Love to Code: Volume 1

Written by Jie Qi
Illustrations by K-Fai Steele

Publisher: Sutajio Ko-Usagi Pte Ltd
dba Studio Kosagi, in Singapore
info@chibitronics.com

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

ISBN: 978-981-11-4688-6

9 789811 146886 >

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...

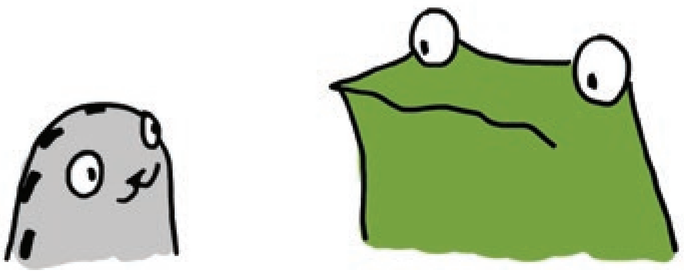
“My friends are making drawings that **do** things!”

Fern wanted to make drawings that lit up too.

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

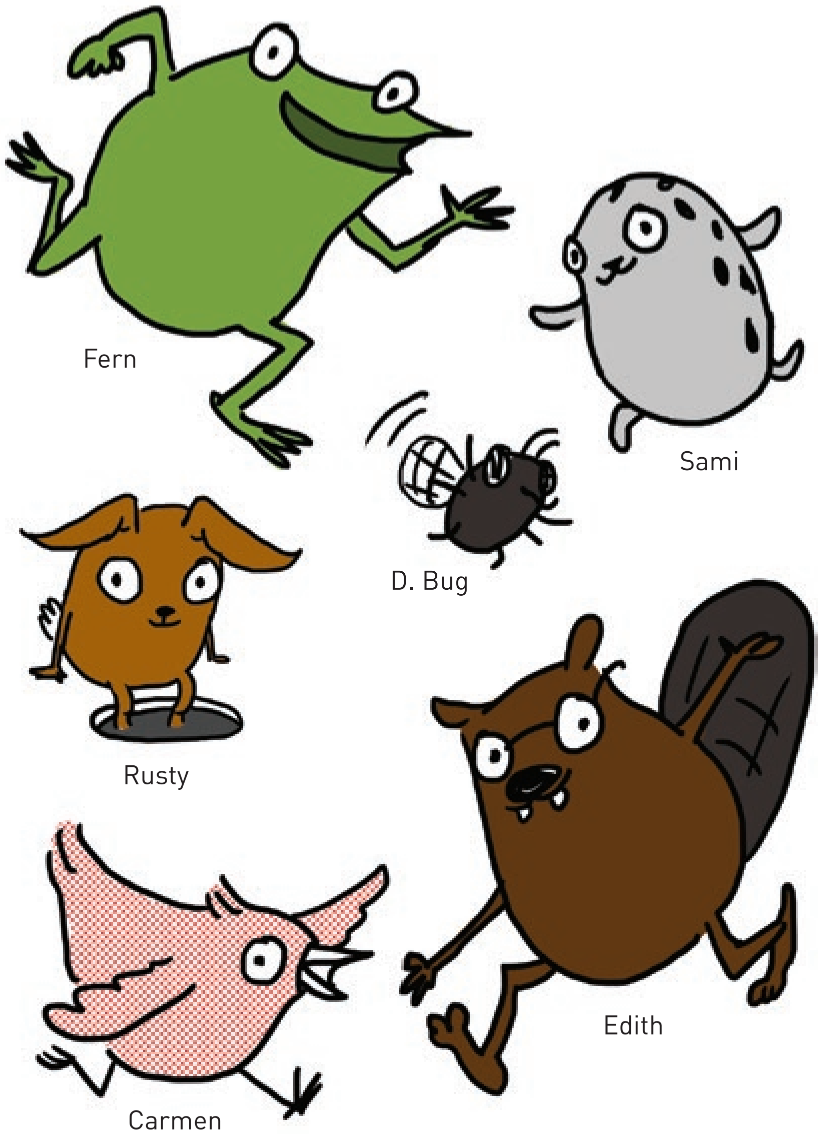
"Fern, do you want me to show you how to add lights to your drawings too?"
"Oh ... you don't have to," said Fern.
"But I can tell you want to do it. And look, you've already started!" Sami barked cheerfully.

"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!"



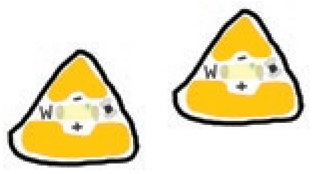
In this book, Fern gets help from not only Sami, but also from her whole gang of friends. Fern — and you — will learn how to add lights and sensors to drawings.

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



LED stickers

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

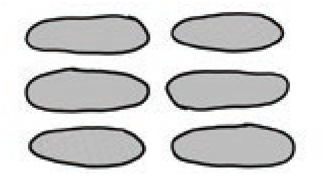


Copper tape

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

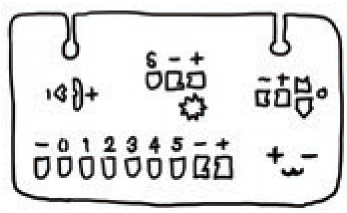
Fabric tape patches

These conductive fabric patches help us fix broken circuit connections.

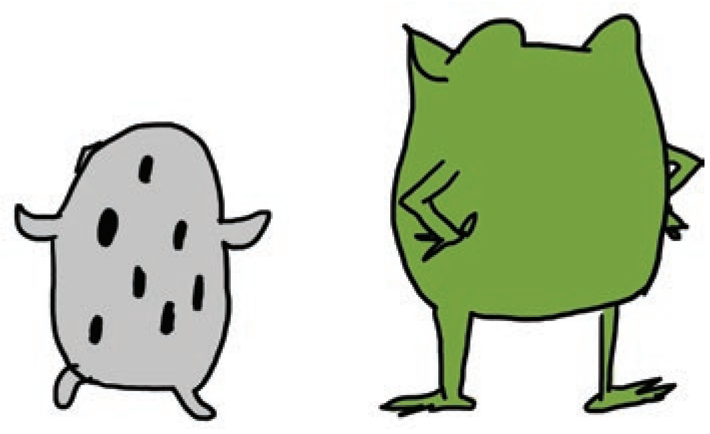


Circuit sticker stencil

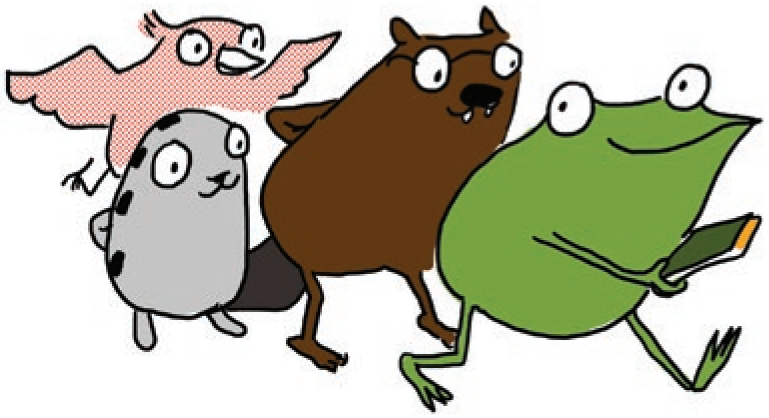
Use this stencil to help you sketch your own circuit designs!



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!

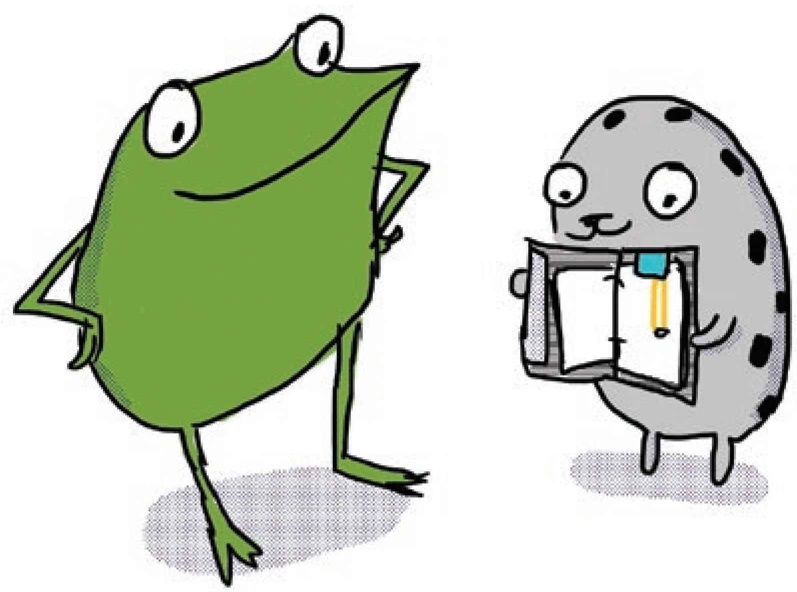


If you need help or get stuck, check out our web tutorials at:
chibitronics.com/lovetocode

Or send a detailed description of your problem to **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!”

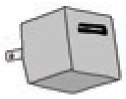
TURN ON A LIGHT!

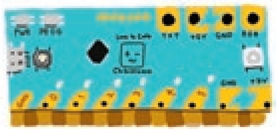
In this activity we will power up our Chibi Chip and use it to light up an LED sticker!

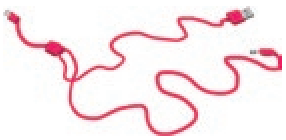
We will need:

Chibi Light LED sticker

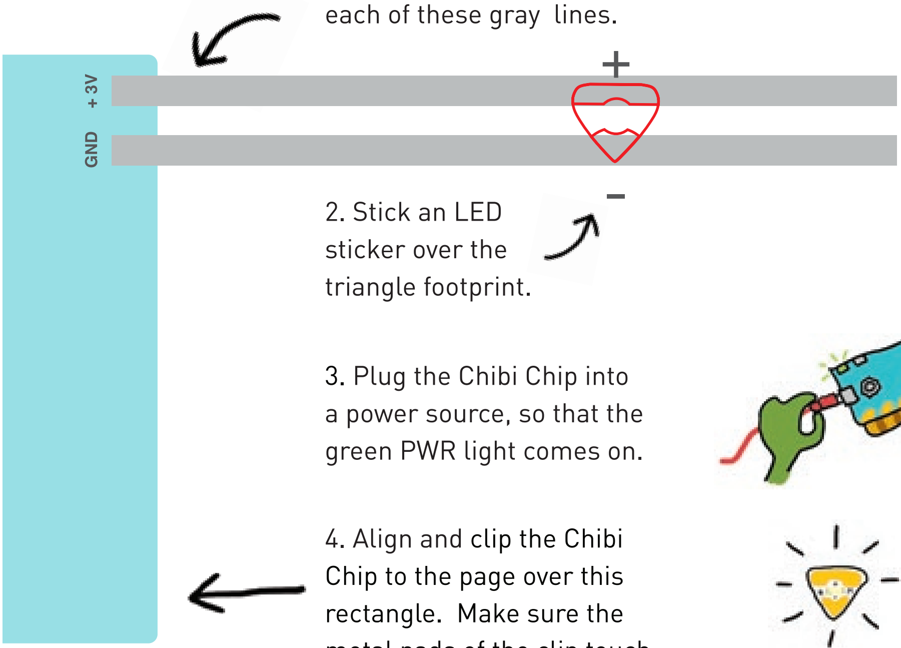
copper tape


USB power
(Chibi Book, laptop or wall plug)


Chibi Chip

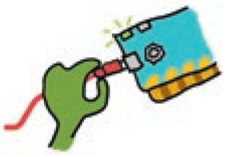

USB cable

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.

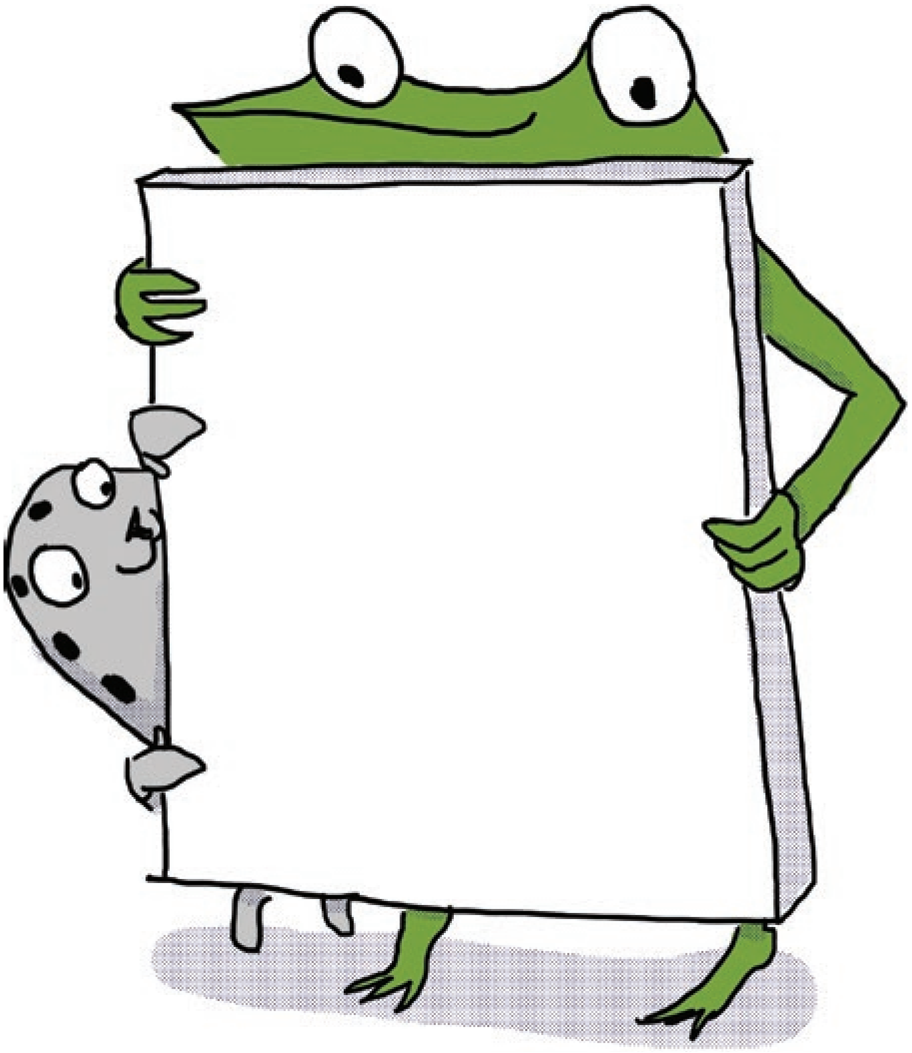


4. Align and clip the Chibi Chip to the page over this rectangle. Make sure the metal pads of the clip touch your circuit.

