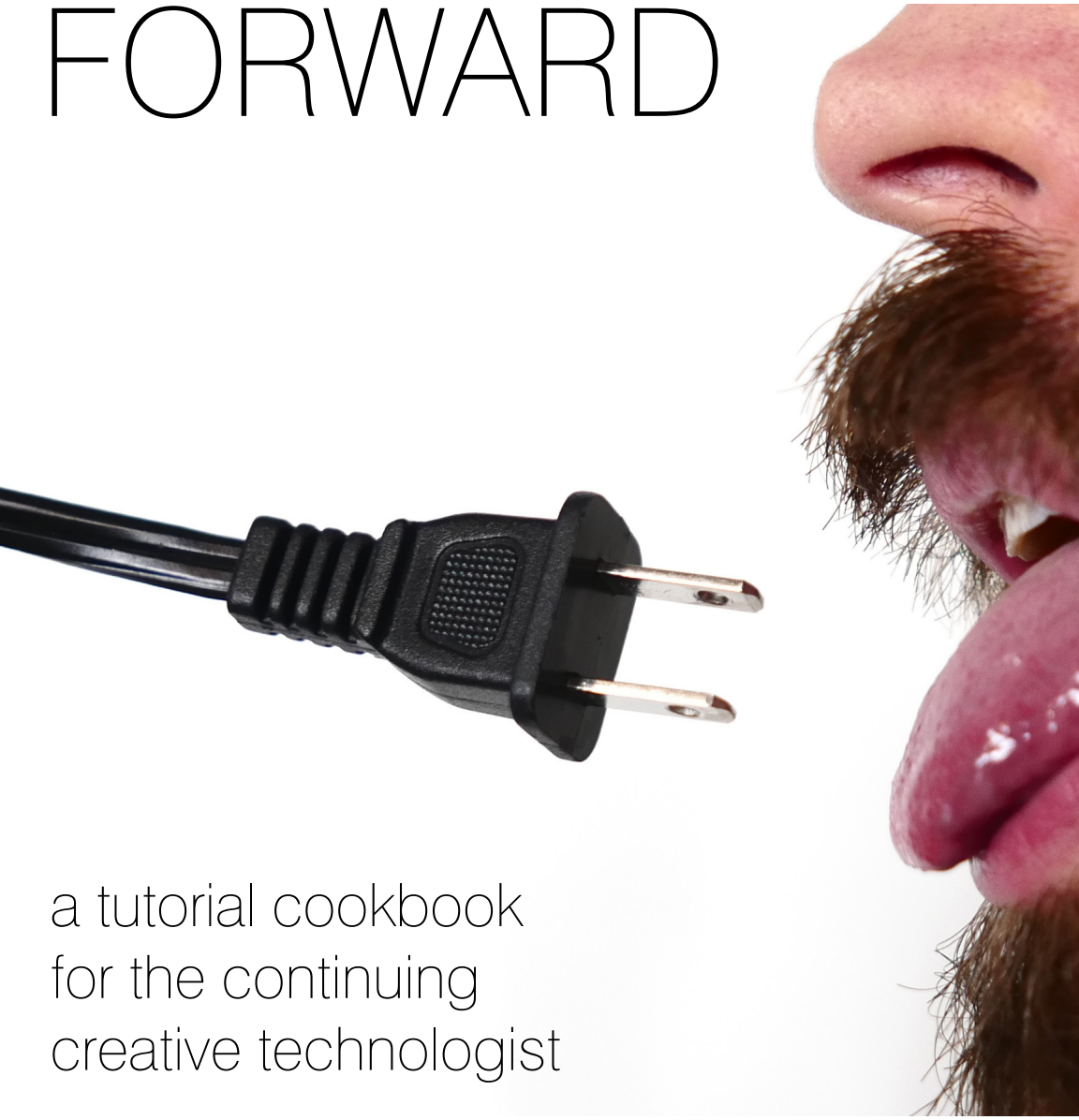
GOING FORWARD



a tutorial cookbook
for the continuing
creative technologist

by dominic barrett

What have you been working on?

This old question. Sure, you need someone to ask. Keeps you accountable. But you're glad it's coming from someone you know. A friend. You can be a bit more honest with them, or opt not to answer. They wouldn't be mad. They would understand. They just genuinely want to know what you've been up to. Certainly not the same feeling when strangers ask. Those people at the party who start by asking, "So what do you do?"

Lately you've started sidestepping that question entirely. "What do I do? Eh. Just an office job." But the conversationinevitably bends back. "Then what do you do outside of work?" "What are you interested in?" "What have you been working on?"

They're trying to be nice, start a conversation. And get to know you, too. It's not their fault the line of inquiry gives you mild anxiety. And really, what have you been working on? You want to be the kind of person who can answer that question. "Oh, a little of this, little of that. There is a project I work on sometimes. Been teaching myself to code."

But with your friends, you can be a bit more real. Vent about your frustrations, indulge your small victories. Last time one of them checked in, you were describing some problems you'd been having with an array in Javascript.

"To be honest, I put the coding aside for a little while. Was having some trouble with it, haven't really been able to make much of anything."

"Sure, sure. Yeah it's hard. But you did a good job with that intro book."

"Kinda. I mean any of the cool stuff I made took a lot from the example code. Yeah I did some fun 'bouncing ball' things. But I don't know trigonometry. Do you know trigonometry?"

"No."

"But you've been coding for a while. You're good with all that stuff."

