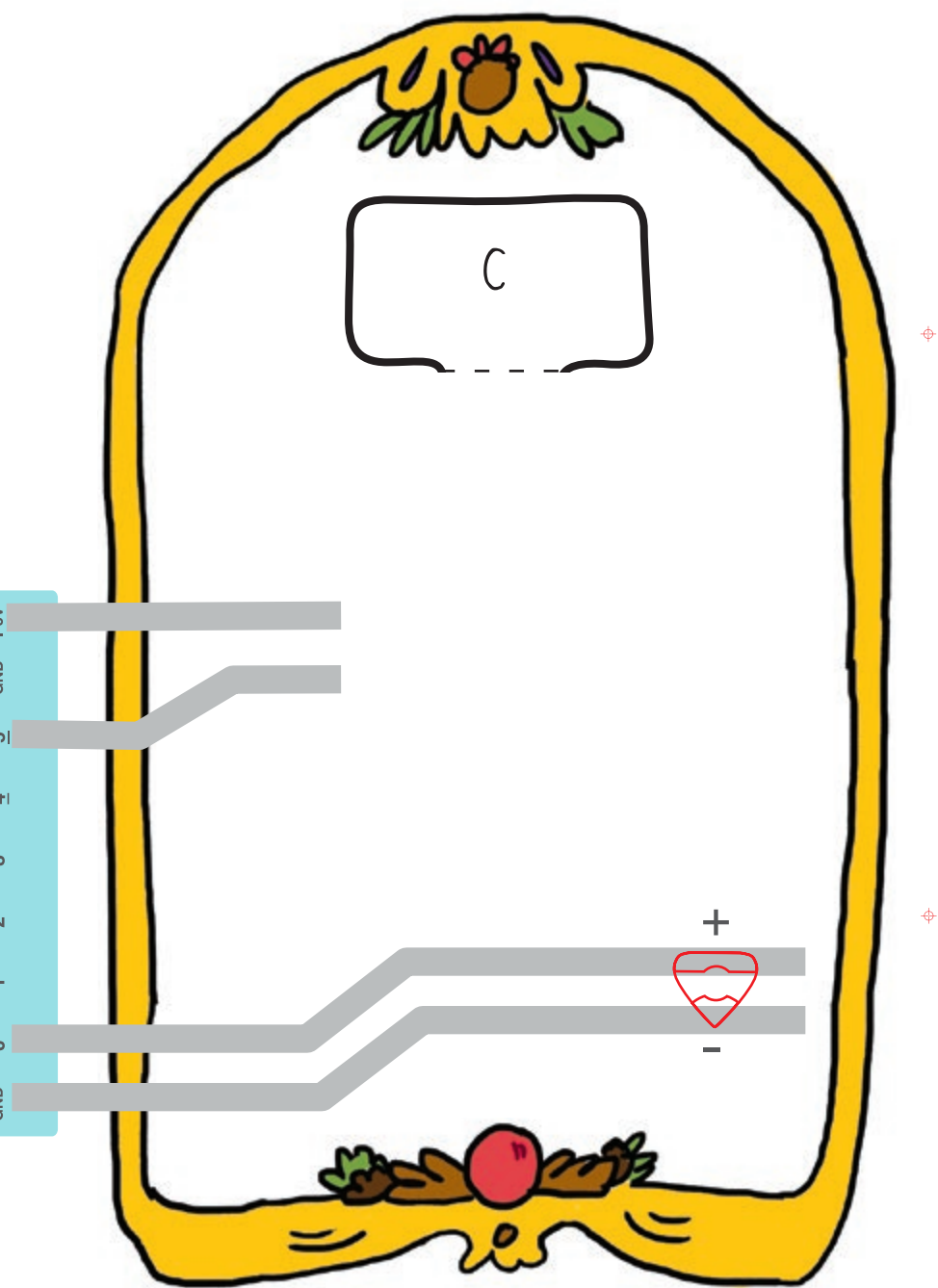


Chibi Light  
LED stickers



non-conductive tape

What glows when the pointer reaches the switch? Craft the switch by following the instructions on page 3-23!



POP-UP SWITCH

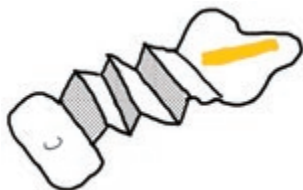
## Make a Pop-Up Switch



1. Cut out the pointer, using the red outline on the back side of this page.



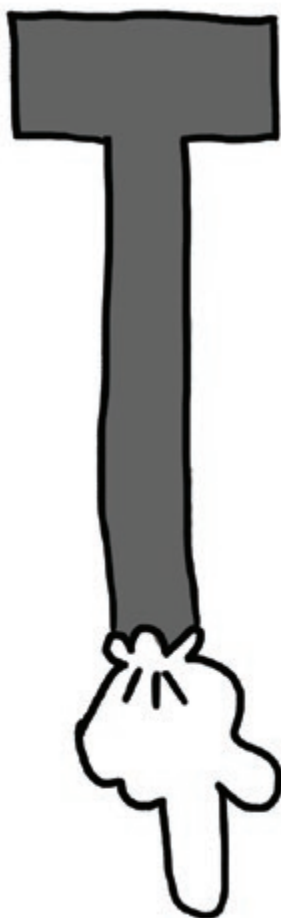
2. Add copper tape over the gray line.

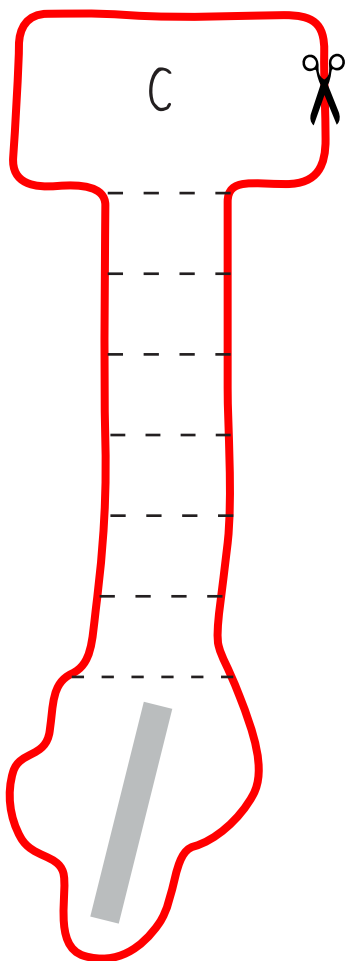


3. Fold along the dotted lines so that the pointer becomes a zig zag.



4. Tape the flap marked C over the footprint marked C on page 3-22.





Here's what the finished pop-up switch looks like! Stretch the pointer finger so reaches the switch contacts, and then press to activate the switch. When we let go, the pointer springs back up!



## Make a Press-the-Flap Switch



1. Cut out the boot, using the red outline on the back side of this page.



2. Add copper tape over the gray line.

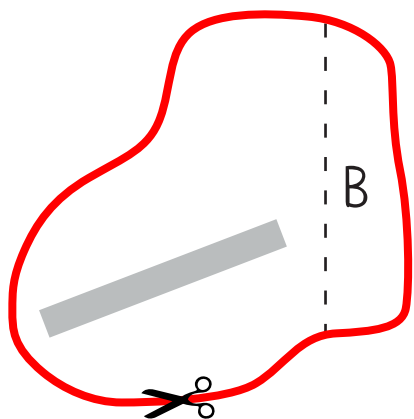


3. Fold along the dotted line.



4. Tape the flap marked B over the footprint marked B on page 3-27.

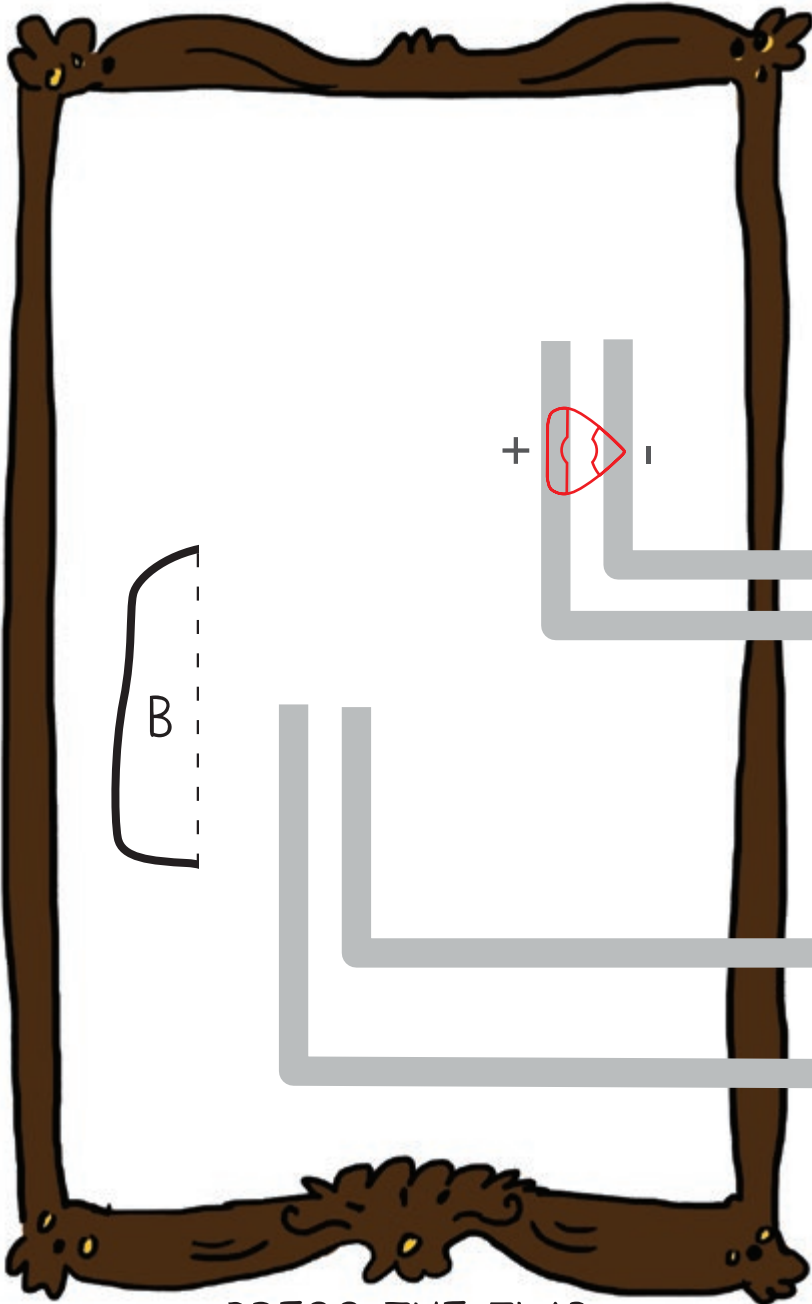




Here's what the finished press-the-flap switch looks like! Press the boot to close the switch.