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

“You can see the LED’s light through the paper!” exclaimed Sami. “Fern, let’s draw something around the light.”

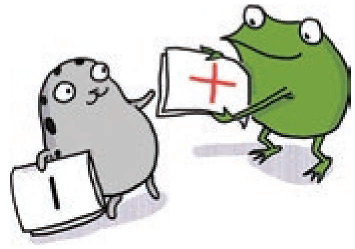
What should Fern draw?



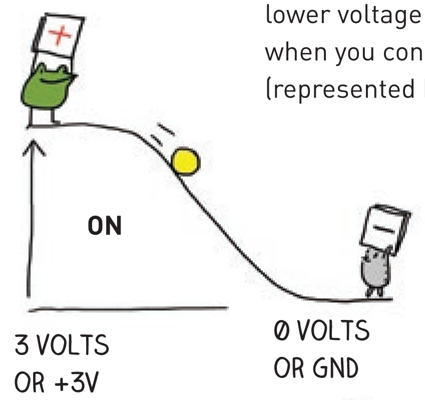
DOWN THE RABBIT HOLE: LED CIRCUITS

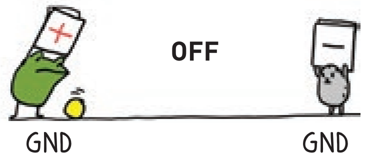
Electricity is a form of **energy** that flows to turn on your light. This flow of energy is called a **current**. Like a ball rolling down a hill, a current will only flow when there is a difference in height, from high to low.

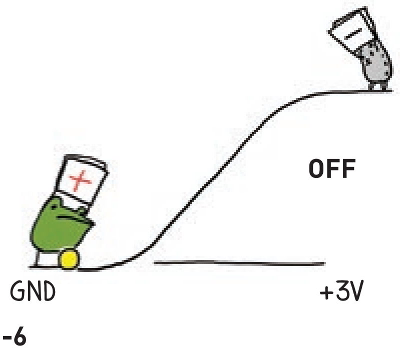
In electronics, the difference in height is called a **voltage**. Voltage is a measurement of potential energy, just like the height of a ball on a hill.

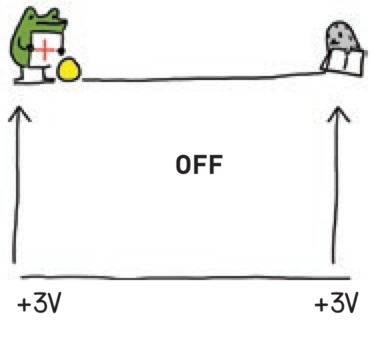


For an LED to turn on, current must flow through the LED from a higher voltage on the (+) side to a lower voltage on the (-) side. Here's what happens when you connect the (+) and (-) ends of your LED (represented by Fern and Sami) to different voltages:

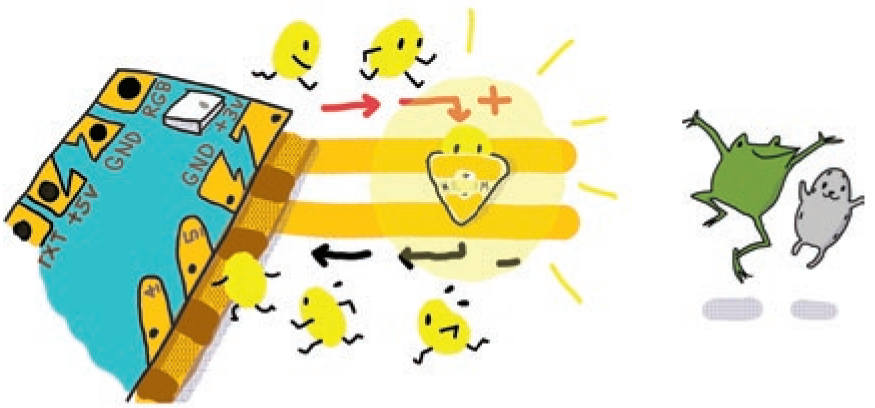








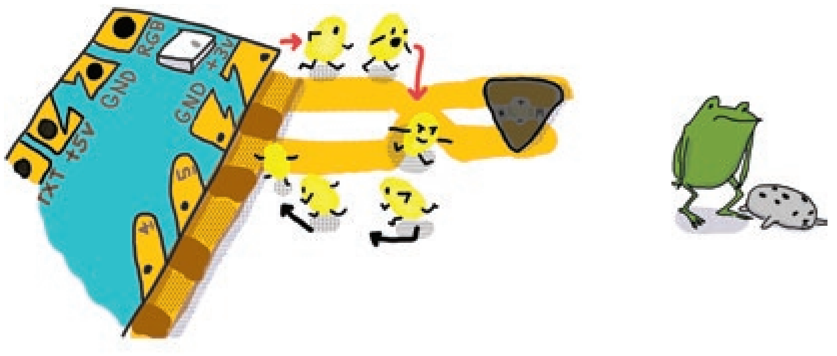
To summarize: when we connected the (+) side of the LED sticker to 3 volts and the (-) side to GND (or 0 volts), we provided enough voltage in the right direction for the LED to turn on!

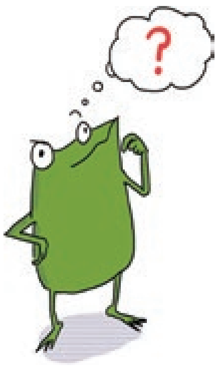


Don't worry if you accidentally connect the LED backwards. Nothing bad will happen, it will just not turn on.

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.

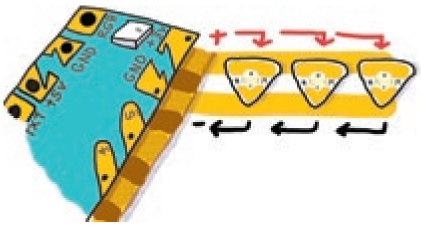


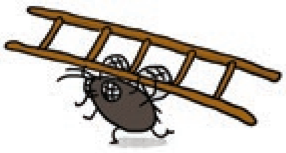


HOW CAN WE TURN ON A BUNCH OF LIGHTS?

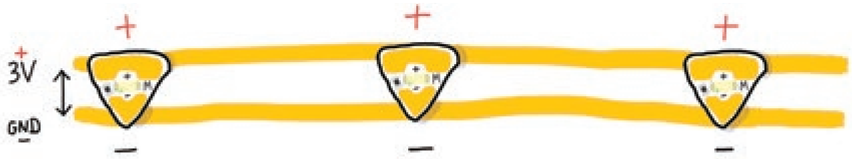
Here are a couple of ideas:

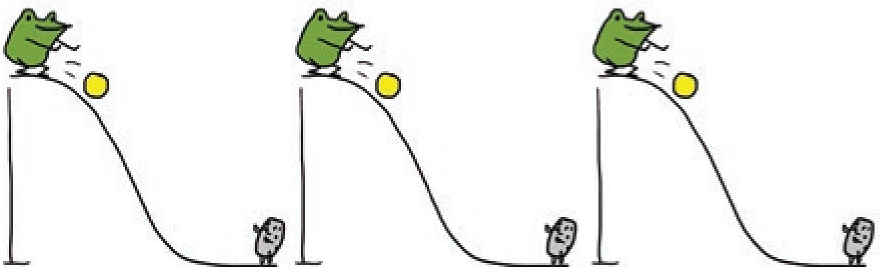
Parallel Circuits
Parallel circuits are where each LED shares a trace between +3V and GND. To connect LEDs in parallel, stick the LEDs next to each other between a (+) and a (-) track, like rungs on a ladder. You can turn the circuit on page **1-2** into a parallel circuit by just adding LEDs!





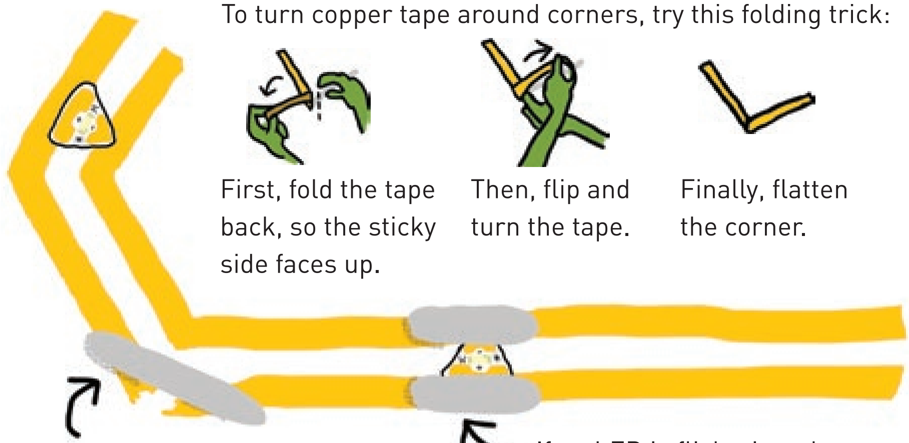
As the illustration above shows, all of the (+)'s go to the +3V pin, and all of the (-)'s connect to ground. This way the power source provides enough voltage ("height") for each LED. Since each LED needs at least 2.5V, connecting them in parallel to the +3V pin will turn all the LEDs on.



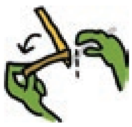


LET'S PLAY!

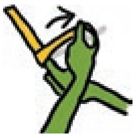
It's now time to design your own scene! Before we get started, here are some tips and tricks on how to use copper tape.



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



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

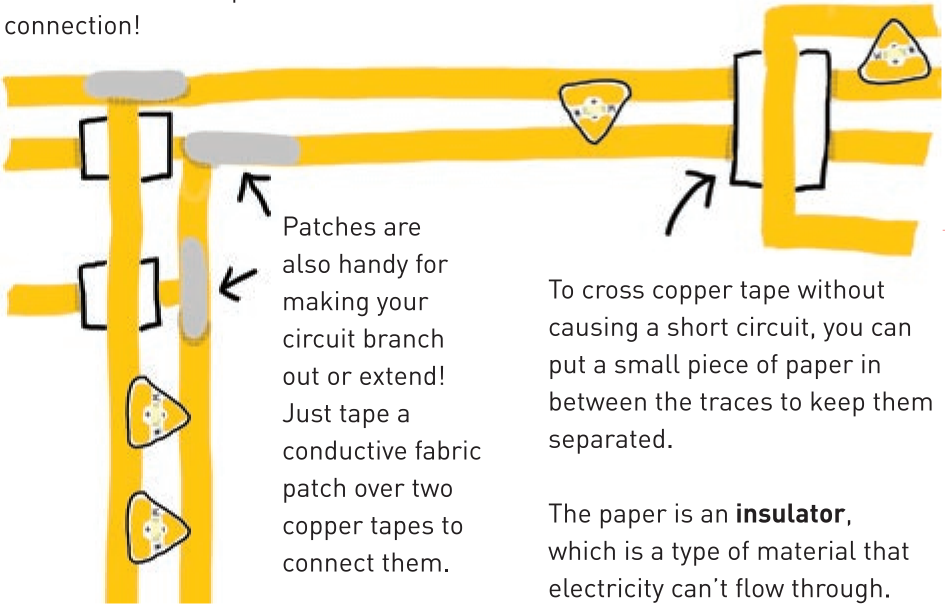


Then, flip and turn the tape.

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.

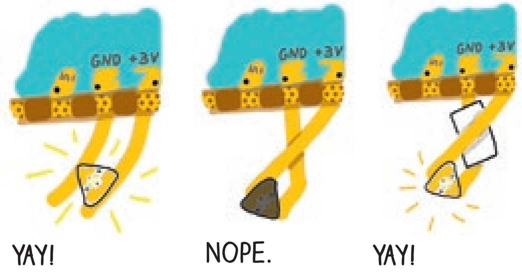
To cross copper tape without causing a short circuit, you can put a small piece of paper in between the traces to keep them separated.

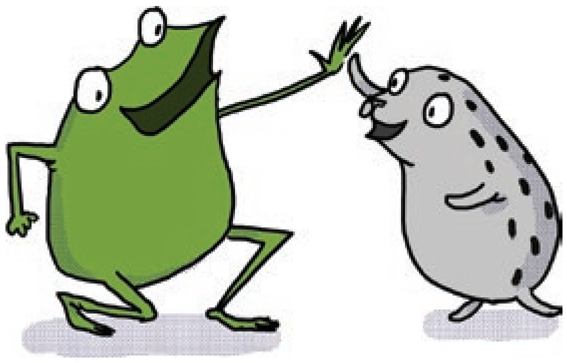
The paper is an **insulator**, which is a type of material that electricity can't flow through.

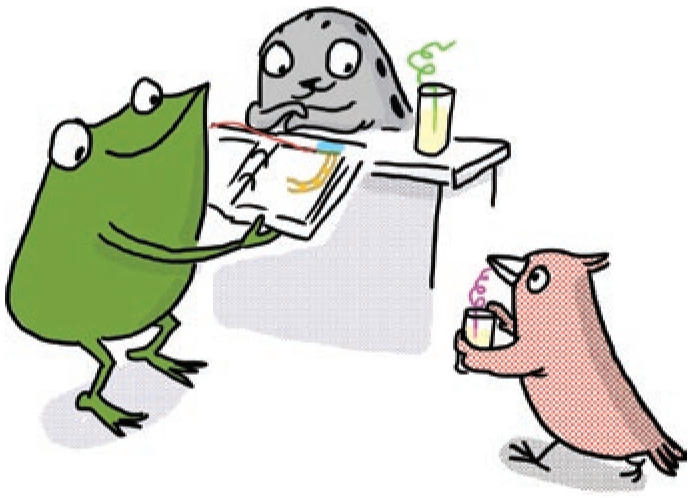
Look on the facing page! It's a magical flower pot! What's growing? What's glowing?

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!





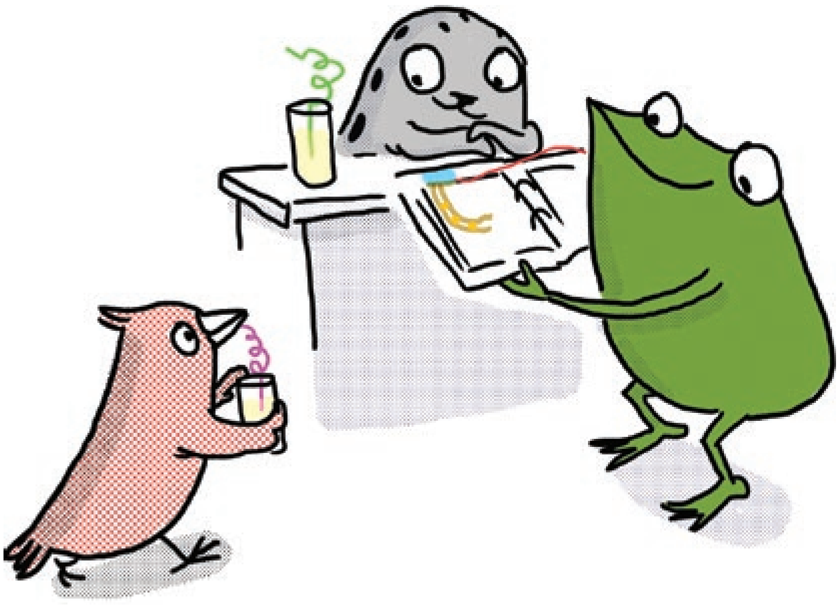


“Yay! This feels like magic!” Fern exclaimed.

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!



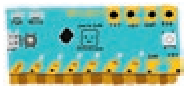
"Wow, if those lights could blink, then my drawings would really come alive!" said Fern enthusiastically. "Can I really do that?"

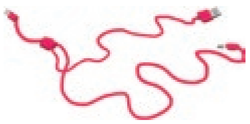
"We can control lights by writing programs, through a process called coding," said Carmen the bird. "Let's get started!"

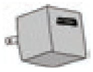
LET'S GET CODING!

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

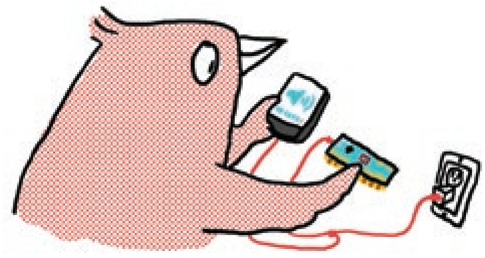

USB power
(Chibi Book, laptop or
wall plug)

Internet
connection

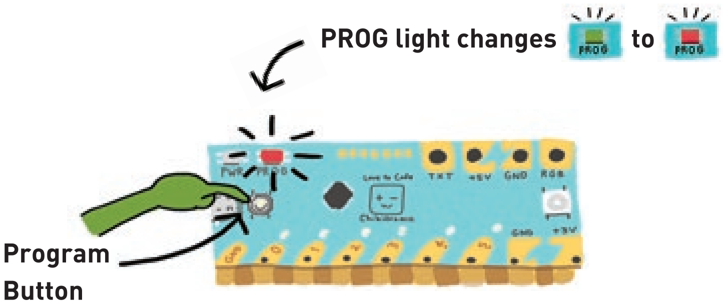
A device with a web browser (phone, computer, or tablet): this is the programming device

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.

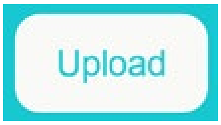
4. **Turn the volume up to the max** on the programming device and make sure it's not muted. Since code is uploaded to the Chibi Chip through the audio cable, we need to turn the volume up to 100% so the Chibi Chip can hear the code loud and clear!



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.

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
}
```

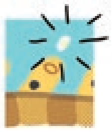
In the loop function we have three different kinds of instructions: `on()`, `pause()`, and `off()`. These three kinds of instructions allow us to blink a light by turning it on, pausing, then turning it off and pausing again.

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.

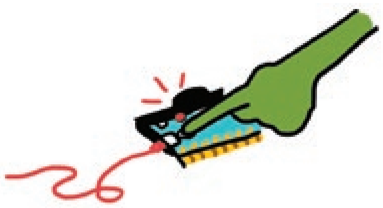
PLAY WITH THE CODE

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.

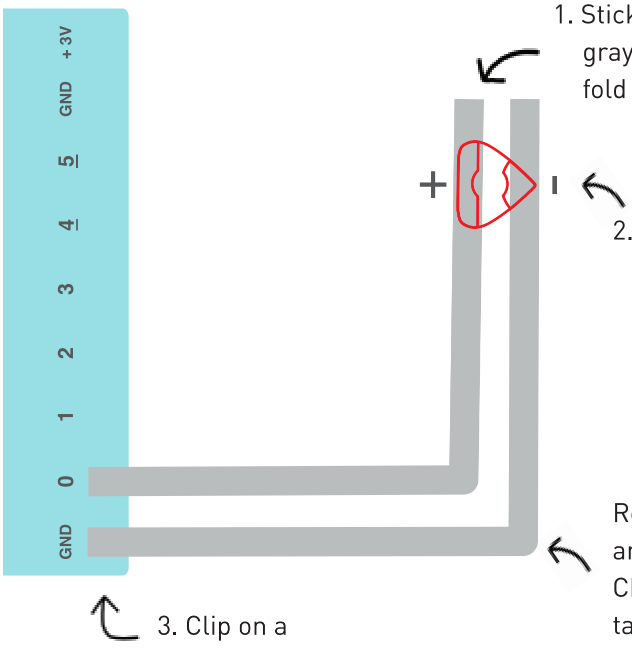


REMEMBER TO PRESS THE PROG BUTTON ON YOUR CHIBI CHIP BEFORE CLICKING THE UPLOAD BUTTON, SO IT KNOWS TO LISTEN FOR CODE!

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?

PROGRAM THIS CIRCUIT

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



1. Stick copper tape on the gray lines. Remember to fold the tape at the turns!

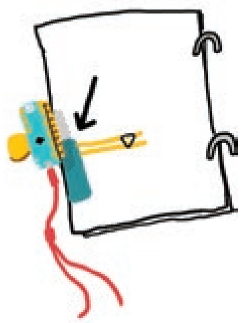
2. Stick down a LED.

3. Clip on a Chibi Chip.

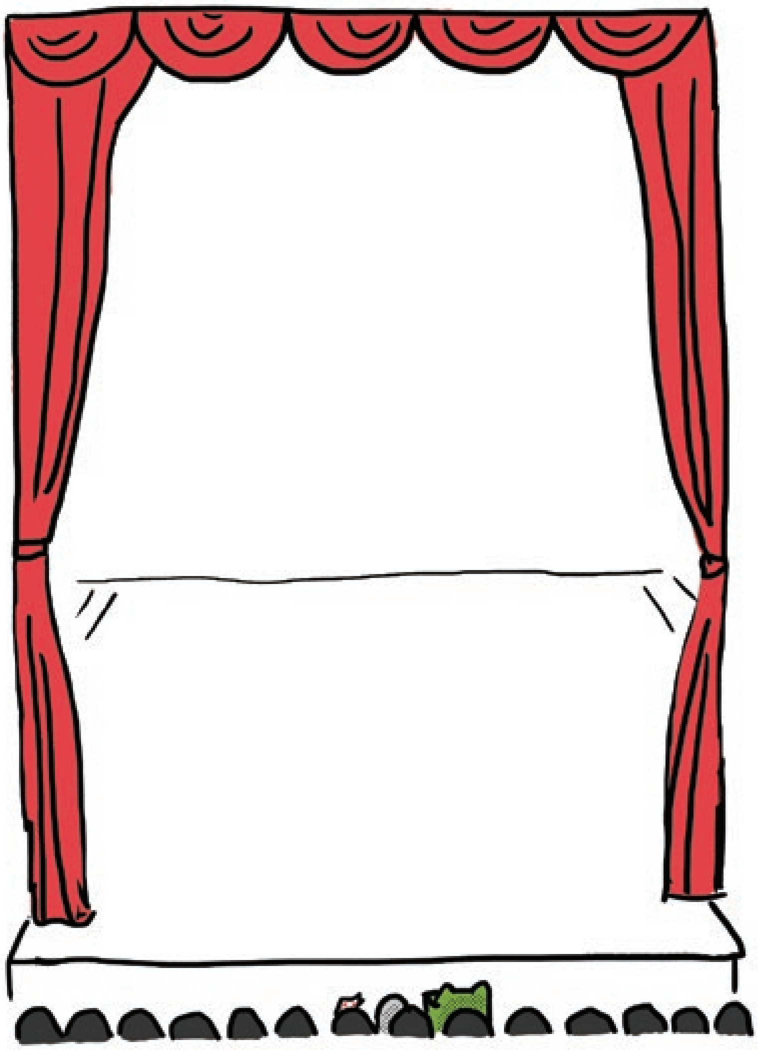
Remember to fold your tape and not cut or tear at the turns. Check out the facing page for a tape folding trick to make neat turns!

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



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:

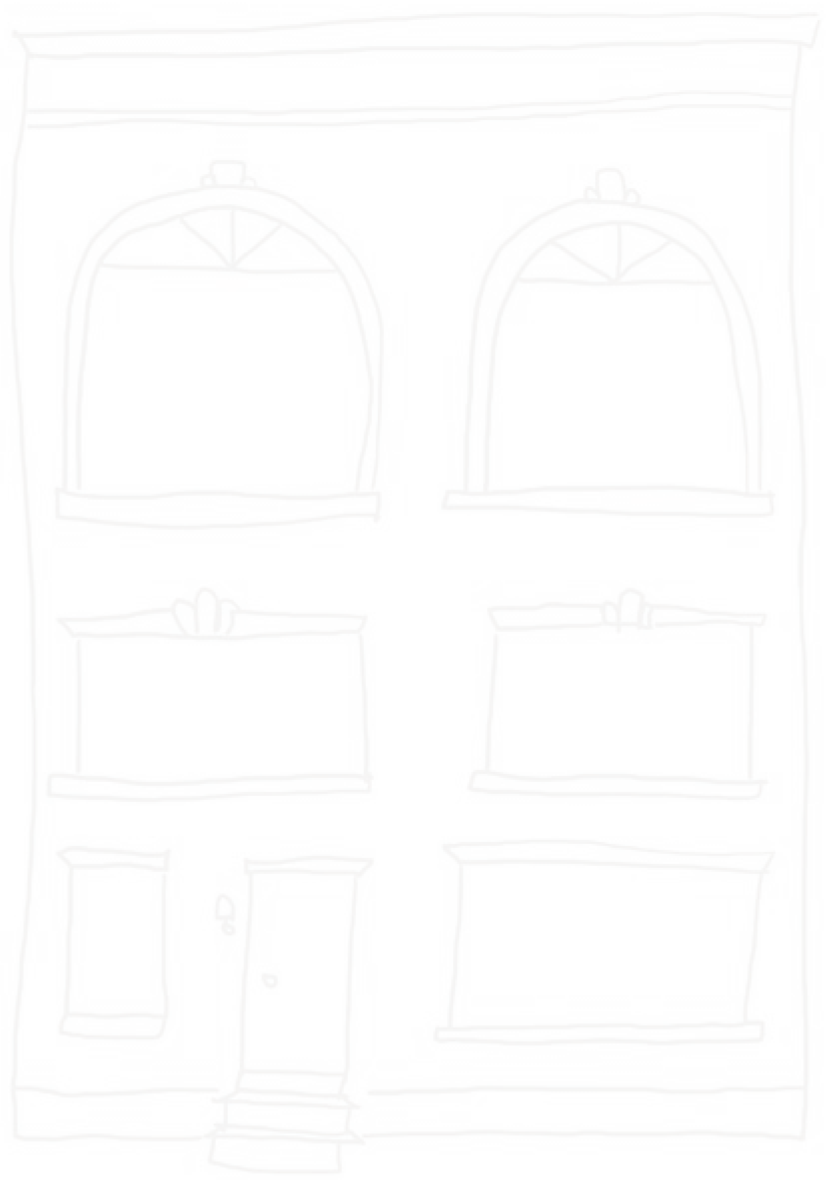
```
void loop() {
  on(0);
  pause(1000);
  off(0);
  pause(1000);
  on(3);
  pause(1000);
  off(3);
  pause(1000);
}
```

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);
}
```

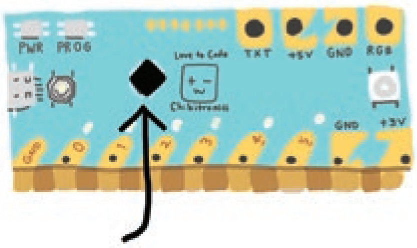
Even though pin 3 and pin 0 are controlled with statements on different lines, the Chibi Chip goes from one step to the next so fast that they seem to happen at the same time! Try adding or modifying `pause()` statements to make your own patterns.