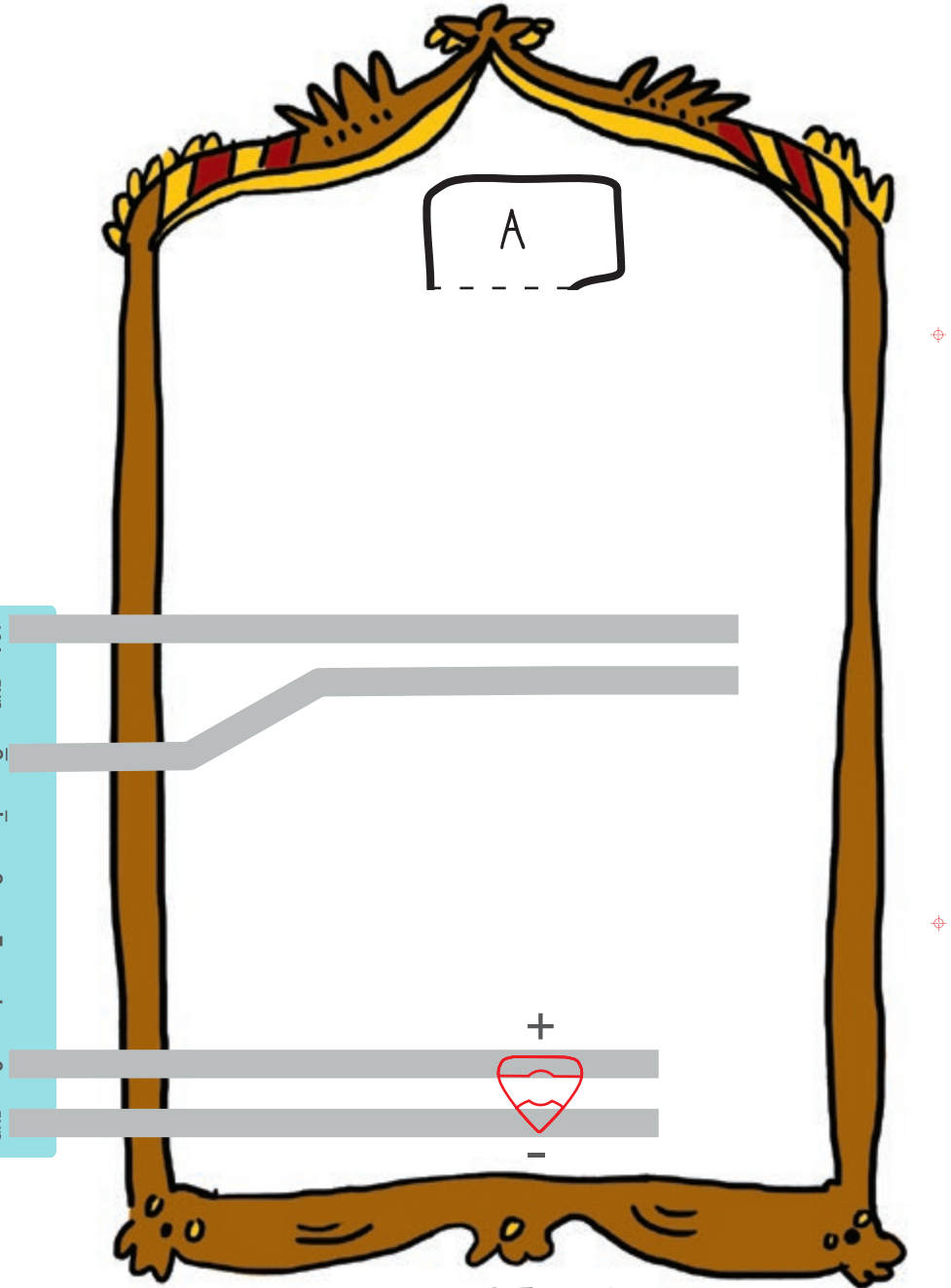


What glows when we press on the boot? Make it below!



PRESS-THE-FLAP

What glows when we blow on the flower? Craft the flower by following the instructions on page **3-29**!



WIND SENSOR



## Make a Wind Sensor



1. Cut out the flower, using the red outline on the back side of this page.



2. Add copper tape over the overlapping gray lines on the back side of the flower.

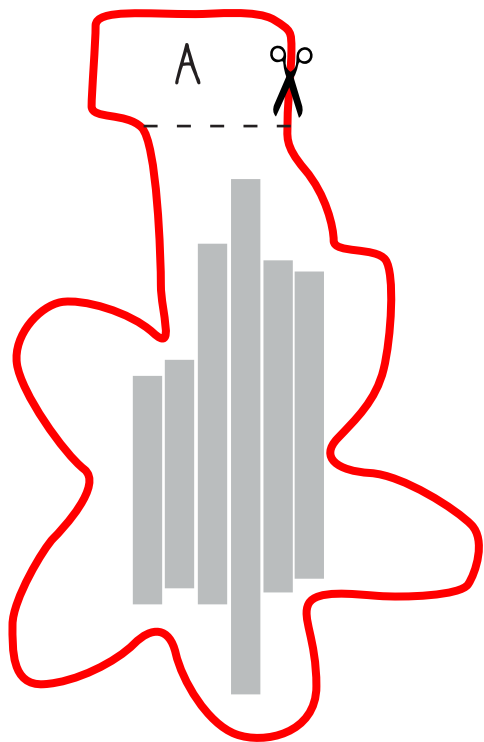


3. Fold along the dotted line.



4. Tape the flap marked A over the footprint marked A on page **3-28**.





Here's what the finished wind sensor looks like! Blow on the flower to activate the switch. Try adjusting the position of the flower slightly to improve the contact between the copper tape on the flower and the paper.

You can also press on the flower to operate the switch as well!



## Make a Pocket Character Switch



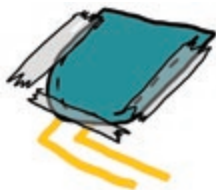
1. Cut out the cat and the pocket, using the red outline on the back side of this page.



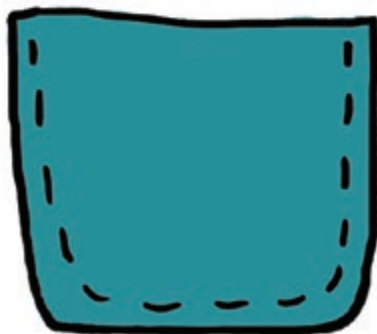
2. Roll up the paper along the dotted lines at the bottom of the cat cutout. This makes the paper thicker so it presses more securely against the switch, making a stronger connection.

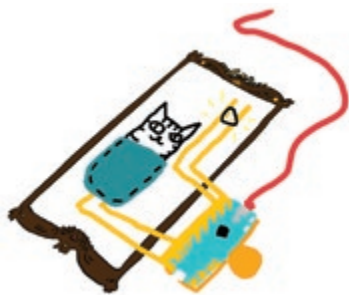
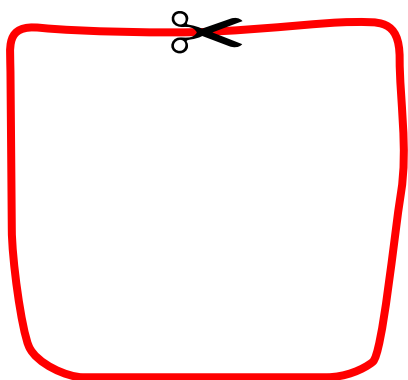
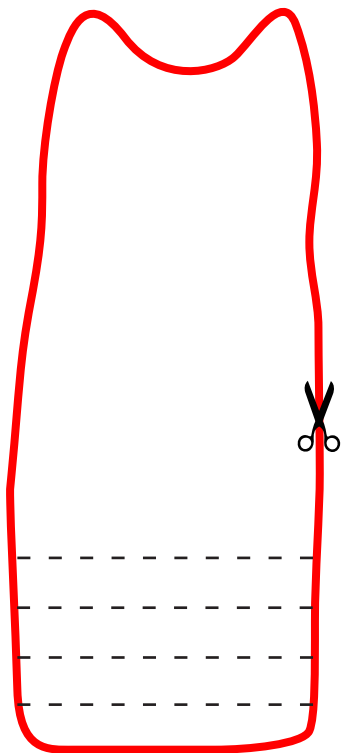


3. Add copper tape to the gray line, then build the circuit on page 3-33.

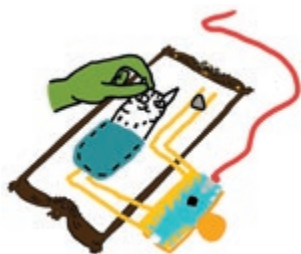


4. Tape the pocket over the footprint marked D, by taping the left, right and bottom sides. *Be sure to leave the top open.* Now the pocket is ready to hold the cat!

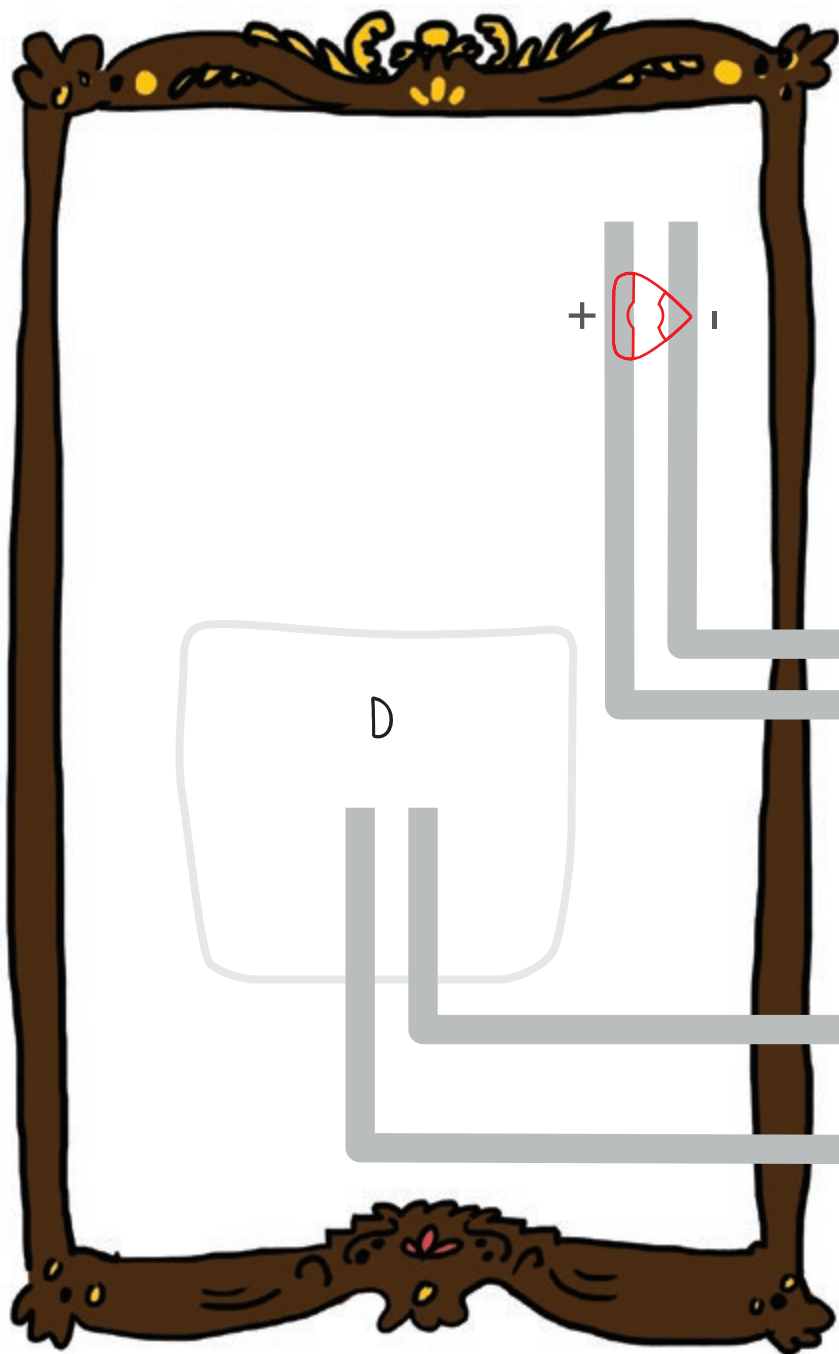




Here's what the finished pocket character switch looks like. When we put the cat in the pocket, it will close the circuit, turning the switch on. Make sure to push the cat all the way into the pocket for a secure connection.