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!



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.

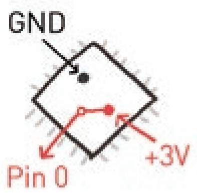


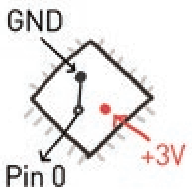
The microcontroller runs instructions coded by your program. Some of these instructions change how the pins behave.

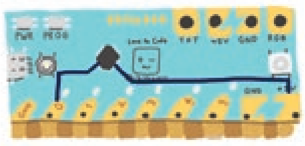
Unlike the +3V and GND pads, which are permanently wired to +3V and GND, all the numbered pins can be programmed by statements in our code to connect to +3V, GND, or neither. All of this happens inside the tiny microcontroller!

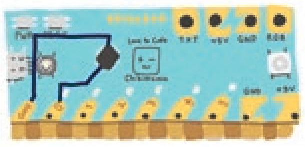
```
on(0);
```

```
off(0);
```



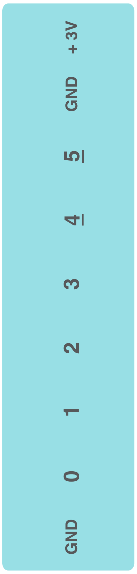


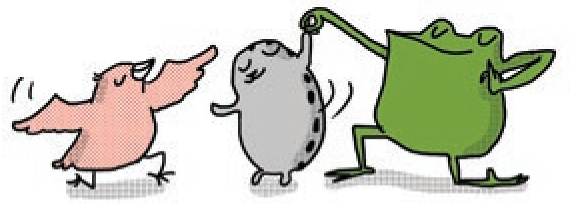




LET'S PLAY!

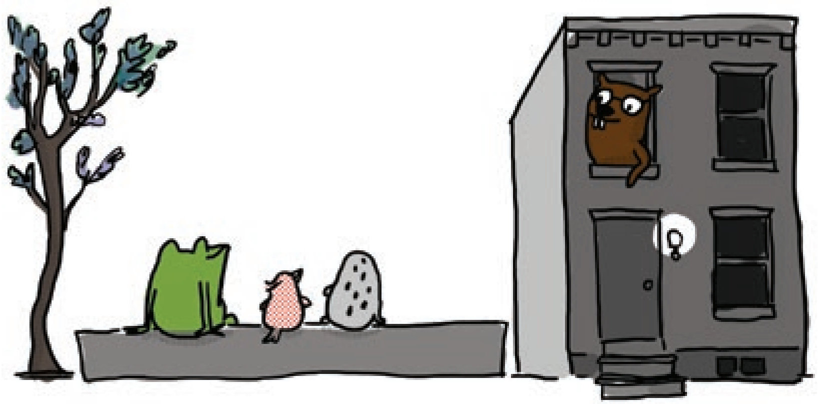
Now it's your turn to design your own circuit and light up the night sky (turn to page **2-24** for a preview).





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!



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.

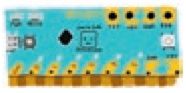
Chapter 3: Add a Switch!

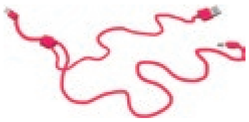
"I love building things! I can build switches out of paper and copper tape," Edith the beaver quipped. "If we add a switch to our circuit, then we can interact with the lights!"
Thwacking her tail enthusiastically, as if smacking a giant switch, Edith continued,"When someone presses the switch, we can make something happen, like blinking the lights."
"Amazing!" exclaimed Fern. "Show us how!"

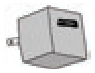
PROGRAM A SWITCH

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

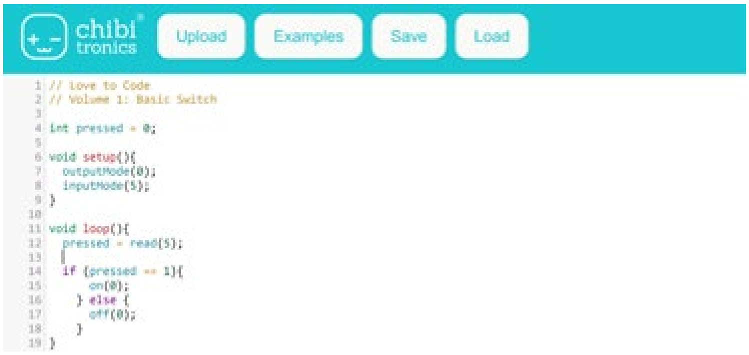
Chibi Light
LED Sticker

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**



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

int pressed = 0;

void setup(){
  outputMode(0);
  inputMode(5);
}

void loop(){
  pressed = read(5);
  |
  if (pressed == 1){
      on(0);
    } else {
      off(0);
    }
}
```

PUSHBUTTON SWITCH TEMPLATE

1. Stick copper tape over the gray lines.

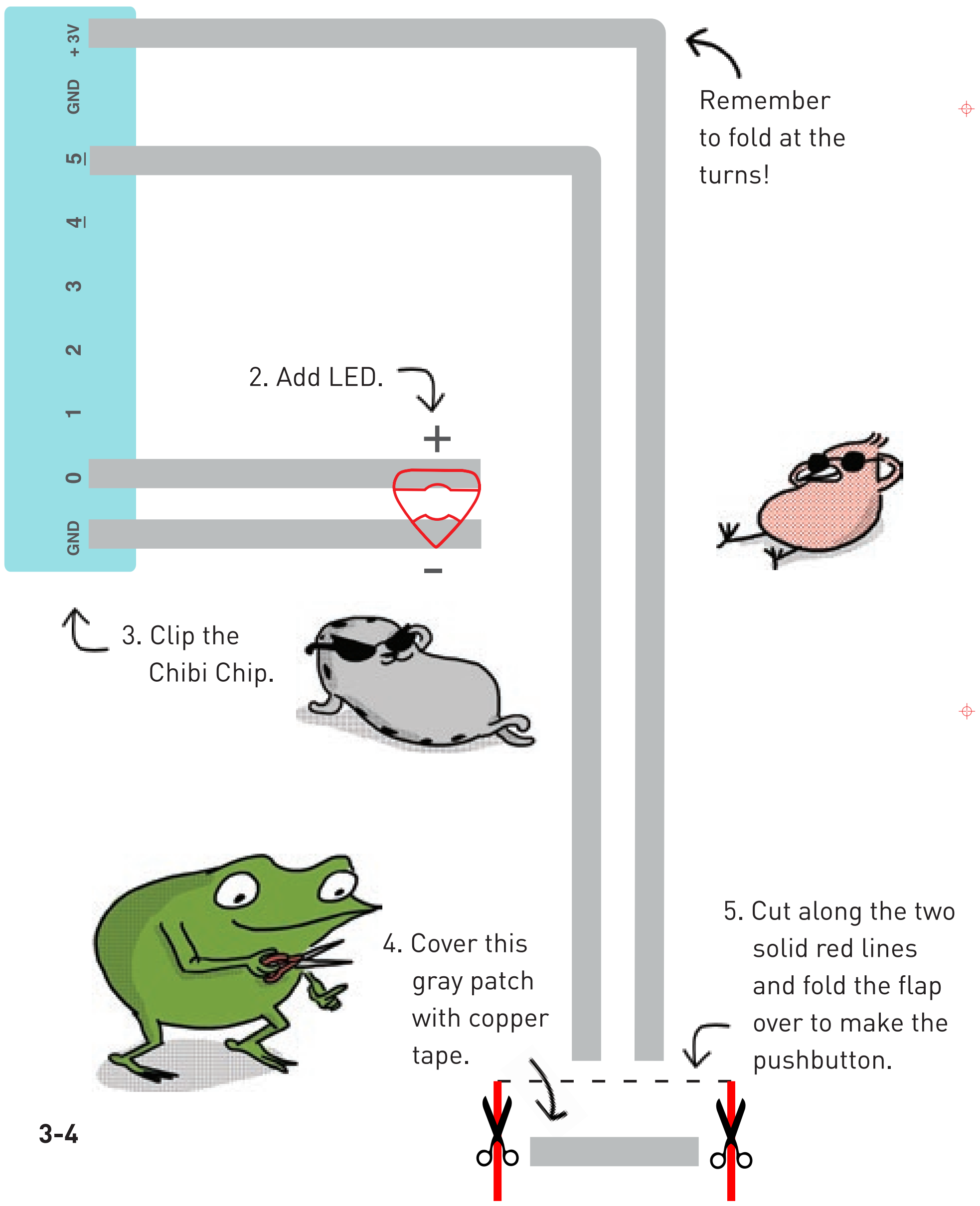
Remember to fold at the turns!

2. Add LED.

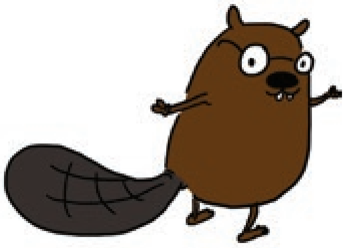
3. Clip the Chibi Chip.

4. Cover this gray patch with copper tape.

5. Cut along the two solid red lines and fold the flap over to make the pushbutton.



What happens when you
press on Edith's tail?



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:

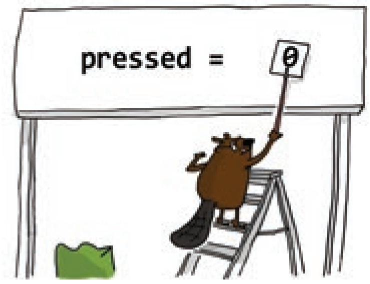
```
int pressed = 0;
```

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.

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
  }
```

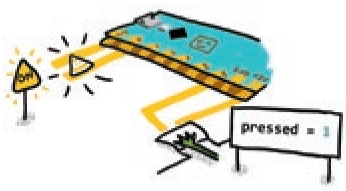
An **if()** statement allows us to change the behavior of the circuit based on the answer to a question. This question is called a **condition**.

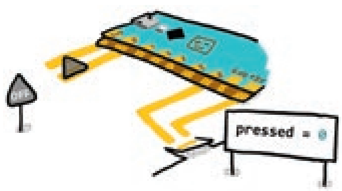
If the condition is **TRUE**, then the **Option 1 code** between the first set of curly braces **{}** runs. If the condition is not true, or **FALSE**, then the **Option 2 code** between the second set of curly braces **{}** runs.

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.

SETUP
if
OPTION 1
OPTION 2
←TRUE!
FALSE?→

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

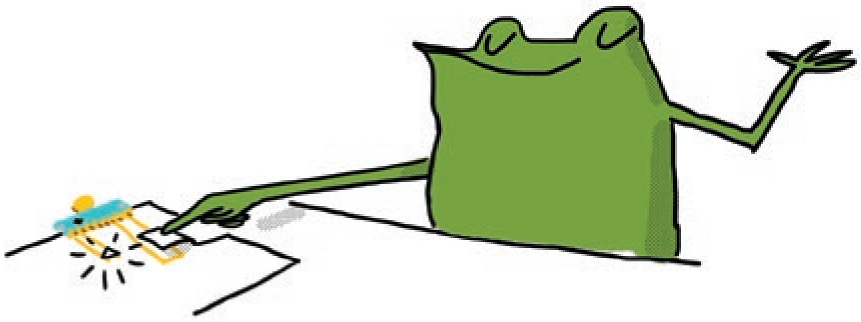
Switch is not pressed →
`pressed = 0` →
condition is **FALSE** →
turn **OFF** pin 0

SETUP
LOOP
if
ON
OFF
←TRUE
FALSE→

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!



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);
}
```

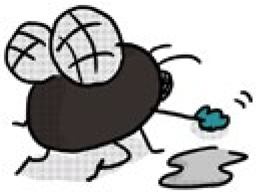
After:
New code comments out blink code, so the light stays on when the switch pressed.

```
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!

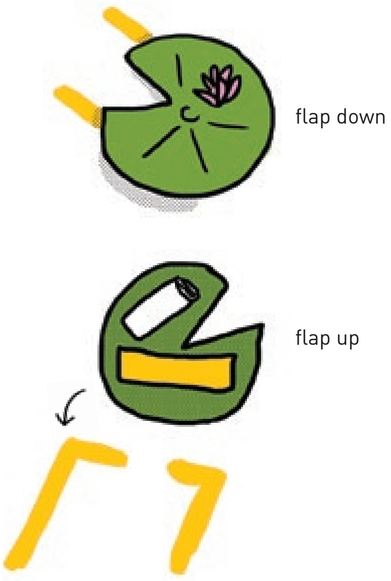
```
int starLED = 3;
int flowerLED = 4;

void setup(){
  outputMode(starLED);
  outputMode(flowerLED);
}
```

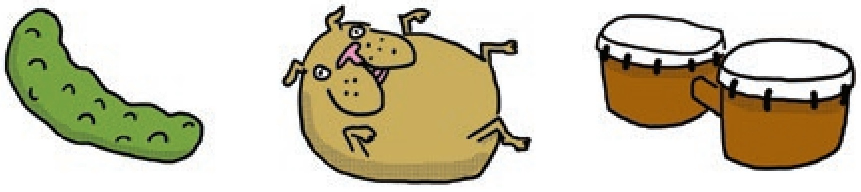
PAPERCRAFT SWITCHES

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.

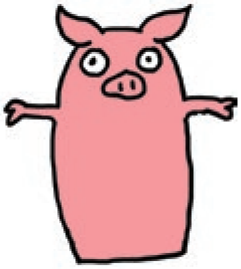
Press-the-Flap Switch
Use a flap of paper with copper tape on it to close a circuit anywhere. This way, we can make switches anywhere on the page, not just on the edge or corner of the paper! To make sure the switch doesn't accidentally press itself and turn on, insert a bit of paper or foam tape as a spacer between the flap and the circuit.
We can also use fun shapes for the flap, like some of the ones below!