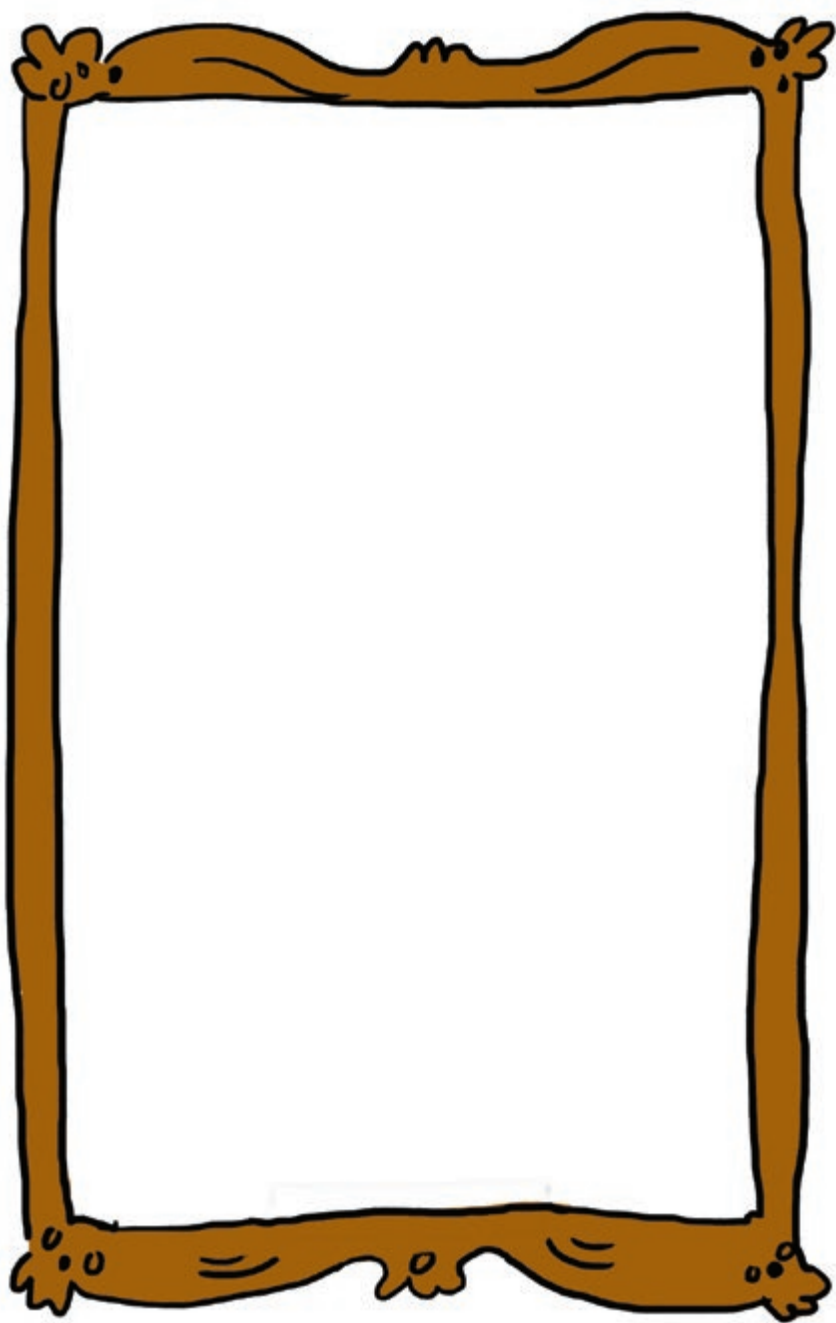
We can turn off the switch by pulling the cat out of the pocket, or by flipping the cat around so that the copper tape doesn't touch the circuit.



## POCKET CHARACTER SWITCH

*Add a Switch!*

**3-33**



YOUR SWITCH!

# DESIGN YOUR OWN SWITCH!

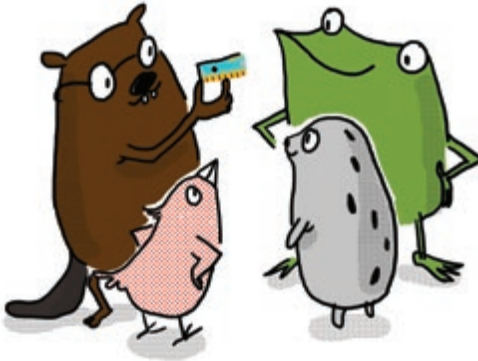
Well done! Now that we've crafted a bunch of switches, try drawing your own switch from scratch. Remember: it's all about making and breaking connections in the circuit!



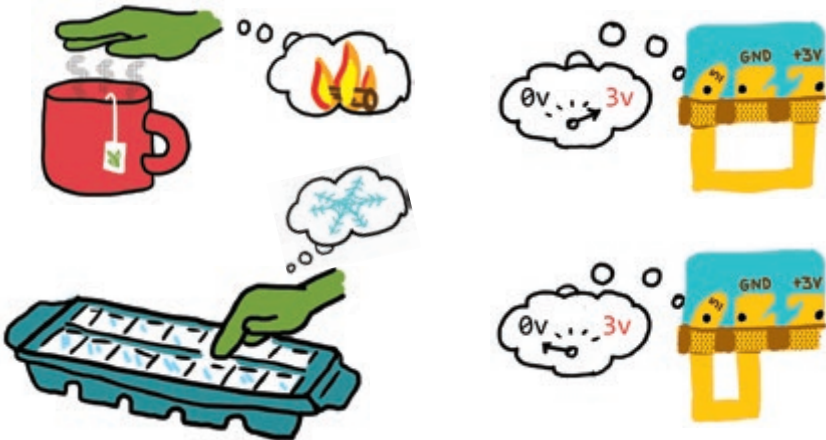


# DOWN THE RABBIT HOLE: INPUT - PROCESS - OUTPUT

So we've played a bunch with switches now, and even designed our own! But how does the Chibi Chip actually know when to turn things on and off with the switch?

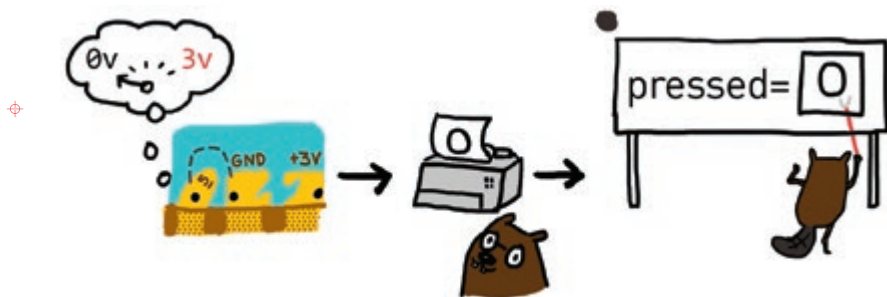


It works by taking in information from the world through the **input pin**. The pin is named "input" because information goes "in" to the board. What information, you ask? Voltage!

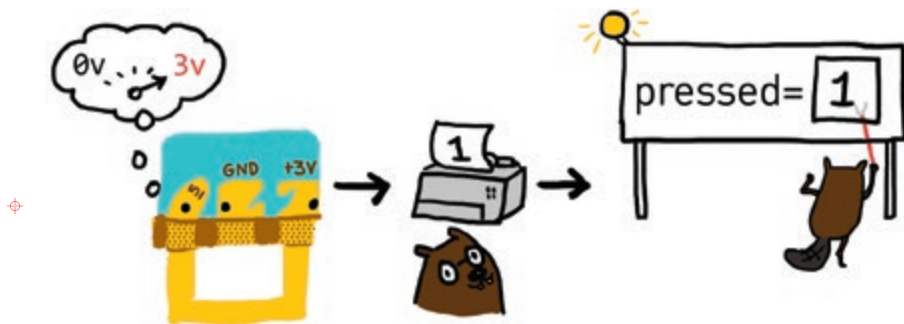




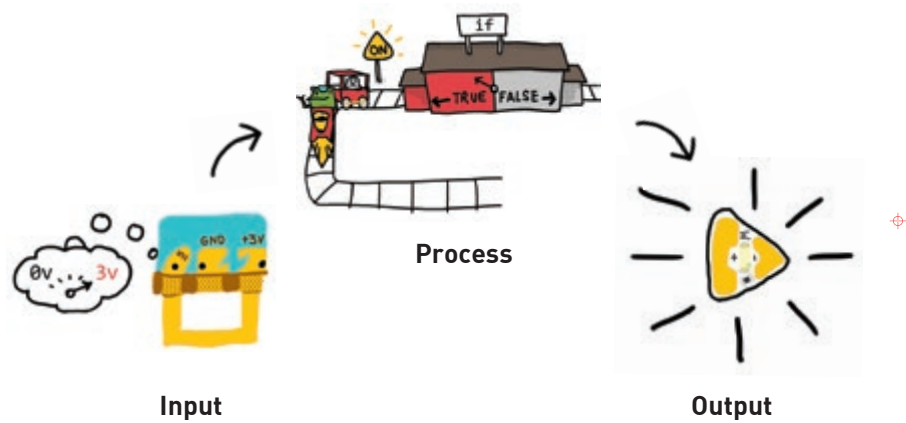
Just as we take in information about hot or cold through our fingers, a Chibi Chip takes in information about voltage through its pins. When nothing is connected to the input pin, a special circuit inside the Chibi Chip gently “rests” the pin at a value of 0 volts, so it reads **0**. The indicator light above the input pin matches this reading and is thus off when nothing is touching the pin.



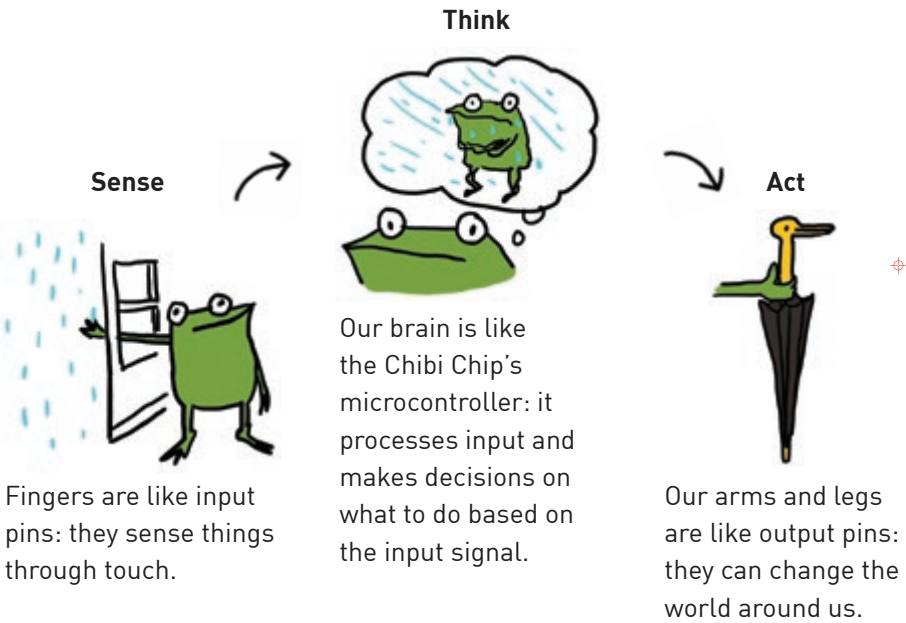
When we connect the input pin to +3V by pressing the switch, the pin can sense that the voltage has gone up. The voltage on the pin becomes 3V, so it reads **1**, and the indicator light turns on.



The code in the microcontroller processes the information from reading the input voltage to make choices and compute the proper output. We save the voltage reading in our `pressed` variable, process it through the `if()` statement, and then turn on or off the LED.



It's a bit like how our hands work with our brain to sense the world around us and to do stuff in response:





"We've learned so much about coding!" said Fern.  
"Here are some of the things we learned!"

### Code Structure:

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

Code inside `{ }` runs once at the beginning, when the Chibi Chip is first turned on.

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

Code inside `{ }` runs over and over again after setup.

### Output Commands:

```
outputMode(pin number);
```

 Turns a pin into an **output**, which can turn things like LEDs on and off.

```
on(pin number);
```

 Turn ON **pin number** by connecting it to +3V.

```
off(pin number);
```

 Turn OFF **pin number** by connecting it to GND.

### Input Commands:

```
inputMode(pin number);
```

 Turns **pin number** into an **input**, which can then read sensors and switches.

```
read(pin number);
```

 Reads the voltage connected to **pin number**, and reports it as a **0** or a **1**: if the voltage is closer to GND, the reading is **0**. If the voltage is closer to +3V, the reading is **1**.

### Control Flow:

```
pause(milliseconds);
```

 Makes the code pause for the amount of milliseconds (there are 1000 milliseconds in a second).

```
if(condition) {  
  // option 1  
} else {  
  // option 2  
}
```

 If the condition is true, run option 1. Otherwise, run option 2.

“What else can we do with these lights?” asked Fern.  
Everyone sat down and thought for a moment.  
“Instead of going directly from off to on, could we make them  
transition gradually?” Carmen asked.

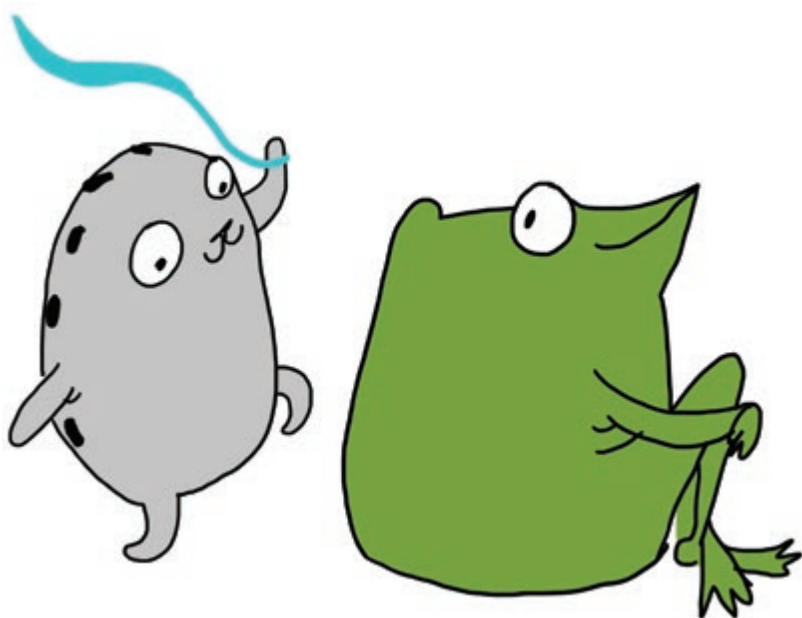


“You mean like fading them in and out?” said Edith.  
“Yeah!” exclaimed Fern. “That would make a really pretty effect  
that can go with all kinds of scenes.”  
“I feel like we could figure out how to do that!” Sami said.  
“Shall we try?”



# Chapter 4:

## Fade in and Out!



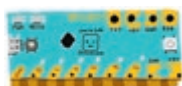
"We've figured out how to turn the lights on and off, but how can we make them fade slowly?" asked Fern the frog.

"I think I can explain!" said Sami the seal as she danced around the room. "And while we're figuring it out, we can make dance costumes. We can have a parade later, and if we decorate our costumes with twinkling lights we will look even more fabulous!"

# SET BRIGHTNESS LEVELS

Rather than turning LEDs fully on or fully off, we can also dim them to intermediate brightness levels! This lets us make fun new lighting effects, such as fading our lights in and out smoothly, or creating a twinkling effect!

## You will need:



Chibi Chip,  
mounted in a Clip



Internet connection



Programming  
and power cable



USB Power



A device with a web  
browser (phone,  
computer, or tablet):  
this is the programming  
device

Upload the **Set Level** example code to the Chibi Chip. Go to **Examples → Love to Code Vol 1 → Set Level**. When the code is done uploading, the light on pin 0 will turn on and off by stepping through different brightness levels!



UPLOAD NOT WORKING? TRY THESE DEBUGGING TIPS. IF THESE DON'T HELP, CHECK OUT THE DEBUGGING SECTION IN THE BACK OF THE BOOK!



MAKE SURE THE  
VOLUME IS ALL THE  
WAY UP.



DID THE PROG LIGHT TURN  
STEADY RED BEFORE  
PROGRAMMING? IF NOT,  
PRESS AND HOLD THE PROG  
BUTTON UNTIL IT TURNS RED  
AND TRY UPLOADING AGAIN.



DID THE UPLOAD SOUND  
WAVEFORM ANIMATION  
APPEAR? IF NOT, TRY  
REFRESHING THE PAGE AND  
CLICKING UPLOAD AGAIN.

# DECODE THE CODE

Let's take a look at the **Set Level** example code to see what's going on with our LED!

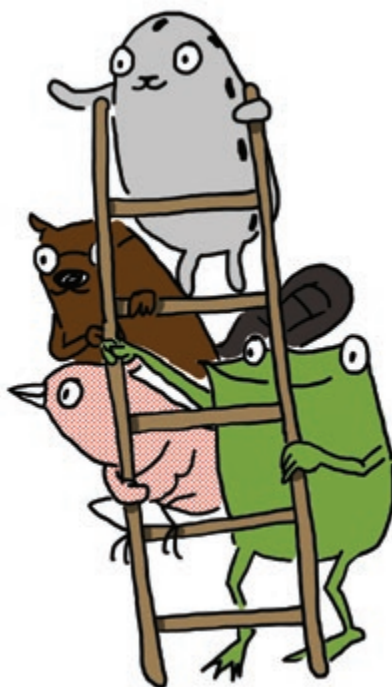
```
// Love to Code  
// Volume 1: Set Level
```

```
int LED = 0;           // initialize LED as pin 0
```

```
void setup() {  
  pinMode(LED); // set LED pin as output  
}
```

```
// Use repeated setLevel(pin, percent) statements  
// to increase brightness level to 100 (fully on)  
// and then back down to 0 (fully off)
```

```
void loop(){  
  setLevel(LED, 25);  
  pause(500);  
  setLevel(LED, 50);  
  pause(500);  
  setLevel(LED, 75);  
  pause(500);  
  setLevel(LED, 100);  
  pause(500);  
  setLevel(LED, 75);  
  pause(500);  
  setLevel(LED, 50);  
  pause(500);  
  setLevel(LED, 25);  
  pause(500);  
  setLevel(LED, 0);  
  pause(500);  
}
```



`setLevel(pin, level)` sets the brightness level of a pin. It's like using a dimmer instead of an on/off switch to turn on a lamp. Instead of all the way on or all the way off, it lets us set in-between brightness levels. Here's how it works:

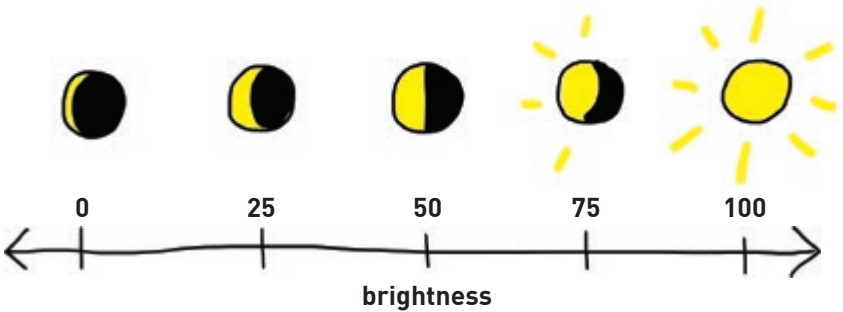
```
setLevel(0, 25);
```

This tells the Chibi Chip which pin we are setting.

This is the brightness level we want. It can be any whole number, from 0 for all the way off, to 100 for all the way on.



In our example code, we first used `setLevel(LED, 25)` to set the LED pin, or pin 0, to 25% brightness. We then set pin 0 to varying brightness levels, anywhere from 0 for all the way off to 100 for full brightness, in increments of 25.





# PLAY WITH CODE

Play with our example code by setting pin 0 to different brightness levels to create your own patterns.



After programming a new pattern, clip the Chibi Chip to a previous circuit, like the one on page 2-19, and see how it changes the scene!



That's the great thing about programming. Even with the same circuit and the same scene, we can tell different stories just by changing the code!

# DECODE THE CODE

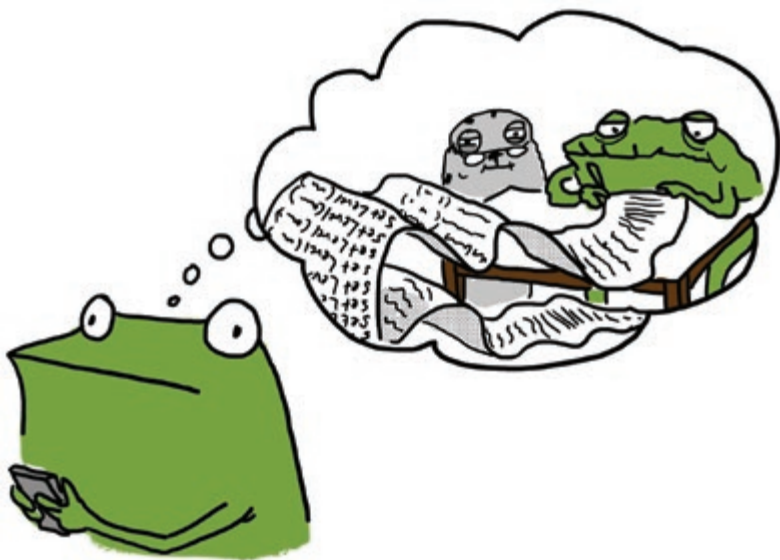
Rather than suddenly switching between brightness levels, how do we get the lights to transition even more smoothly? We want our lights to glide up and down like a ramp, instead of stepping up and down like a staircase.

Right now, the brightness increments are big, at 25% per step. Perhaps things would smooth out if we could make the increments smaller?



The smallest brightness increment is 1. So to fade more smoothly from off (0) to full brightness (100), we could write `setLevel(LED, 0)`, then `setLevel(LED, 1)`, `setLevel(LED, 2)`, `setLevel(LED, 3)`, and so on all the way to `setLevel(LED, 100)`.

But that would take so many lines of code, and take forever to write!



Luckily, there is a neat code structure called a loop to save us! To try out a looping style known as the **while()** loop, go to the code editor and load the example found at **Examples → Love to Code Vol 1 → Fade with While Loop**.



Once we've loaded this code into the Chibi Chip, the LED over pin 0 should be gently fading in and out. Here's our new code, starting with the **loop()**:

```
void loop() {  
  int brightness = 0;
```

Create a variable called **brightness** to store our current brightness level.

```
  while(brightness < 100) {  
    setLevel(LED, brightness);  
    pause(10);  
    brightness = brightness + 1;  
  }
```

#### Fade in the LED

Set the LED to the current brightness, increase the brightness level by 1 and then repeat this loop until the brightness is 100, or fully on.

```
  while(brightness > 0) {  
    setLevel(LED, brightness);  
    pause(10);  
    brightness = brightness - 1;  
  }  
}
```

#### Fade out the LED

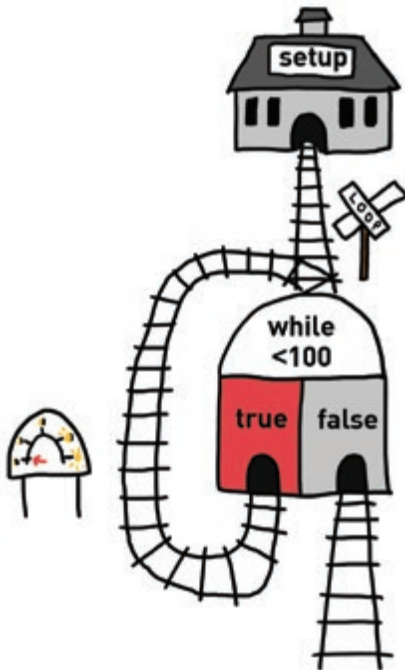
Set the LED to the current brightness, decrease the brightness level by 1 and then repeat this loop until the brightness is 0, or fully off.

Whew! It would've taken us about 400 lines of code to write this program with only **setLevel()** and **pause()** statements, but we were able to write it in only 11 lines with the help of the **while()** loop!

The `while()` loop lets us repeatedly run a snippet of code without having to write it over and over. Any code inside the `while()` loop runs only if the `condition` is `TRUE`:

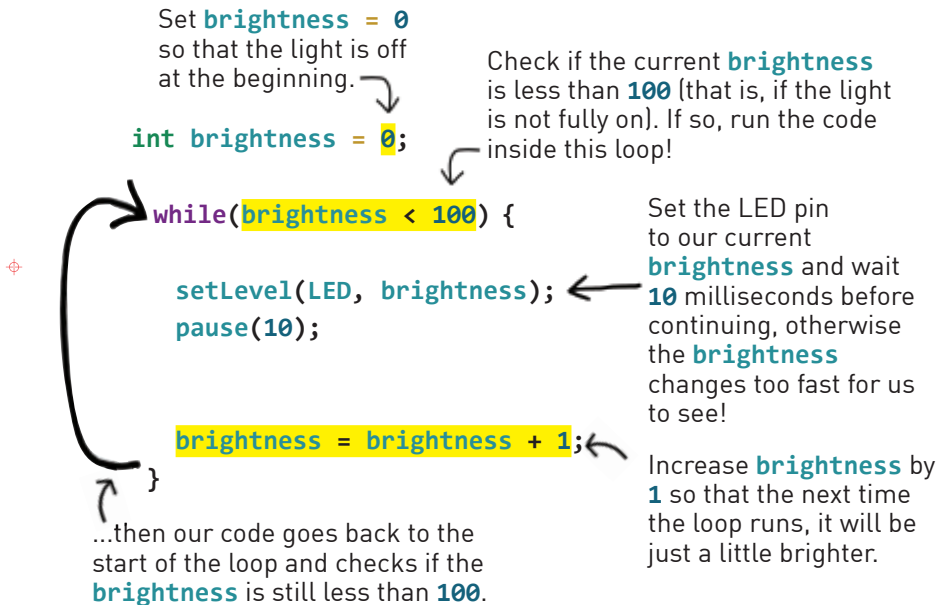
```
while(condition) {  
  // Code here runs over and over  
  // while condition is TRUE  
  Update current condition;  
}
```

First the program checks if the `condition` is true. If so, it will run the code inside the `while()` loop. When it's done running the inner code, it goes back to the top and checks again to see if the `condition` is still true. If so, it goes back and runs the `while()` loop code again. This loop repeats until the `condition` is no longer found to be `TRUE`.