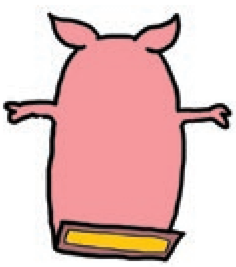


Pocket Character Switch

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.

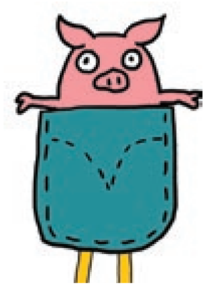


Front

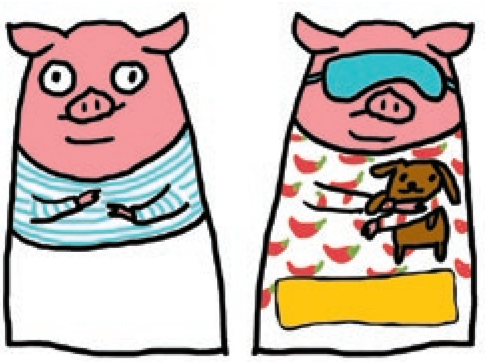


Back

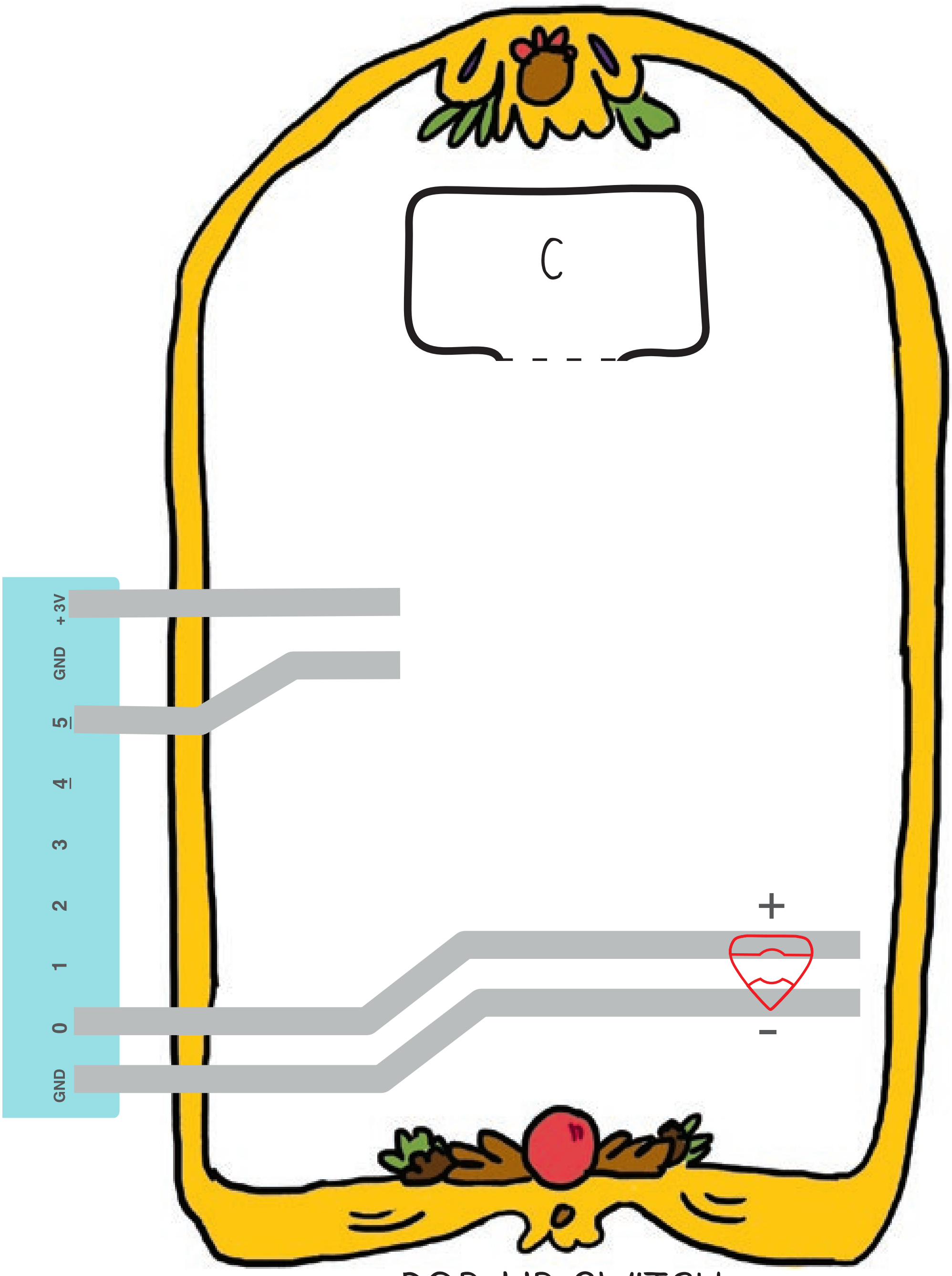
When the character is placed inside the pocket, the copper tape on the character closes the switch gap and completes the circuit!



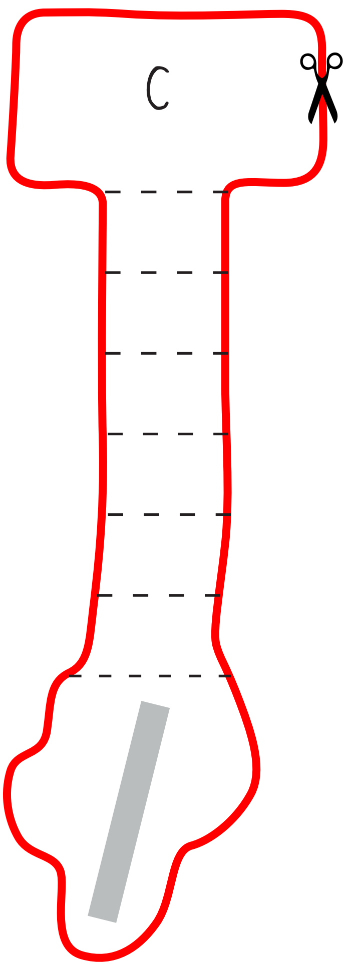
A character can work as an ON/OFF switch too! Just flip the character over so that the copper tape faces away from the circuit. That way, the switch is off even when the character is in the pocket!

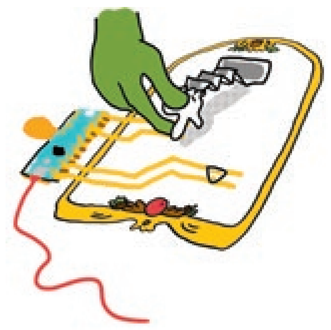


What glows when the pointer reaches the switch? Craft the switch by following the instructions on page **3-23**!

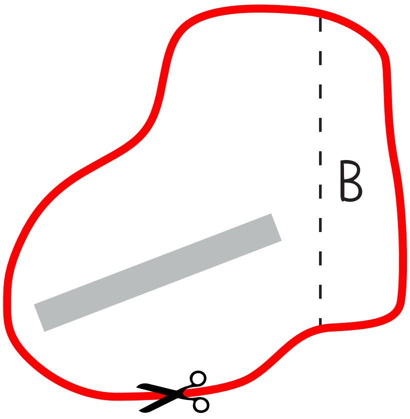


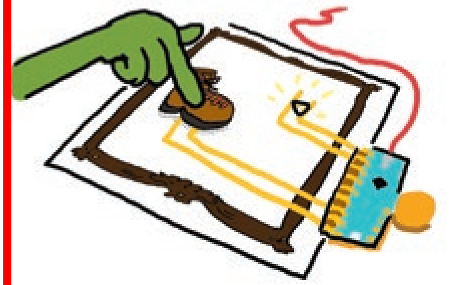
POP-UP SWITCH





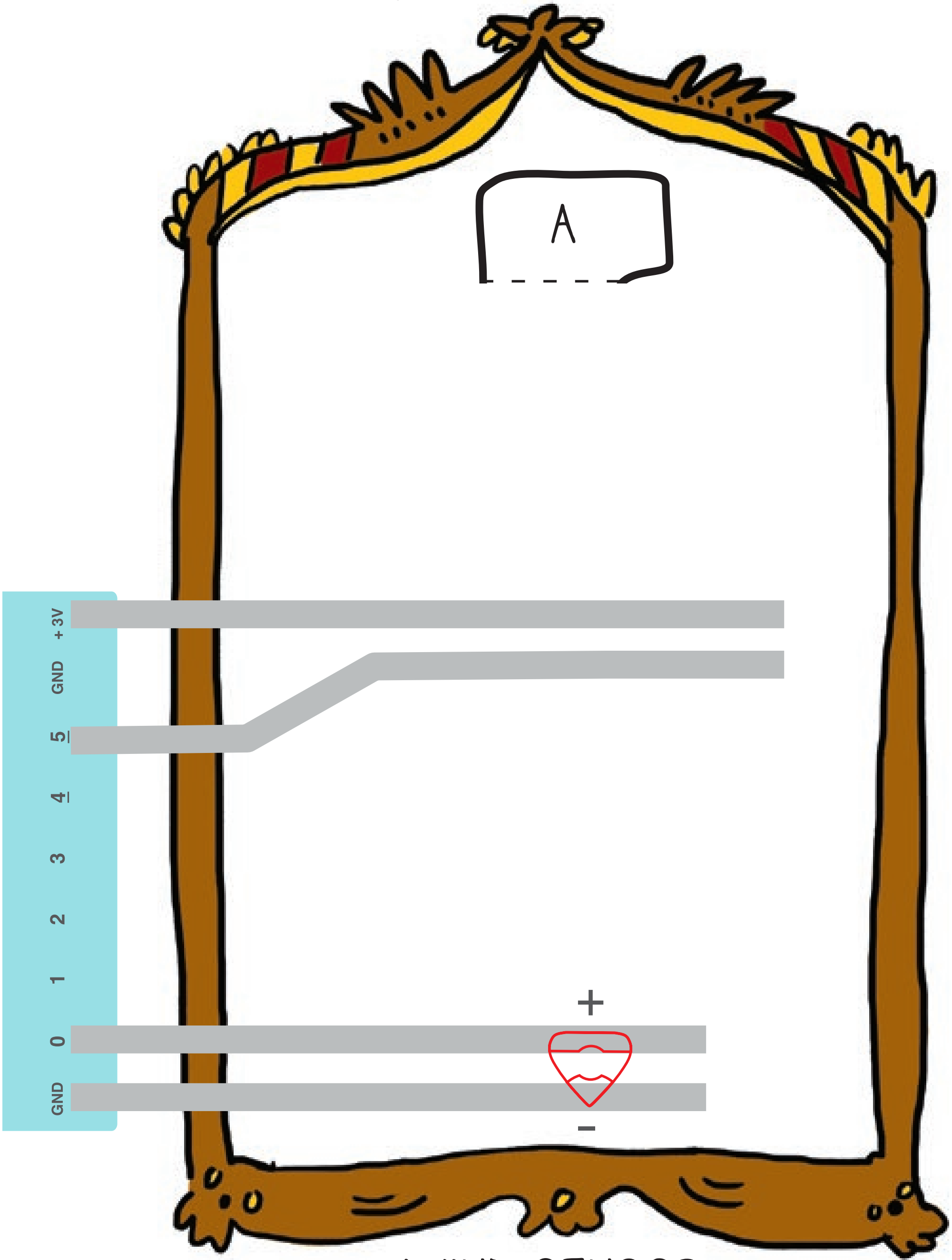
Here's what the finished pop-up switch looks like! Stretch the pointer finger so reaches the switch contacts, and then press to activate the switch.
When we let go, the pointer springs back up!



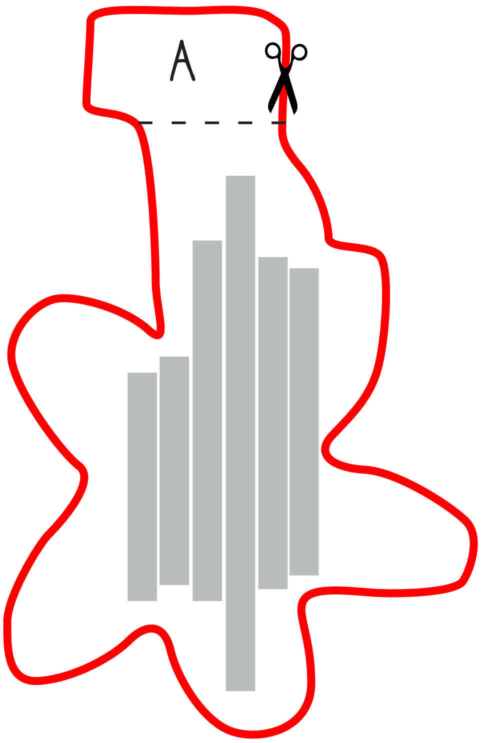


Here's what the finished press-the-flap switch looks like! Press the boot to close the switch.

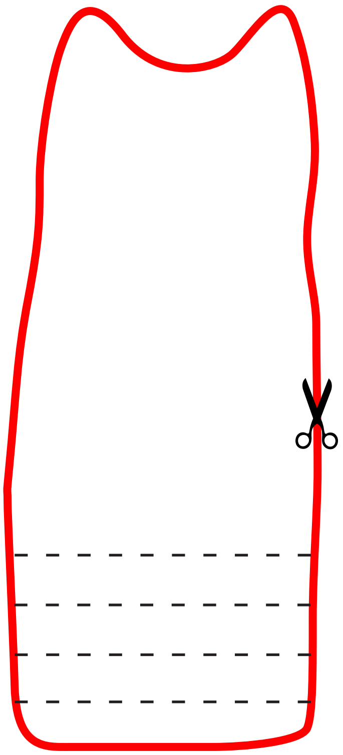
What glows when we blow on the flower? Craft the flower by following the instructions on page **3-29**!