To make sure we don't get stuck forever inside the `while()` loop, we must update our current condition somewhere in the `while()` loop's inner code so that at some point in time, the condition statement will no longer hold true. If we forget to do this, the Chibi Chip gets stuck inside the `while()` loop, which is called an **infinite loop**.

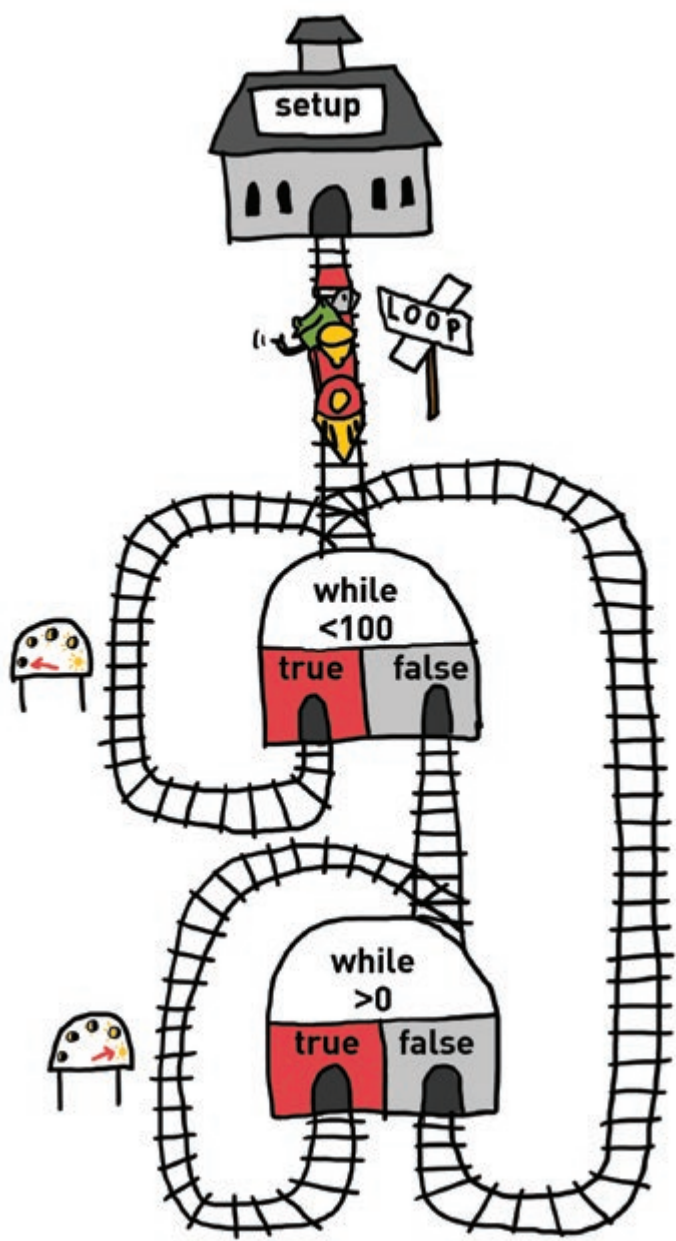
So to gradually fade our light from off to on, we set the **brightness** using a **while()** loop like this:



This keeps going until our **brightness** reaches **100**. At this point, the light is fully on and the code will stop running our **while()** loop and move on to the next line of code following the loop. Fading to darkness works the same way, except now the brightness level starts at **100** and decreases by **1** each time the loop repeats until the brightness level becomes **0** again!



Putting it all together, here's what a map of our example code looks like for fading in and fading out using two back-to-back `while()` loops!



# PLAY WITH CODE

Try playing around with the `while()` loop code to create your own fading light effects!

Can you make the lights fade faster? Now how about slower? Just like with the **Blink** example code, we can change the `pause()` time to make the fade effect go faster or slower.



We can also try changing the brightness increment. For example, changing the increment from `brightness = brightness + 1` to `brightness = brightness + 5` will make the LED fade faster. However, if the increment becomes too big, the fade goes from a smooth ramp back to choppy steps!

The light doesn't have to fade all the way on or off! We can fade to a medium brightnesses by changing the condition statement in the `while()` loop. For example `while(brightness < 50)` will fade the light until it is only halfway on!



Try it out! Can you make a candle-like flickering effect?



# DECODE THE CODE

Now that we have one pin doing fun fade effects, how do we get multiple pins to fade in different patterns? The easiest way is through **multithreading**! Multithreading means running multiple pieces of code at the same time. Each running bit of code is called a **thread**.



Try loading **Examples → Love to Code Vol 1 → Basic Multithreading** onto a Chibi Chip to see multithreading in action! We'll see each of the six indicator lights above the pins flashing different effects. Here's the code:

```
#include "ChibiOS.h"
```

```
#include "SimpleThreads.h"
```

we have to include these two special lines of code at the beginning to use multithreading

```
//// thread 0
void setup0() {
  // thread 0's setup code here
  outputMode(0);
}
void loop0() {
  // thread 0's loop code here
  on(0);
  pause(300);
  off(0);
  pause(300);
}
```

Thread  
0:  
Blink pin 0





The code is longer, because we've crammed six programs into one, so it keeps going:

Thread 1:  
Blink pin 1

```
//// thread 1
void setup1() {
  outputMode(1);
}
void loop1() {
  on(1);
  pause(500);
  off(1);
  pause(500);
}
```

Thread 2:  
Blink pin 2

```
//// thread 2
void setup2() {
  outputMode(2);
}
void loop2() {
  on(2);
  pause(800);
  off(2);
  pause(800);
}
```

Thread 3:  
Blink pin 3

```
//// thread 3
void setup3() {
  outputMode(3);
}
void loop3() {
  on(3);
  pause(1000);
  off(3);
  pause(1000);
}
```

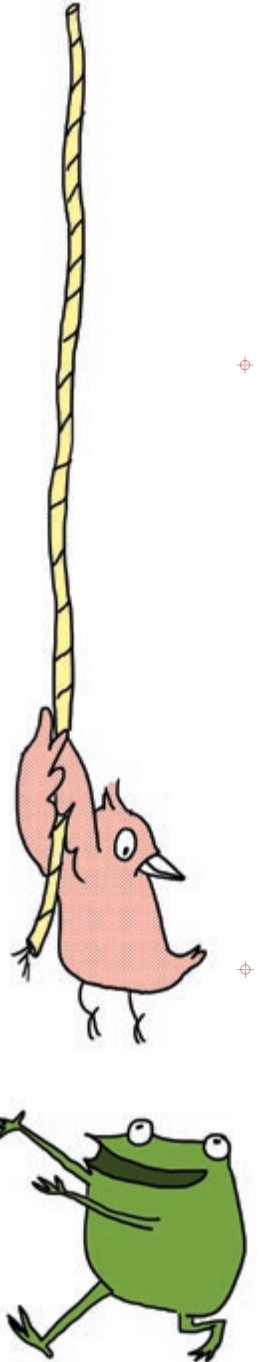


Thread  
4:  
Fade pin 4

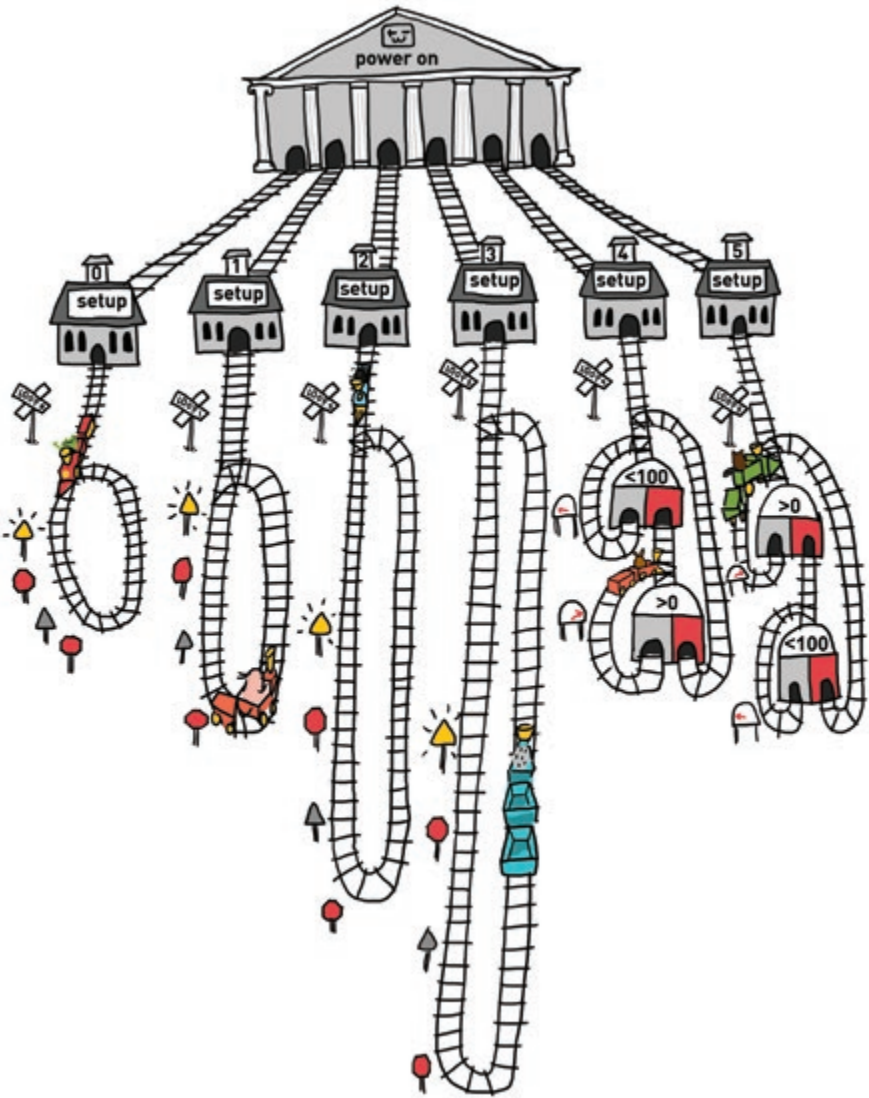
```
//// thread 4
int led4 = 4;
void setup4() {
    outputMode(4);
}
void loop4() {
    int brightness4 = 0;
    while(brightness4 < 100) {
        setLevel(led4, brightness4);
        pause(10);
        brightness4 = brightness4 + 1;
    }
    while(brightness4 > 0) {
        setLevel(led4, brightness4);
        pause(10);
        brightness4 = brightness4 - 1;
    }
}
```

Thread  
5:  
Fade pin 5

```
//// thread 5
int led5 = 5;
void setup5() {
    outputMode(5);
}
void loop5() {
    int brightness5 = 100;
    while(brightness5 > 0) {
        setLevel(led5, brightness5);
        pause(10);
        brightness5 = brightness5 - 1;
    }
    while(brightness5 < 100) {
        setLevel(led5, brightness5);
        pause(10);
        brightness5 = brightness5 + 1;
    }
}
```



Whew, that's all six threads! Each thread is made up of an individually numbered `setup()` as well as an individually numbered `loop()` function. It's like having six trains running at the same time on six different tracks!



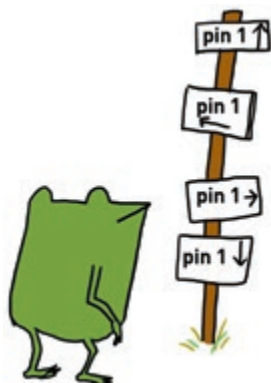
Just as real trains could collide if we put them on the same track at the same time, a Chibi Chip could get confused if two threads fight over a common resource. For example, if `loop0()` and `loop1()` both try to control pin 0, the result would be a confusing mix of commands from `loop0()` and `loop1()`. This kind of collision is called a **race condition**. In order to keep things **thread-safe**, which means preventing collisions, we just have to make sure that different threads don't ever try to control the same pin.



Likewise, we should also make sure the names of variables are different between threads, so the Chibi Chip doesn't get confused. It's like having two people with the same name share a mailbox — they'd have no way of knowing which message was meant for which recipient!

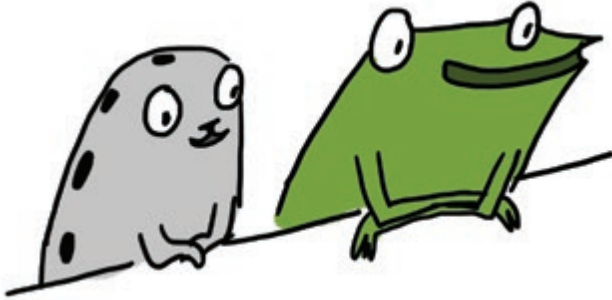
That's why we've named all the variables with their respective thread number. We could name variables whatever we want, but naming them something different for each thread keeps things simple to understand and thread-safe.

Multithreading is easy, as long as we're careful to keep each thread's resources separate!



# PLAY WITH CODE

Ready to play with some multithreading? Let's go! Try changing the blink and fade patterns inside each thread.



Remember, in order for our multithreading code to work, we need to include these two special lines at the beginning of your code:

```
#include "ChibiOS.h"  
#include "SimpleThreads.h"
```

We also need to make sure that all six **setup0()** through **setup5()** and **loop0()** through **loop5()** functions are in the code, or else it will not run. If you don't want to use a particular thread, just make the setup and loop functions of that thread empty, like this:



```
void setup0() {  
}  
void loop0() {  
}
```

Finally, if you want to control more than one pin in one thread, that's okay. Just make sure you're not trying to control the same pin in another thread too, or else there might be a collision!



# MAKE A SCENE

You've learned so much! Now use your coding chops to help Fern and friends design some light-up outfits for the parade.



You can make a scene bigger by using conductive fabric patches to connect between two pages. Try it now!

First, remove pages **4-19** through **4-22** from the binder. Craft your circuits on this page, **4-18**, and on page **4-23**, using conductive fabric patches to bridge between the pages.

Once finished, lay page **4-20** over page **4-18**, and page **4-21** over page **4-23**. The lights will shine through the parade!











# MAKE A SCENE - CONTINUED

Continue your circuit here! Use conductive fabric patches to extend the circuit from page **4-18** on to this page, and then lay the middle pages on top to complete the scene.





# DOWN THE RABBIT HOLE: PULSE WIDTH MODULATION

So how does a Chibi Chip fade lights in and out? It may seem like the Chip is changing the voltage going to the LED to make it brighter or dimmer, as previously explained in Chapter 1, but it's actually turning the light on and off very quickly in a process called **pulse width modulation (PWM)**.

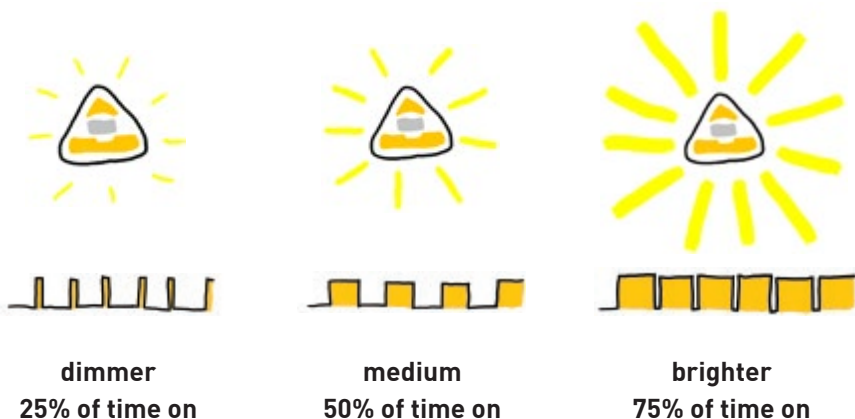


The Chibi Chip can only turn a pin fully on or fully off, so it cannot dim a LED by creating different voltages. Instead, to make the light look like it's partially on, the Chibi Chip blinks the light very quickly, around 400 times a second: so fast that we can't even notice it turning on and off!

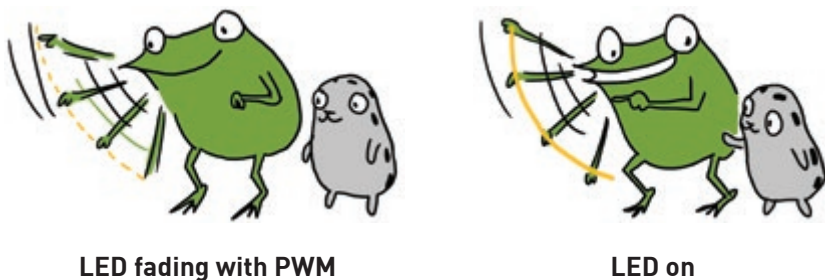


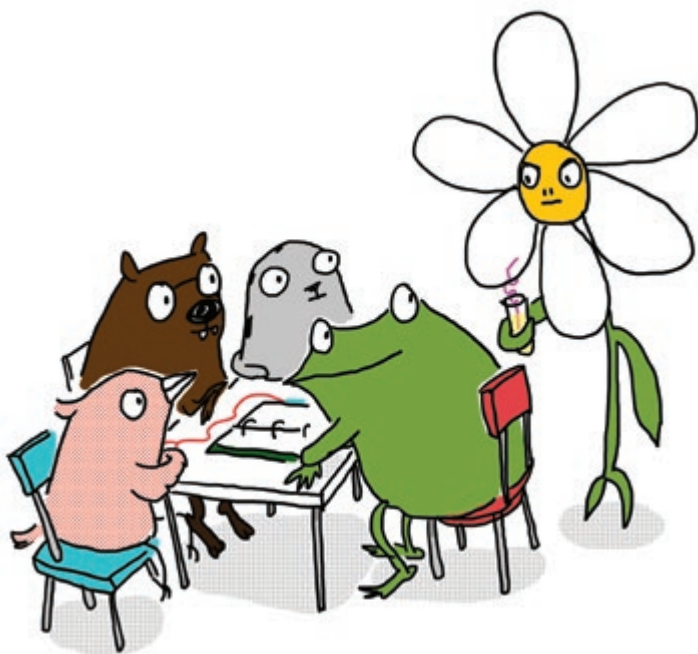
Instead, our eyes blur the rapid blinking so it seems like the LED is shining at a constant, but dimmer, light level.

Each blink is called a **pulse**. How dark the light fades depends on how much of the time the light spends in the on state versus the off state. This ratio is called the **duty cycle**, and it is expressed in terms of a percentage. So, when we call `setLevel()`, the brightness argument is actually specifying the duty cycle of the PWM.



There's a trick you can try to see the blinking for yourself. First, program a LED to fade and then wave it very quickly in front of your eyes. Instead of seeing a continuous streak of light, you will see a "dotted line" of light. This is the light turning on and off very rapidly! Next, fix the light to on, by programming it using `on()` and try waving it around again. This time, you'll see a continuous, solid streak!





"This is really cool!" exclaimed Fern.

"It's not **that** cool," muttered a smug, sarcastic voice. It was George the flower. "There are way cooler switches you can make using a light sensor. Haven't you heard of one before?"

Carmen and Edith both rolled their eyes. "This guy," whispered Sami.

Fern thought for a minute. Maybe George could teach her something new she didn't already know. "Hey George," she said, "can you show us?"

George heaved a sigh. "I guess. If you can keep up."

"I'm pretty sure we can," said Fern with a smile.

You're amazing! You've finished Love to Code Volume 1. For new adventures with Fern and friends, like how to use a light sensor, check out [chibitronics.com/lovetocode](http://chibitronics.com/lovetocode).

# Debugging



Stuff isn't working? No matter! That's what debugging is for! **Debugging** means looking closely at our project, finding the problems — also known as **bugs** — and then fixing them so that our project works as expected.



WELCOME TO MY WORLD!  
DON'T WORRY WHEN THINGS  
DON'T WORK THE FIRST TIME.  
FIGURING OUT WHAT WENT  
WRONG IS HOW DISCOVERIES  
ARE MADE!

# BREAKING IT DOWN

When a project doesn't work, it can seem overwhelming. The problem could be anywhere!



That's why we break it down into smaller parts and look at them one by one. Then it's not so scary anymore! It's just like how we take small bites when we eat, rather than swallowing the whole meal all at once!





Here are some of the smaller pieces we can break our Love to Code project into:

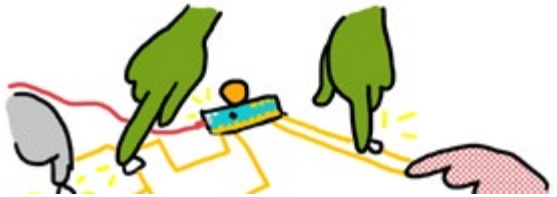
**Power:**

Is the Chibi Chip getting enough electricity to turn on and run our circuit?



**Circuit:**

Is everything in our circuit connected properly?



**Upload:**

Are we able to send code from our programming device to the Chibi Chip?



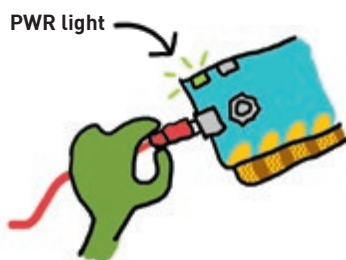
**Code:**

Are there errors in our code? Does our code actually say what we mean?

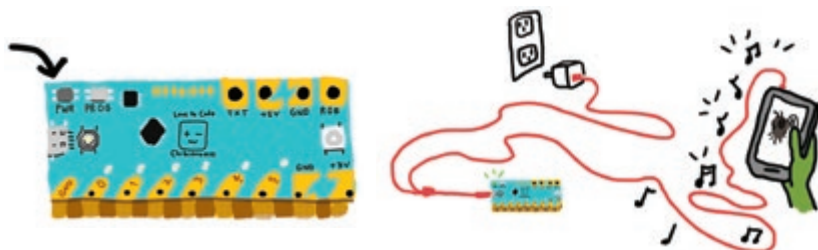


# POWER

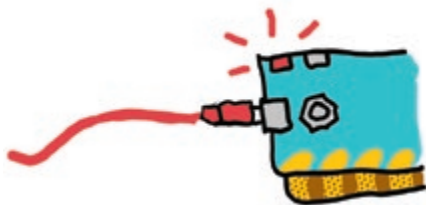
Let's start by checking the power! If everything is working properly, the power (PWR) light on the Chibi Chip will be green to show that it has enough power.



**1. Is the PWR light off?** If so, it means the Chibi Chip is off because it isn't getting enough power! Try plugging the Chibi Chip into a power supply that you've recently used to charge a phone, like a USB wall plug or computer's USB port. Phones take a lot of power to charge, so if the USB port can charge a phone, it can power a Chibi Chip!



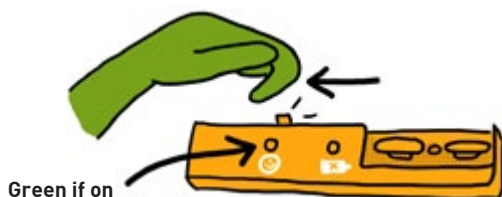
**2. Is the PWR light red instead of green?** That typically means we have a short circuit connecting +3V and GND directly to each other, draining power from the Chibi Chip! If you're powering the Chibi Chip from a computer, an error message may also pop up on your screen about too much power or current being drawn.



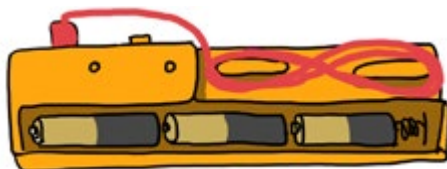
If this happens, unplug the Chibi Chip from the power source, find the connection causing the short circuit, and remove it. To learn more about short circuits, go to page **1-8**.

If you're using a Chibi Book for power, it's worth checking to see if the battery pack is working properly. Is the battery pack's green power light on? If not, here are a few things to check:

**3a. Is the battery pack switched on?** Make sure the switch at the top of the battery pack is flipped to the left, and that the green light is on.



**3b. Are the batteries inserted properly?** There should be three AA batteries placed in the battery pack like this with the + side (nub side) facing left:



**3c. Is the red "low battery" light on or flashing? Is the green light flashing on and off?** This usually means the batteries are running really low. The Chibi Book is trying to find the power to run your circuit, but exhausting itself and shutting down, only to try again after a short rest. Try placing fresh batteries into the Chibi Book's battery pack. The Chibi Book works best with alkaline or NiMH rechargeable batteries!



ONCE WE KNOW POWER IS GETTING TO THE CHIBI CHIP,  
WE CAN CHECK THE REST OF OUR CIRCUIT FOR OTHER  
PROBLEMS!



# CIRCUIT

Is the Chibi Chip powered on, but the LEDs are still not turning on as expected? There might be a bug in the connections of our circuit! Maybe the circuit is incomplete because we forgot to make a connection, or the connection is faulty. Or maybe something is connected that should not be, causing a short circuit.



LET'S TAKE A LOOK AT SOME COMMON  
CIRCUIT BUGS!

**1. Are your Chibi Light LEDs installed in the correct direction?** Make sure that the pointy (-) end of every LED is connected to GND, and the wide (+) side of every LED is connected to a numbered pin or +3V. Otherwise, the LED is installed backwards, and it won't turn on.