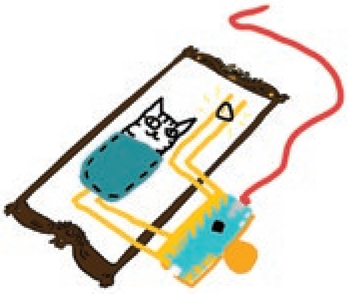


WIND SENSOR

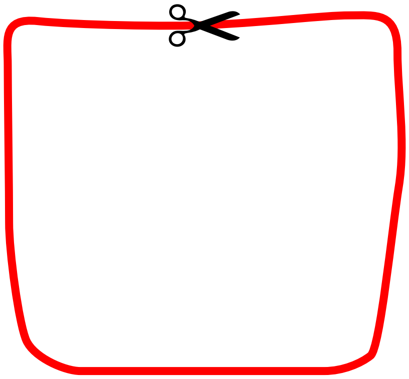


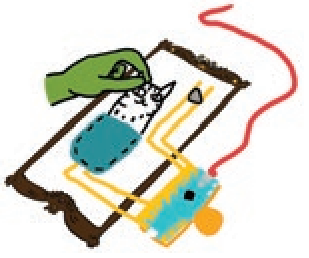
Here's what the finished wind sensor looks like! Blow on the flower to activate the switch. Try adjusting the position of the flower slightly to improve the contact between the copper tape on the flower and the paper.
You can also press on the flower to operate the switch as well!





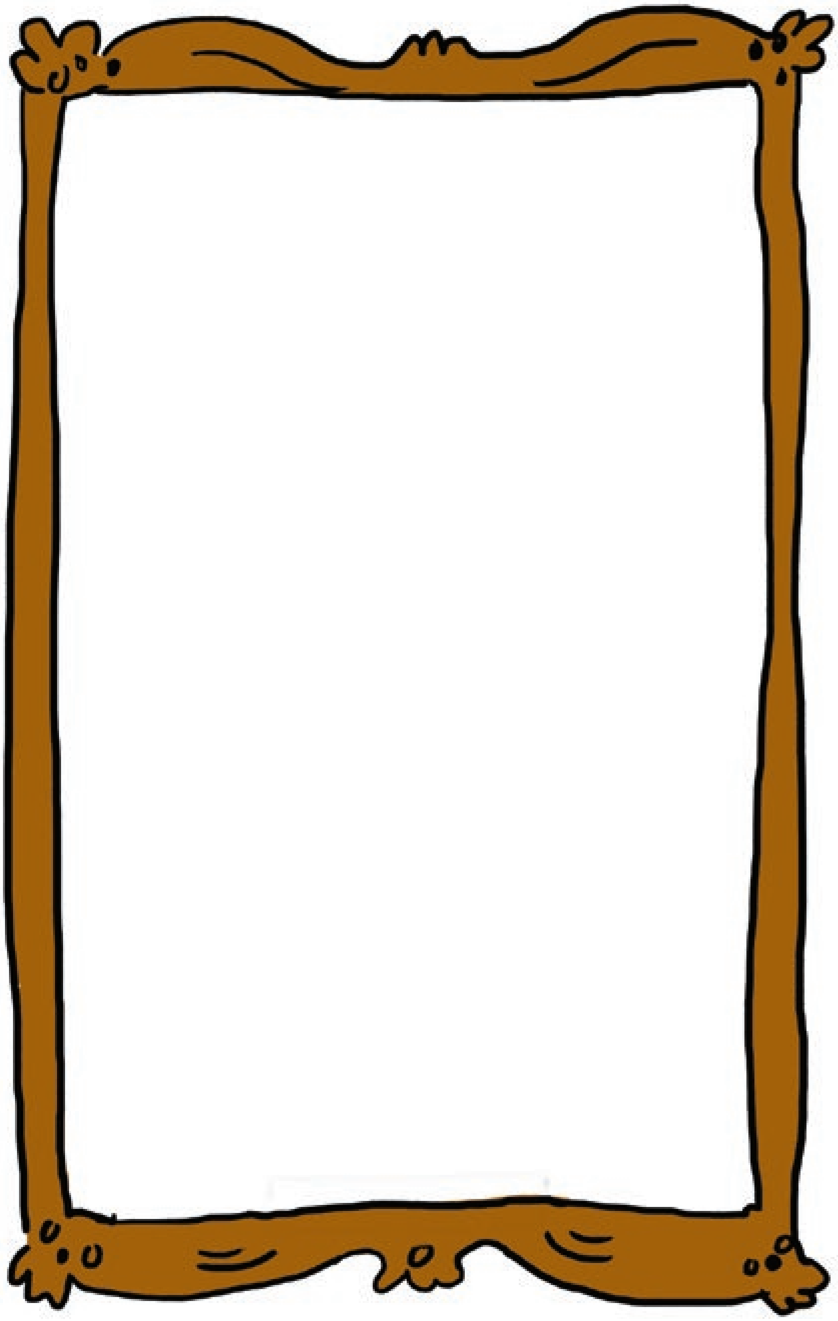
Here's what the finished pocket character switch looks like. When we put the cat in the pocket, it will close the circuit, turning the switch on. Make sure to push the cat all the way into the pocket for a secure connection.





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.

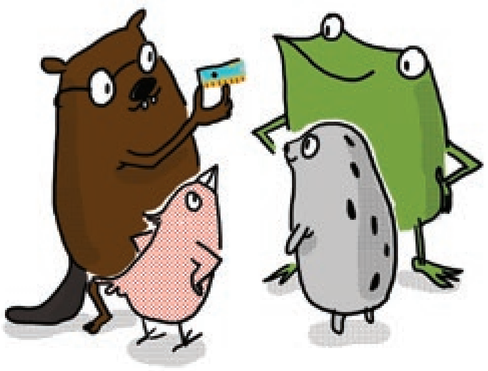
GND 0 1 2 3 4 5 GND +3V



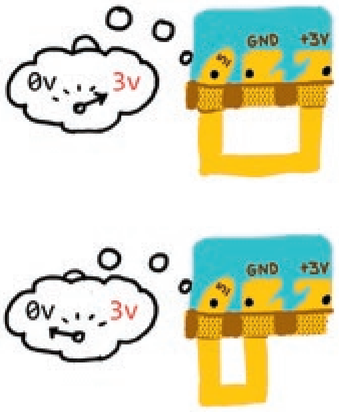
YOUR SWITCH!

DOWN THE RABBIT HOLE: INPUT - PROCESS - OUTPUT

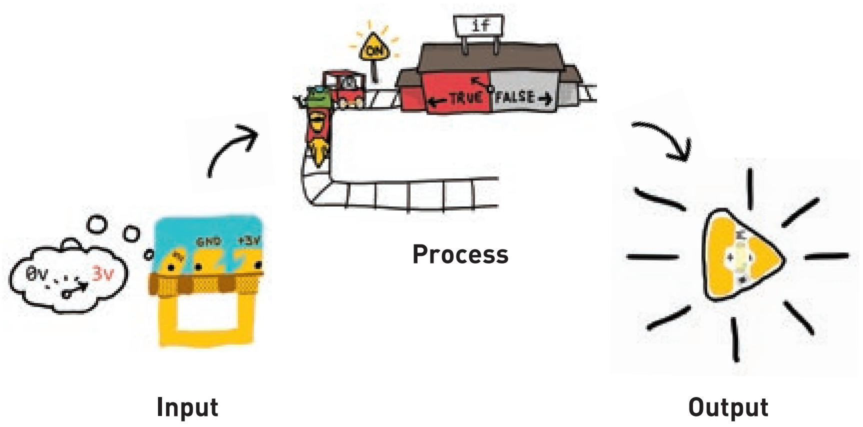
So we've played a bunch with switches now, and even designed our own! But how does the Chibi Chip actually know when to turn things on and off with the switch?



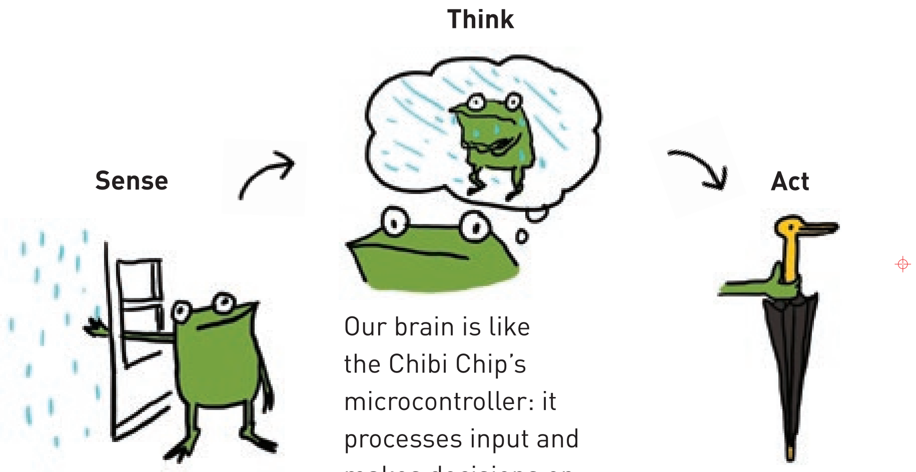
It works by taking in information from the world through the **input pin**. The pin is named "input" because information goes "in" to the board. What information, you ask? Voltage!



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:



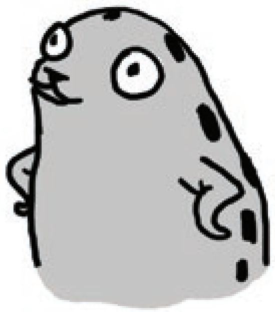
Fingers are like input pins: they sense things through touch.

Our brain is like the Chibi Chip's microcontroller: it processes input and makes decisions on what to do based on the input signal.

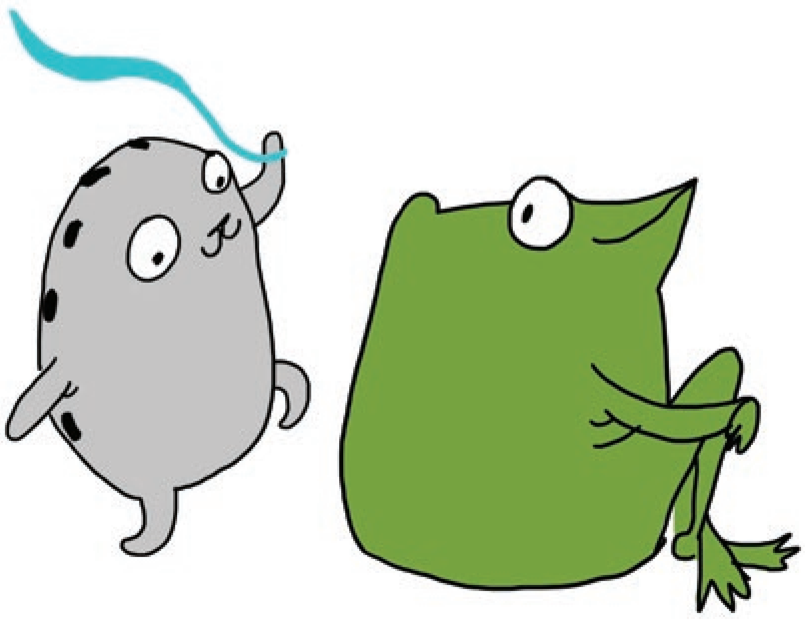
Our arms and legs are like output pins: they can change the world around us.

“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!



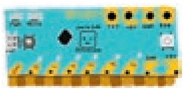
“We’ve figured out how to turn the lights on and off, but how can we make them fade slowly?” asked Fern the frog.

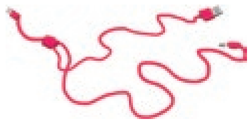
“I think I can explain!” said Sami the seal as she danced around the room. “And while we’re figuring it out, we can make dance costumes. We can have a parade later, and if we decorate our costumes with twinkling lights we will look even more fabulous!”

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!

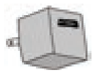
You will need:


Chibi Chip, mounted in a Clip

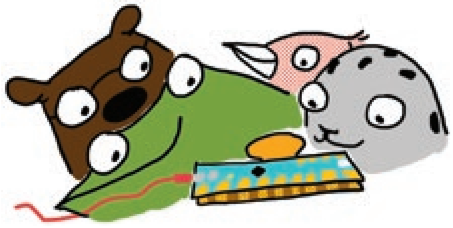

Programming and power cable

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

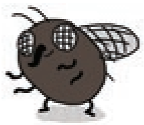
Internet connection


USB Power

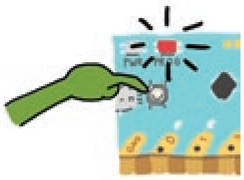
Upload the **Set Level** example code to the Chibi Chip. Go to **Examples → Love to Code Vol 1 → Set Level**. When the code is done uploading, the light on pin 0 will turn on and off by stepping through different brightness levels!



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.

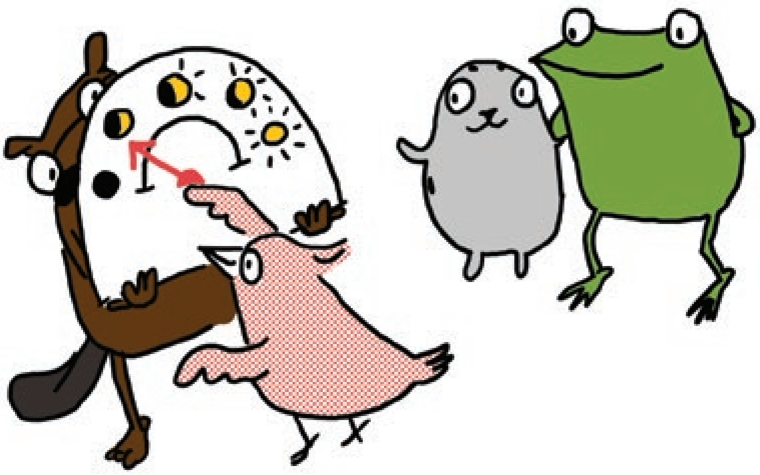
DID THE UPLOAD SOUND WAVEFORM ANIMATION APPEAR? IF NOT, TRY REFRESHING THE PAGE AND CLICKING UPLOAD AGAIN.