YAY!



NOPE.

**2. Are any LEDs causing a short circuit?** Make sure that the shiny metal pads of your LEDs are touching only one strip of copper tape. If one pad of an LED is touching two different copper wires, then there is a short circuit and your light will not turn on. In the example below on the right, the wide (+) side is accidentally touching both GND and +3V, causing a short circuit!



YAY!



NOPE.

**3. Is there enough overlap between the pads of your LED and the copper tape?** There needs to be plenty of overlap between the metal pad of your LED and the copper tape for there to be a strong electrical connection. If the overlapping area is too small, then power can not flow to the LED.



YAY!



NOPE.

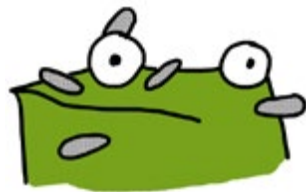
**4. Are all the LEDs stuck down firmly?** The conductive adhesive used on circuit stickers require a good press to make a solid connection. So, try pressing down on the metal pads of your LEDs to make the connection stronger. If an LED turns on when pressed, and off after letting go, that means there's a weak connection between the copper tape and the LED's metal pads.



If you have more than one LED connected in parallel, and only some of them are on, the most likely issue is a weak connection on the LEDs that aren't turning on. You can fix these weak connections by taping conductive fabric patches between the LED's metal pads and the underlying copper tape.



YAY!



NOPE.

**5. If you've tried everything and an LED is still not turning on, try switching it out for a new LED.** LEDs can break if they are creased, or if they are exposed to water, dirt, or grease.

# CIRCUIT (CONT'D)

**6. Is the Chibi Chip aligned properly with the circuit?** Make sure the shiny metal pads on your Chibi Chip line up with and touch the copper tape of your circuit.



**7. Is the copper tape really bumpy or wrinkly?** If so, sometimes bumps and wrinkles can prevent solid connections to your LEDs or Chibi Chip. If this is the case, try smoothing out the tape by rolling over it with the flat side of a pen or pencil.



NOPE.

**8. Is the copper tape or LED sticker not sticky anymore?** Make sure your hands are clean and dry before working with the copper tape and stickers. If the stickers or tape get dirty, they may lose their tack, causing weak connections in the circuit. If this happens, try patching weak connections with conductive fabric patches.



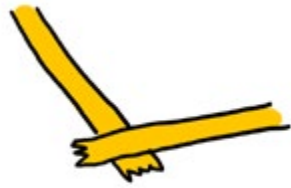
**9. Is there a tear in the copper tape?** If so, the tear breaks the circuit's connection. Fix tears and restore the connection by taping over both sides of a tear with a single conductive fabric patch.



✦ **10. Is your circuit connected at turns and corners?** Copper tape often tears at turns, and it is not enough to stick two pieces of copper tape on top of each other. When making a corner using two pieces of copper tape, reinforce the connection using a conductive fabric patch after sticking down the second piece of copper tape.



YAY!



NOPE.

**Make reliable turns with a single piece of copper tape and save on conductive fabric patches using this origami trick:**



1. Fold tape back, so the sticky side faces up.



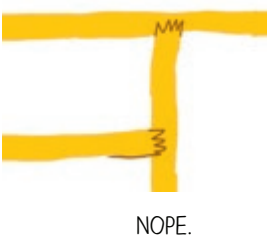
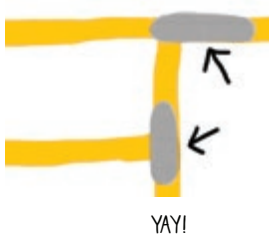
2. Flip and turn the tape at the same time.



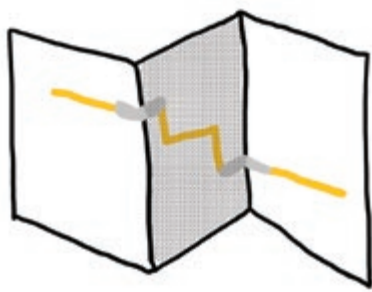
3. Flatten the corner, and you're done!

# CIRCUIT (CONT'D)

**11. Are there branches in your circuit, or do you need to extend your copper tape with another piece?** Make sure to use a conductive fabric patch to connect multiple pieces of copper tape. Just sticking two pieces of copper tape on top of each other will not create a strong or reliable electrical connection. Even if the circuit seems to work at first, over time the connection will break down.



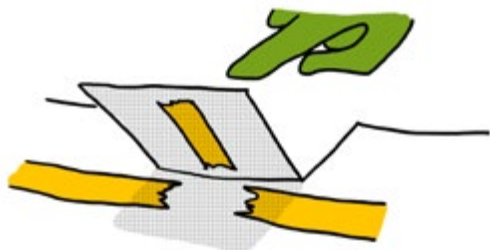
**12. Do you have a moving hinge in your circuit?** Make sure to reinforce it with a conductive fabric patch. Tears are especially common at places where the copper tape gets folded repeatedly. Copper tape will crack when folded too many times.



FABRIC PATCHES WON'T DEVELOP CRACKS EVEN WITH REPEATED FOLDING, SO THEY'RE GREAT FOR REINFORCING MOVING OR BENDING PARTS OF A CIRCUIT!



**13. Is a switch not responding when pressed?** If so, make sure that the copper tape on the switch actually lines up with the gap in the circuit. If it's not aligned, the copper tape won't close the circuit when the switch is pressed, and the switch will not work.



NOPE.

**Make the copper tape patch large and the circuit gap small.** This makes it easier for your switch to close the circuit. This is especially important for more advanced switches, such as the wind sensing switch (page 3-28), where the switch isn't actually pressed by hand.



YAY!

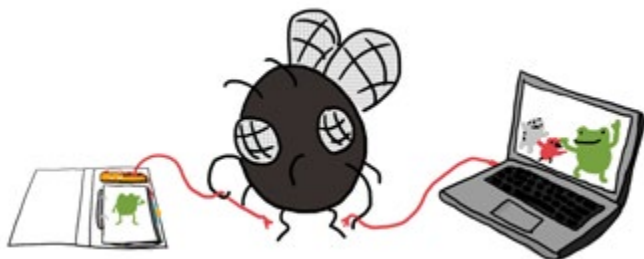
**14. Is the switch sometimes working and sometimes not when pressed?**

That means the switch connection is unreliable. Check that the copper tape is smooth and clean on both halves of the switch. Then, try ways to increase the pressure between the pieces of copper tape in your switch.

For example, in the pocket character switch (page 3-31), if you use a thicker paper, this will make the switch stiffer and more springy. This applies more pressure to the connection, helping to make your switch more reliable.

# UPLOAD

Even if the circuit is done, the Chibi Chip needs to be programmed correctly for our project to work. Sometimes there are problems when we try to send code from our programming device to the Chibi Chip. Here's how to check if this is an issue!



We test the upload process by trying to upload the Blink example program. Save any code you've written and open up the blink example code by selecting **Examples → Love to Code Vol 1 → Basic Blink**. We start with this known code because it's easy to tell if it's working properly. If the upload is successful, we will see pin 0 blink!



If the Basic Blink didn't upload, let's check out some possible reasons why:

**1. Is the volume turned up all the way up? Is the sound accidentally muted?** Make sure to unmute your sound and turn the volume all the way up so that the Chibi Chip can hear the code. One of the most common upload problems is that the audio is simply too quiet!



**2. Is the code being converted into sound?** Click the “Upload” button. Within a few seconds, this orange “sound banner” should appear at the bottom of the screen:



If the sound banner does not appear, the code is not being translated into sound. If this happens, try refreshing the browser. Then, re-open the blink example code, and click “Upload” again.



**3. Can your browser play web audio?** Try unplugging the audio cable from your programming device for a moment, turning the volume up to its maximum level, and clicking upload. When the sound banner appears, you should hear a staticky sound. This staticky sound is the code that the Chibi Chip is listening for.



**If you can't hear the code playing, try testing to see if your browser is compatible with the Love to Code system.** To run the test, visit [ltc.chibitronics.com/test](http://ltc.chibitronics.com/test) and click on the “Test Audio” button. If you hear a short tune play, that means your browser is compatible. If not, try switching to an up-to-date version of Chrome, Firefox or Edge.



Welcome to the Chibitronics browser compatibility test!  
Tap the “Test Audio” button above.

# UPLOAD (CONT'D)

**4. Is the audio being distorted?** Some laptops automatically apply audio “enhancements” (such as Dolby Audio or bass boost). These enhancements will distort sound in a way that the Chibi Chip may not be able to understand. If you have a Windows computer, particularly those made by Lenovo, try following these instructions to disable pre-loaded audio distortions:

2. Select “Turn off Dolby Audio” (you can turn it on again using the same menu item)

1. Right-click “Dolby” icon

**5. Do you hear a static sound while programming the Chibi Chip?** That means your audio cable isn’t plugged all the way in. Make sure to push the audio cable all the way into your programming device, so that the Chibi Chip is hearing the code, and not you!



**6. Is the Chibi Chip in program mode?** Before clicking upload, make sure to press and hold the programming (PROG) button on the Chibi Chip until the PROG LED blinks and stays red. Otherwise the Chibi Chip won’t know to listen for new code.

1. Press hard & hold!

2. PROG light solid red! Let go of the button now.

**7. Is the Chibi Chip hearing the audio code?** During upload, the red PROG light on the Chibi Chip should blink to show that it hears the code. If the PROG light stays solid red, even though the sound should be playing, it means the audio is not making it to your Chibi Chip. If this is the case, try turning up the volume on your device and check that sounds are playing (**Step 3**).



**8. Does the PROG light turn green?** If so, that means the upload is successful. If your project still isn't behaving as expected, try checking the circuit or the code!



**If the PROG light just blinks red, but never turns green** that means your Chibi Chip heard some of the sounds, but was not able to hear the entire program. Check again that your volume is turned all the way up (**Step 1**), and check for hidden add-ons or plug-ins that could be distorting the audio (**Step 4**). Furthermore, if another kind of device works, this means there is likely some kind of distortion in your programming device's audio path.



**8. Still stumped?** Send us a note at [help@chibitronics.com](mailto:help@chibitronics.com) and we'll try our best to debug with you!



# CODE

Sometimes there will be errors in our code that makes our circuits do something other than what we intended. In this case, we have to debug our code!



DEBUGGING CODE CAN OFTEN BE A LONG AND FRUSTRATING PROCESS, BUT REST ASSURED, IT'S REALLY SATISFYING WHEN YOU FINALLY FIGURE OUT WHAT'S WRONG AND GET YOUR PROJECTS WORKING!

## Clicked “Upload” on the browser, but the sound bar doesn’t appear?

There may be formatting errors in your code. If you’re able to upload the blink example code but not your own code, this is likely the case.



The code editor needs your code to be written in exactly the right format, otherwise it won't understand the code and cannot **compile** it. Compiling means translating the text code in your browser into the code song that a Chibi Chip can understand.



As a result, little errors in how the code is written, called **syntax errors**, will stop an entire program from uploading!

Here are some common errors to check for:

1. Missing a semicolon `;` at the end of each line of code
2. Forgetting the closing parenthesis `)` in a function call
3. Forgetting the closing curly brace `}` of a loop or if statement
4. Misspelling a variable or function name
5. Mismatching the capitalization of variable or function names
6. Creating multiple variables with the same name



Luckily, for common syntax errors like these, the editor will add a small red circle next to, or just after, the code lines with problems. If you hover over these circles with your mouse, the editor will pop up additional information about the error. This way it's easier to find the mistakes and fix them!

```
1 //Love to Code
2
3 int LED = 0;
4 int LED = 1;
5
6 void setup() {
7   outputmode(LED);
8 }
9
10 void loop() {
11   on(LED);
12   pause(1000;
13   off(LED)
14   pause(1000);
15
```

Can you spot all the errors in the code above? How would you fix them?



# CODE (CONT'D)

**Code uploaded properly, but not behaving as intended?** That means that the code is formatted correctly, so it compiled and uploaded, but there is an error in what the code tells the circuit to do, causing the program to behave in an unintended way. This type of error is called a **logic error**.



Logic errors can be challenging to spot and fix because we have to figure out our error based on the unexpected behavior of our circuit.