`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:

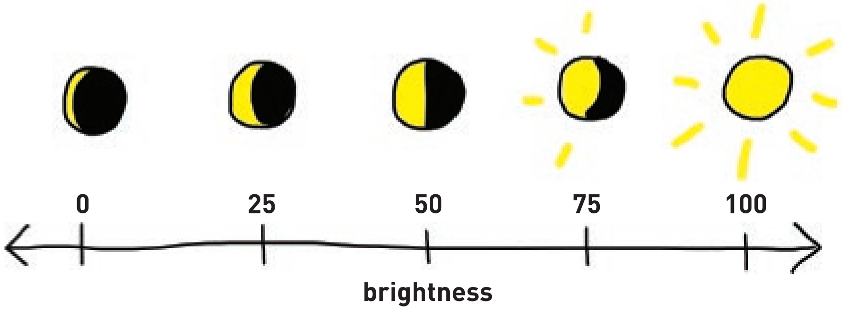
```
setLevel(0, 25);
```

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.



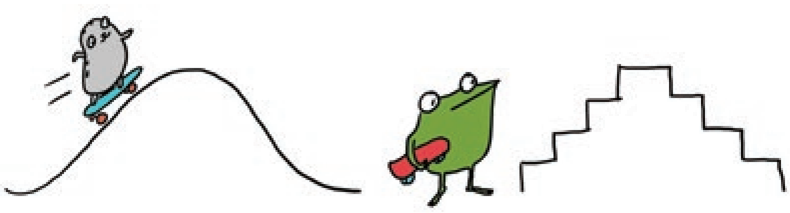
In our example code, we first used `setLevel(LED, 25)` to set the LED pin, or pin 0, to 25% brightness. We then set pin 0 to varying brightness levels, anywhere from 0 for all the way off to 100 for full brightness, in increments of 25.



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?



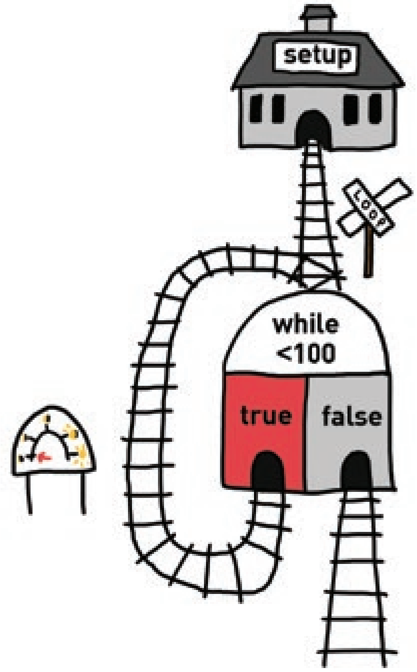
The smallest brightness increment is 1. So to fade more smoothly from off (0) to full brightness (100), we could write `setLevel(LED, 0)`, then `setLevel(LED, 1)`, `setLevel(LED, 2)`, `setLevel(LED, 3)`, and so on all the way to `setLevel(LED, 100)`.

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

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**:

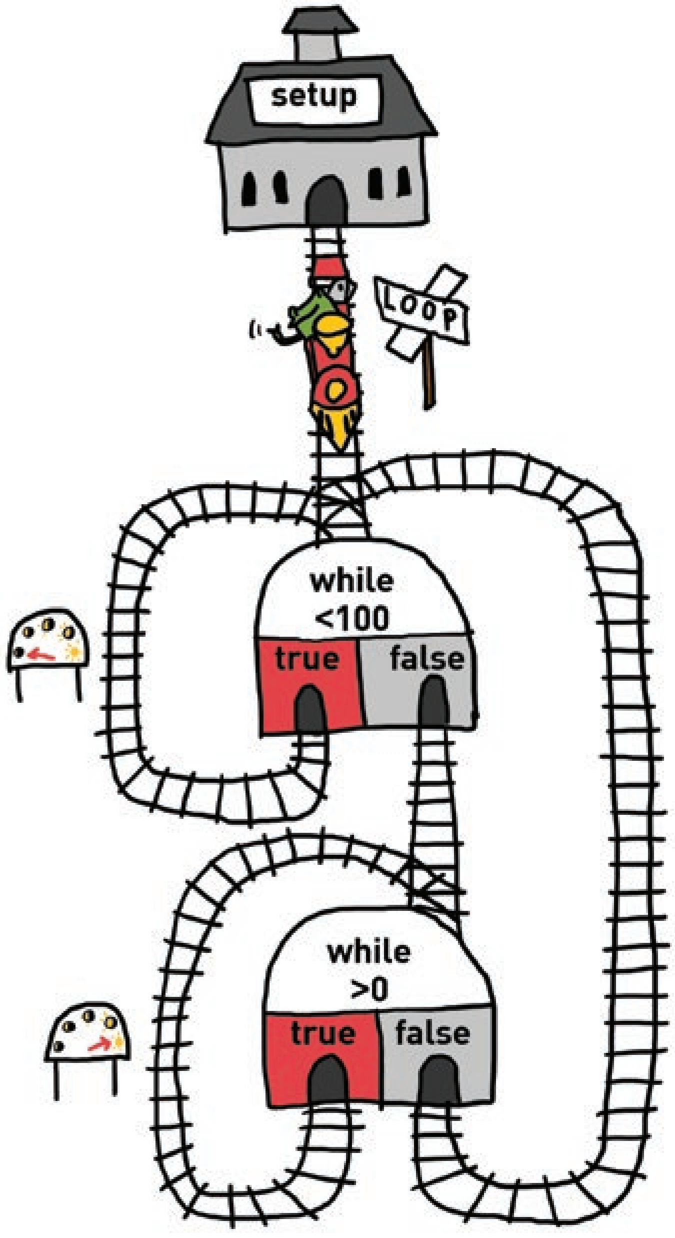
```
while(condition) {
  // Code here runs over and over
  // while condition is TRUE
  Update current condition;
}
```

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**.

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!



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**.

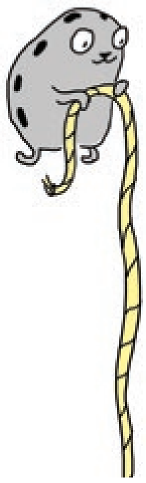
Try loading **Examples → Love to Code Vol 1 → Basic Multithreading** onto a Chibi Chip to see multithreading in action! We'll see each of the six indicator lights above the pins flashing different effects. Here's the code:

```
#include “ChibiOS.h”
#include “SimpleThreads.h”
```

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

Thread 0: Blink pin 0

```
//// 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);
}
```

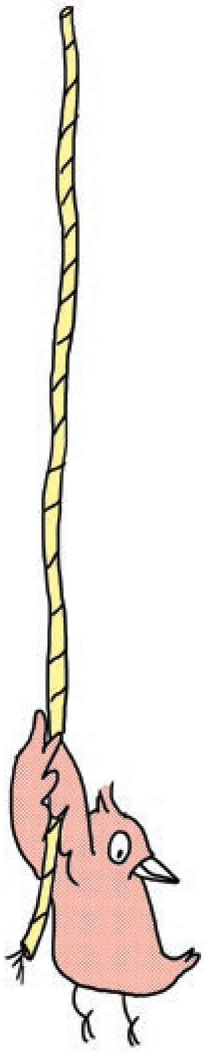


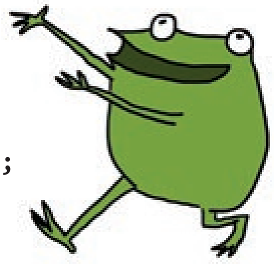
Thread 4: Fade pin 4

```
//// 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;
  }
}
```



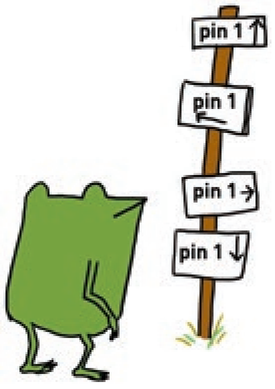


Just as real trains could collide if we put them on the same track at the same time, a Chibi Chip could get confused if two threads fight over a common resource. For example, if `loop0()` and `loop1()` both try to control pin 0, the result would be a confusing mix of commands from `loop0()` and `loop1()`. 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!



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.

GND 0 1 2 3 4 5 GND +3V

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.

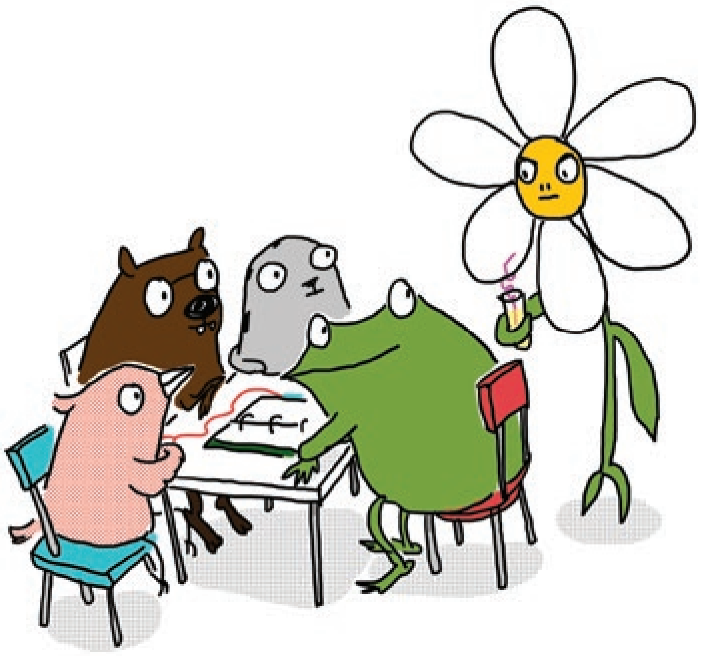
Once finished, lay page **4-20** over page **4-18**, and page **4-21** over page **4-23**. The lights will shine through the parade!

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.



“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.

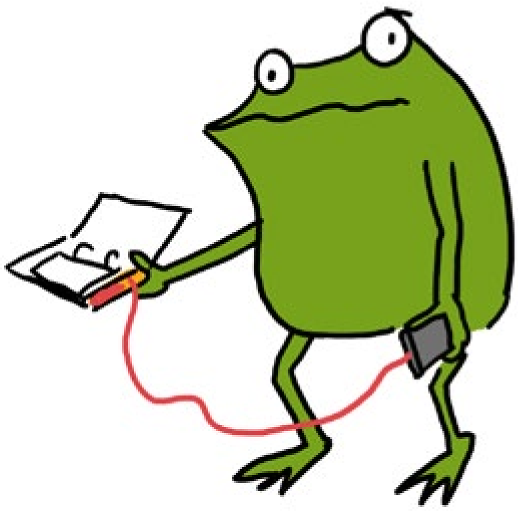
Fern thought for a minute. Maybe George could teach her something new she didn’t already know. “Hey George,” she said, “can you show us?”

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**.

Debugging



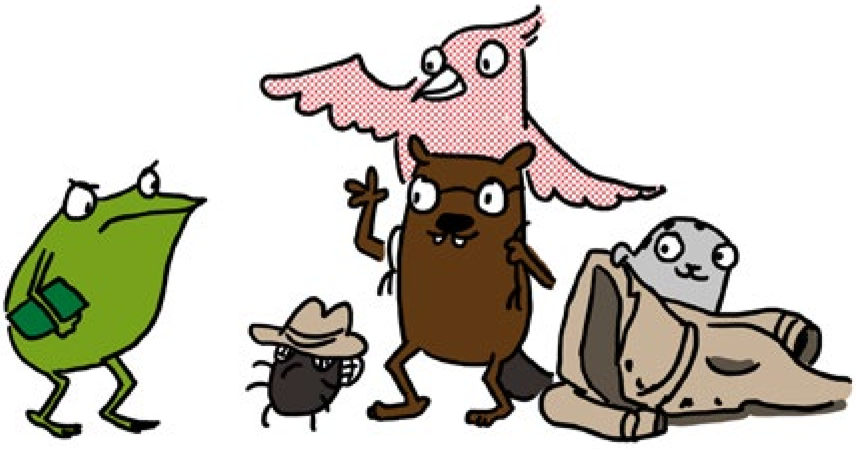
Stuff isn't working? No matter! That's what debugging is for! **Debugging** means looking closely at our project, finding the problems — also known as **bugs** — and then fixing them so that our project works as expected.

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

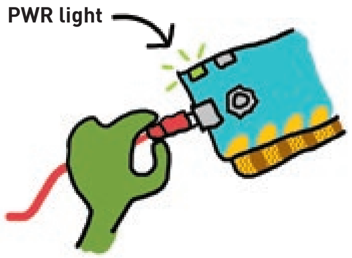
When a project doesn't work, it can seem overwhelming. The problem could be anywhere!

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!

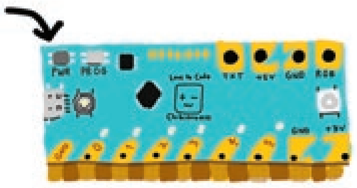


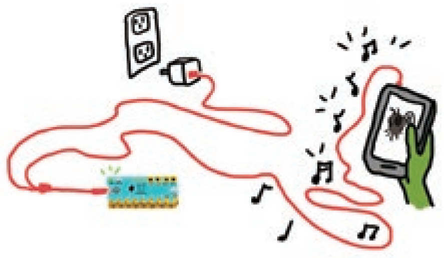
POWER

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.