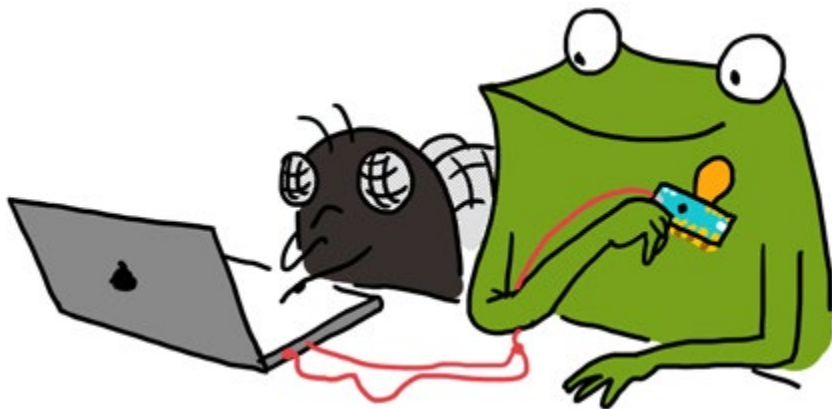


Some logic errors can be quick to spot. Here are a few examples of common logic errors:

**1. Are there enough delays?** If there are not enough delays in your program, or if the delays are not long enough, the logic might be running as intended, but the effects are happening too quickly for us to see.



**2. Do the pins in the code match the pins in the circuit?** It's easy to forget which pin was used when crafting a circuit. For example, when adding a switch, make sure to update the pin number in the example code to match the pin number you've wired the switch to!



**3. Are the LEDs blinking on and off, but not very brightly?** It may be because the LED pin is still in input mode, which is the default mode. If this is the case, the LED will still blink, but the pin will not be able to turn an LED fully on. Make sure to set every pin meant to turn on an LED to be an output using `outputMode(pin number);` in the setup code.

# CODE (CONT'D)

Most logic errors are hard to spot. But don't worry: finding bugs and fixing them is all part of learning to code! Here are some tips for finding the trickier bugs:



## 1. Pretend to be the Chibi Chip, and trace through the code line by line.

Tracing through a program slowly can help catch many bugs. For example: “turn light on”, “wait 1 second”, “turn light off”, “loop ends, repeat”, “turn light on” - Aha, I was missing a delay after the “light off!”, so the light turned off and on so quickly I couldn't see it!



**2. Test one change at a time.** If you make several changes at once, you may not know which change actually fixes the problem. Also, sometimes changes can introduce new bugs, so even if one change fixed the bug, the other change could have broken it again!



### 3. Reduce your code to the simplest possible version to isolate a bug.

Just as we broke our Love to Code projects into power, circuit, upload and code portions to simplify debugging, we can break a program down into smaller pieces of code so it's easier to work with.



For example, if you have a program that is blinking five lights, but one of those lights is not working, save your program into a new file, and simplify it so that it controls only the one light that's having problems.



Once you figure out what's going on with the simple code for one light, you can refer to your original version and apply the fixes, testing each change as you go.



# CODE (CONT'D)

A useful tool for finding and fixing bugs is to add a little extra code in your program that helps monitor the Chibi Chip's progress in running your code. For example, we could insert a few lines of code that turn an LED on and off at a specific point in a program.



If the LED blinks, that means the Chibi Chip is able to run up to that part of the code. Likewise, if the LED doesn't blink, it means that the code leading up to the blink isn't being run. This way, we can use the blinking LED as an indicator for tracing through our code. If possible, try to use an LED that you aren't already using in your project! Below is a starting point for a blinkometer that you might find handy:

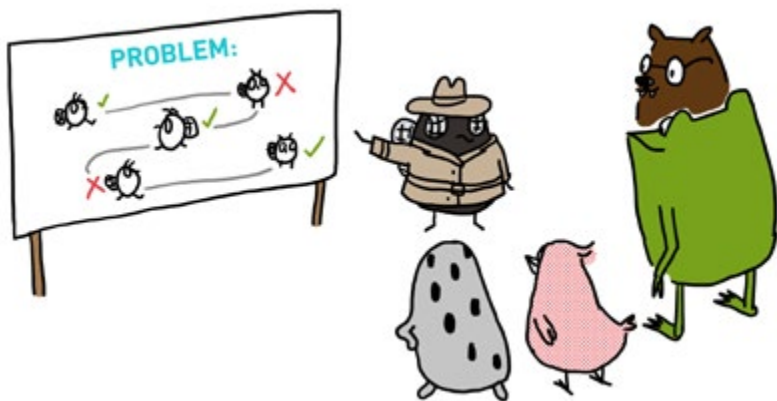
```
// these five lines of code are a blinkometer you can insert
// in your program to see how far it has run!
outputMode(4); // ensure pin 4 can drive an LED
on(4);         // blink pin 4 on and off
pause(500);
off(4);
pause(500);
```

The whole process looks like this:

- 1) Start from the very beginning and insert the LED blink code** to establish that uploading is working, and that the blink code works.
- 2) Move the blink code a few lines down** and upload the code again.
- 3) If the blink code didn't trigger**, look before that point for clues on why it didn't get there.
- 4) If the blink code did trigger**, move the blink code a little further down in the program and upload again.
- 5) Repeat steps 2-4 until you've found all your bugs!**



Finally, debugging a program is not something you have to do alone. **Try explaining the problem you are having out loud to someone else.** Often you will catch the issues even before you finish explaining. If not, the other person may be able to offer suggestions or ask helpful questions. This technique is effective even if the other person hasn't coded before! In fact, some programmers use a technique called "rubber duck debugging" where they first try explaining their problems to a rubber duck.



Instead of simply saying "my code is not working" or "my code is broken!", break your story down into specific intentions and observations: "this is what I expected my code to do, but it's doing this instead." This helps clarify what you're trying to debug, and will also make it easier for someone else to help you.

## CODE (CONT'D)



Debugging is an important programming skill! Don't worry if it takes you a while to solve a problem, you are building useful skills in the process, while growing to understand coding better!

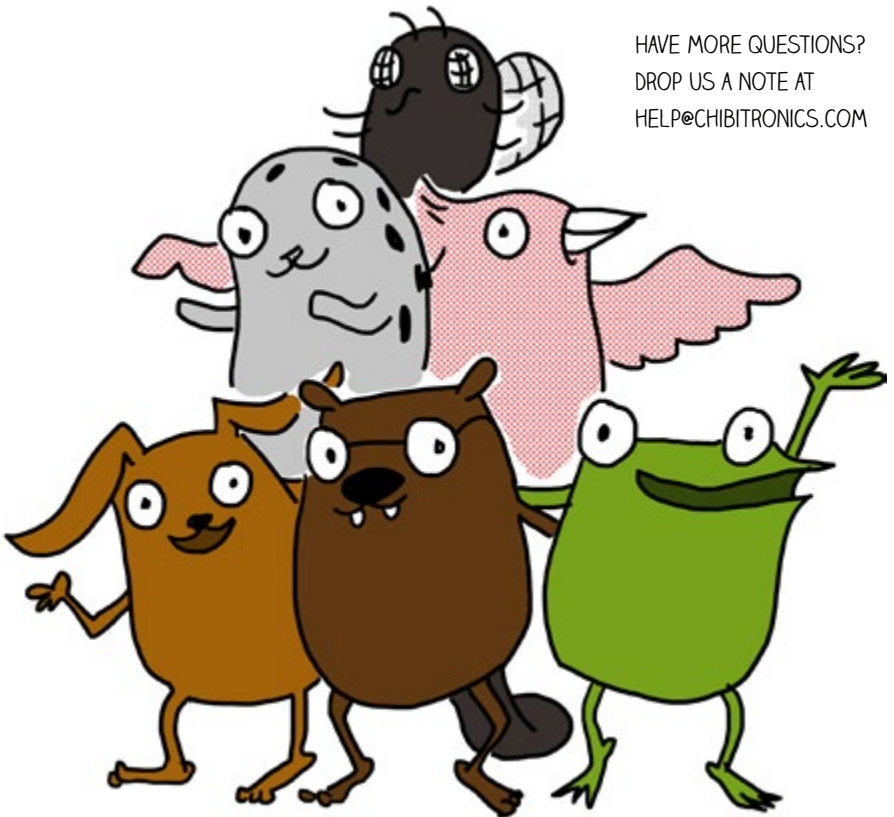
## ...AND BEYOND

The debugging suggestions we shared in this chapter are only some of the most common tips meant to help you get started with the debugging process. For more debugging resources, check out our website at **[chibitronics.com/lovetocode](http://chibitronics.com/lovetocode)**.



Don't worry if it's frustrating at first. Debugging is a skill that takes some patience but you get better at it as you debug more projects. It can also be fun to create (and debug) with a friend. Fresh eyes can catch things that we don't see ourselves, so reach out and ask others for help too!

HAVE MORE QUESTIONS?  
DROP US A NOTE AT  
[HELP@CHIBITRONICS.COM](mailto:HELP@CHIBITRONICS.COM)



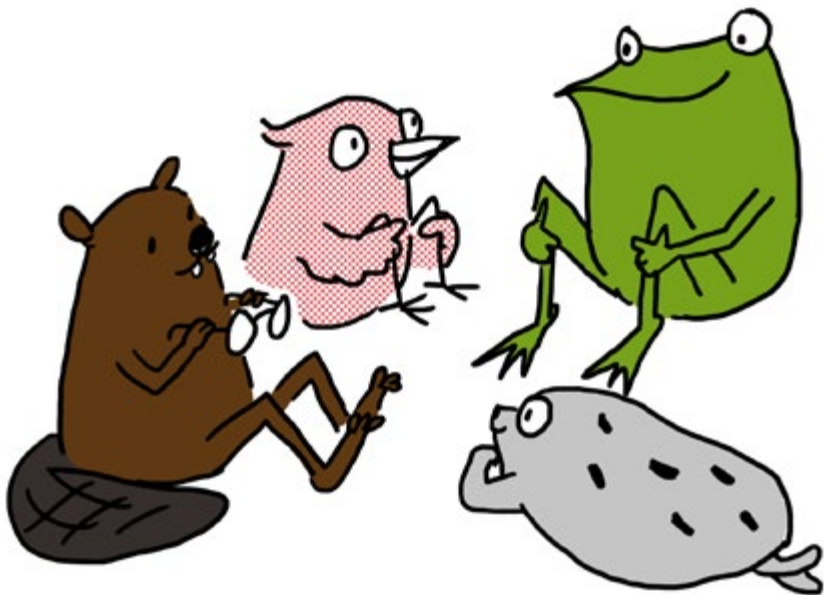




# Conclusion

“We’ve made so much stuff!” said Fern.  
“You’re the one who did it all,” said Edith.  
“We just helped,” said Carmen.  
“Even though you thought you couldn’t,” whispered Sami.

Fern smiled.



Together we've learned how to:



Turn on an LED light using  
our Chibi Chip

Add multiple LEDs to a project  
and make them all blink



Control our lights with various  
kinds of crafted switches

Fade lights in and out gradually,  
and make fun patterns with the  
LEDs using multithreading



Debug and fix our projects when  
something doesn't work!

"This is just the tip of the iceberg," said Sami. "We can do so much more!"



Ready for more? Here's a preview of other adventures waiting for you to take with Fern and friends:



Control circuits using a light sensor!  
Learn how to make projects that can respond to light and dark.



Make projects even more vivid and interactive using special LEDs that can be programmed to any color of the rainbow!



Check out:  
**[chibitronics.com/lovetocode](http://chibitronics.com/lovetocode)**  
for new chapters, new stickers, and more!

Join the party! Go to **[chibitronics.com/projects](http://chibitronics.com/projects)** to see community projects and to share your own!



Happy making and see you again soon!

# Acknowledgments



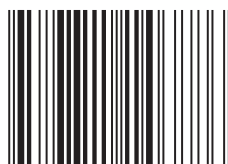
Thanks to the many people who inspired, advised, informed, reviewed and otherwise helped make this book possible:

Sean “xobs” Cross, Natalie Freed, Pauline Ng, Leah Buechley, David Mellis, Sari Widman, Amy Wibowo, Kristin Osiecki, Asli Demir, Colleen Graves, Josh Burkner, David Cole, Jeannine Huffman, Molly Adams, Jill Dawson, Nicole Fuerst, Lou Buran, Rafranz Davis, Sylvia Martinez, Ricarose Roque, Alicia Gibb, Joy Schultz, Christopher Sweeney, Susan Klimczak and the Learn 2 Teach, Teach 2 Learn group at the South End Technology Center, Ryan Jenkins and Nicole Catrett of Wonderful Idea Co, Paula Bontá and Brian Silverman of the Playful Invention Company, Mitchel Resnick, Natalie Rusk, Andrew Sliwinski, Eric Rosenbaum, Jennifer Jacobs and the rest of the thoughtful and generous team in the Lifelong Kindergarten Group at the MIT Media Lab ... and many more!









9 789811 146886 >