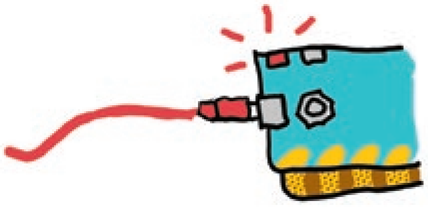


1. Is the PWR light off? If so, it means the Chibi Chip is off because it isn't getting enough power! Try plugging the Chibi Chip into a power supply that you've recently used to charge a phone, like a USB wall plug or computer's USB port. Phones take a lot of power to charge, so if the USB port can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





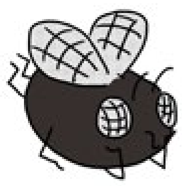
2. Is the PWR light red instead of green? That typically means we have a short circuit connecting +3V and GND directly to each other, draining power from the Chibi Chip! If you're powering the Chibi Chip from a computer, an error message may also pop up on your screen about too much power or current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

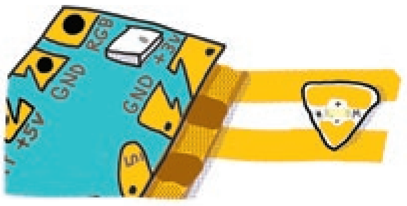
CIRCUIT

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.

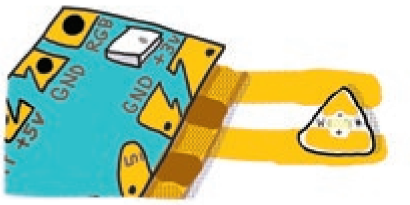


LET'S TAKE A LOOK AT SOME COMMON CIRCUIT BUGS!

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.

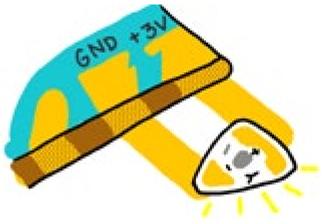


YAY!

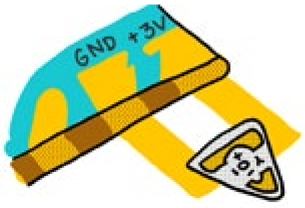


NOPE.

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!



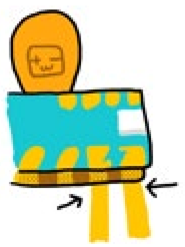
YAY!

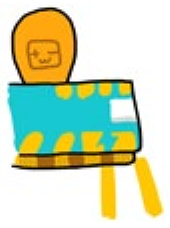


NOPE.

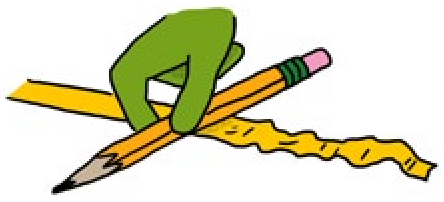
CIRCUIT (CONT'D)

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.





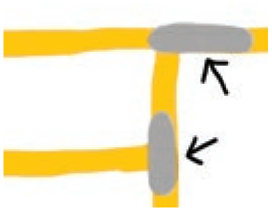
7. Is the copper tape really bumpy or wrinkly? If so, sometimes bumps and wrinkles can prevent solid connections to your LEDs or Chibi Chip. If this is the case, try smoothing out the tape by rolling over it with the flat side of a pen or pencil.

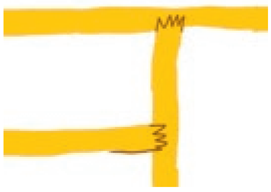


8. Is the copper tape or LED sticker not sticky anymore? Make sure your hands are clean and dry before working with the copper tape and stickers. If the stickers or tape get dirty, they may lose their tack, causing weak connections in the circuit. If this happens, try patching weak connections with conductive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

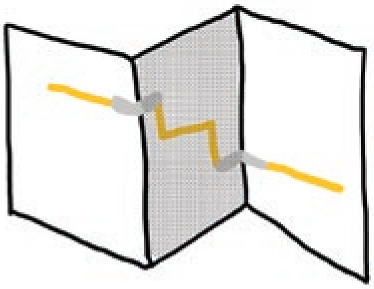
CIRCUIT (CONT'D)

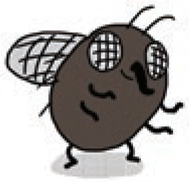
11. Are there branches in your circuit, or do you need to extend your copper tape with another piece? Make sure to use a conductive fabric patch to connect multiple pieces of copper tape. Just sticking two pieces of copper tape on top of each other will not create a strong or reliable electrical connection.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


YAY!


NOPE.

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.

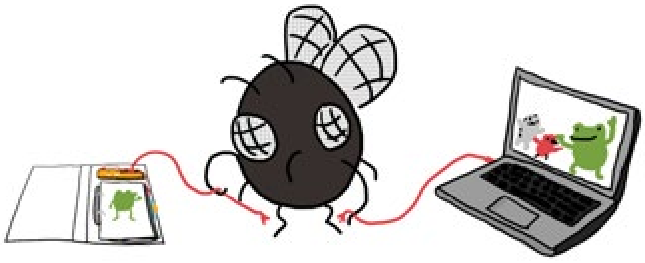




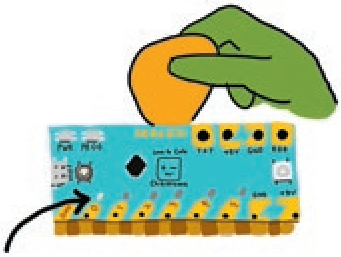
FABRIC PATCHES WON'T DEVELOP CRACKS EVEN WITH REPEATED FOLDING, SO THEY'RE GREAT FOR REINFORCING MOVING OR BENDING PARTS OF A CIRCUIT!

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!

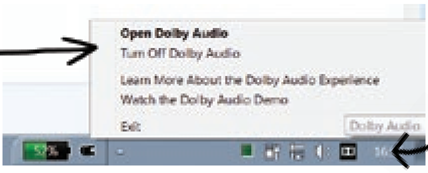


If the Basic Blink didn't upload, let's check out some possible reasons why:

1. Is the volume turned up all the way up? Is the sound acccidentally 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!

UPLOAD (CONT'D)

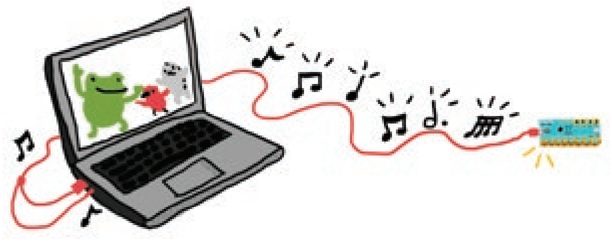
4. Is the audio being distorted? Some laptops automatically apply audio "enhancements" (such as Dolby Audio or bass boost). These enhancements will distort sound in a way that the Chibi Chip may not be able to understand. If you have a Windows computer, particularly those made by Lenovo, try following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


Open Dolby Audio
Turn Off Dolby Audio
Learn More About the Dolby Audio Experience
Watch the Dolby Audio Demo
Exit
Dolby Audio
57%

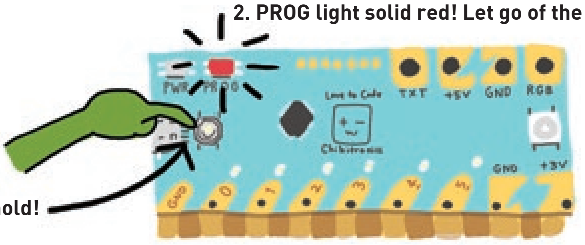
2. Select "Turn off Dolby Audio" (you can turn it on again using the same menu item)

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!

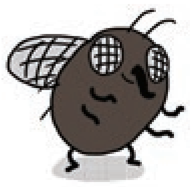


6. Is the Chibi Chip in program mode? Before clicking upload, make sure to press and hold the programming (PROG) button on the Chibi Chip until the PROG LED blinks and stays red. Otherwise the Chibi Chip won't know to listen for new code.



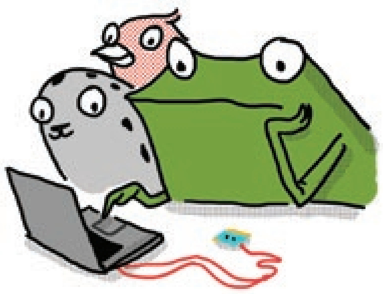
CODE

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!

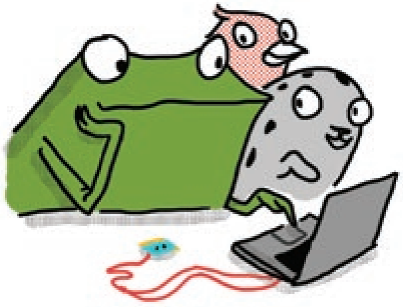


DEBUGGING CODE CAN OFTEN BE A LONG AND FRUSTRATING PROCESS, BUT REST ASSURED, IT'S REALLY SATISFYING WHEN YOU FINALLY FIGURE OUT WHAT'S WRONG AND GET YOUR PROJECTS WORKING!

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 wont 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!

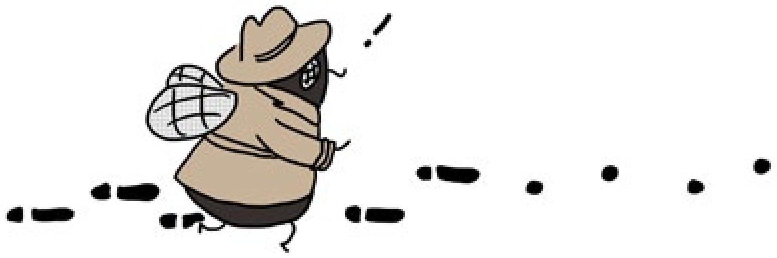
CODE (CONT'D)

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**.

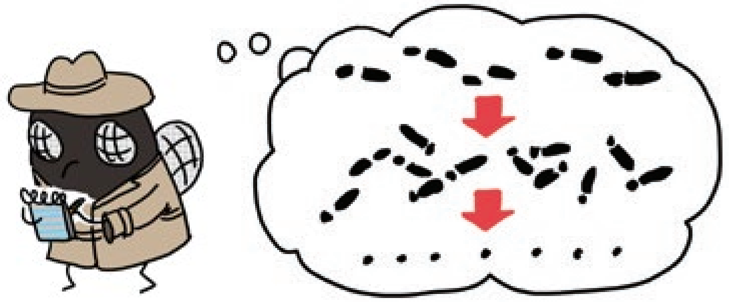
Logic errors can be challenging to spot and fix because we have to figure out our error based on the unexpected behavior of our circuit.

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:



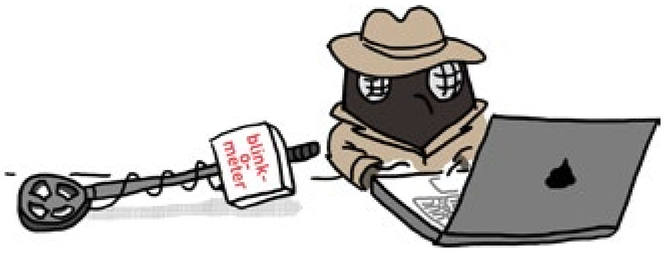
1. Pretend to be the Chibi Chip, and trace through the code line by line. Tracing through a program slowly can help catch many bugs. For example: "turn light on", "wait 1 second", "turn light off", "loop ends, repeat", "turn light on" - Aha, I was missing a delay after the "light off!", so the light 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!

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!

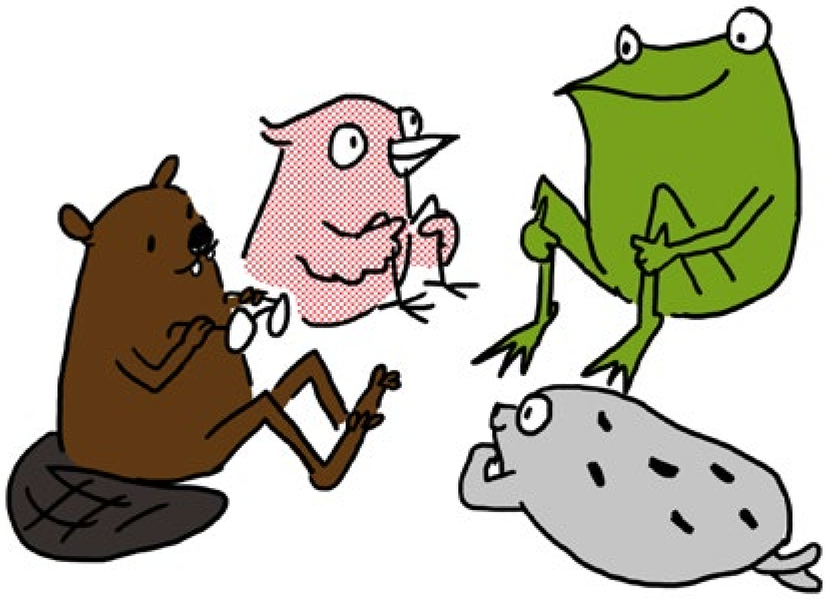
CODE (CONT'D)

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!

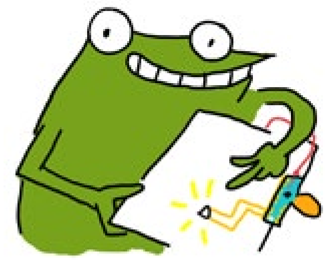
Conclusion

"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:

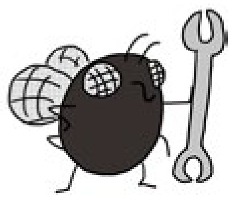


Turn on an LED light using our Chibi Chip

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!

Join the party! Go to **chibitronics.com/projects** to see community projects and to share your own!

Happy making and see you again soon!

9 789811 146886 >