"Yeah but-"

"I just don't think I'm that kind of person, ya know? I never was that great in math, and I certainly didn't 'like' it. That's for sure. And I didn't get to start coding as early as you did."

You could go on, but you don't. Yes, you finished the intro programming stuff. And actually made some fun little things. But it's light years away from anything sophisticated. Far away from where you would want to be as a "programmer artist" or whatever this weird profession is supposed to be. In between where you are now and where you would want to be is an endless series of hurdles. And you still have go back to the intro material to remind yourself of things you forget constantly. If you were going to be any good at this, wouldn't you have gotten past needing to use the intro stuff by now?

"But you have all these crazy project ideas that you want to make! That installation piece you were talking about, the weird musical instrument you wanted to build…"

"Maybe that just makes me the ideas person."

"Maybe. Maybe." There is a moment of silence. Your friend is thinking, torn between moving onto a new conversation or continuing the thread. But what else is there to say?

"I have this book," they continue, startling you by breaking the silence. Oh great. Another book. Either a whole other beginners book that puts me back at square one, or- "...and it's not some crazy advanced openGL math book."

"Ok, then what is it?"

"It's like a 'next step' book. Doesn't re-teach you what a variable is or anything like that. But there's a bit of intro-level review, like what an object is. And even then it's not just what an object is, but also how to

use it artistically. Actually, there are a bunch of artistic exercises in the book. I didn't really need the tech review, but reading it gave me some good project ideas."

"Hmm… ok… and it's Javascript?"

"p5js. Just like your intro book. But it does make some switches to other languages…"

"Haha yeah like I'm ready for more languages. I don't even know Javascript yet."

"It is just simple stuff, though. And really it's more about thinking about how you switch to a new language, and why… and didn't you say you wanted to learn more about Max MSP?"

"Yeah…"

"...and there is some Arduino stuff. You were always wondering what to do next with that thing, right?"

You nod. Now it is your moment of silence. More reading, more learning. Does it ever stop? But the art exercises… with code… that is the kind of stuff you are interested in...

"...and I have the pdf. I'll just send it over to you."

Your friends know you better than you know yourself, sometimes.

"Oh, yeah? Sure, then send it over why not. What's it called?"

"It's called 'Going Forward'... and… something something I can't remember. But the subtitle specifically says 'for Creative Technologists'. It's not just about programming; it's for people who want to make art with programming."

"Cool. Thanks, yeah send it over I'll give it a shot."

You get the link a few days later:

"Going Forward v1 Final - Copy.pdf" has been shared with you!

It sits in your inbox for a few days. Busy week at work. Sitting in front of a screen all day doesn't exactly motivate you to sit in front of a screen in your free time. Though, that doesn't stop you from watching a couple episodes of your favorite Netflix shows when you get home. You're tired. Bad fast food for dinner the first few days of the week, but you got your act together around mid way through. Kale, homemade soup, simple things.

Your sleep becomes increasingly deep, heavy. Multiple nights in a row, you have dreams of cables, plugs, and connectors. "Who plugged this in this way? It's all wrong…"

Out for drinks with friends on Friday. You hear songs playing on the jukebox that you haven't heard since forever. There is laughing, catching up. Romantic advice. A new movie is out, that director hasn't done anything in years. Did you ever see the first one they did? I never knew about any of it until I was in college.

Somehow, you never discuss work. More than usual, you write down random project ideas in your smartphone's notes app. Between pints, you brandish the device with a deft hand. The brightness is all the way down. Sure, people notice, but no one is distracted. Ideas are jotted down. "A roomba, but paints?" "Synanthrope: cell phones" "Projectors on white shoes, strapped to shins"

Hours of wonderful conversation, and you still somehow come home earlier than expected. Sleep comes easily, again, but this night there are no dreams. You wake up Saturday morning, refreshed. Coffee. You have time to sit with it. It's aroma spreads through your apartment. The sun is shining.

You feel good. Motivated, but not sure towards what. Inspired, but not in regards to anything in particular. Without justification, you feel erudite, holding your coffee and looking out the window. You use the feeling while it's still around.

Grabbing your laptop, you sip the last bit of your coffee while you download Going Forward v1 Final Copy.pdf.

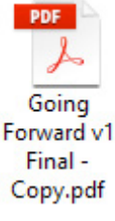
Return to the pot of coffee, it's still steaming. Half coffee, then some cream, and then pour in more coffee again. The cream blooms, mushrooms and then dissipates against the stream.

"Going Forward v1 Final - Copy.pdf has finished downloading"

Your downloads folder could use a good cleaning. But no, that doesn't tempt you like it might on other days. "Later," you tell yourself.

You double click the pdf.

GOING FORWARD



Hi!

Hi! You've chosen to do this tutorial because you are either interested in upping your creative coding skills, need a little bit of review, or want some fresh coding ideas to get the creative juices flowing. Maybe all three!

Today we're going to be working in p5js, with the online editor that can be found here: http://alpha.editor.p5js.org/

These concepts can be brought to other coding languages and environments. But for the sake of quickness and ease of access, we'll stick with p5js' online editor for now.

Lets talk about randomness

One way to spice up your creative coding routine is to utilize randomness. You have most likely gone over randomness if you've done an intro class on p5js. To review, lets run the following sketch in the online editor:

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  console.log(random(10));
}
```

Casting

What do you see in the console log below? If you are like me, it probably wasn't what you expected. Numbers like 4.17533182786781, 3.89760047946547, 9.2887500600597, and so on. We do a basic review of what we already know, and already we are getting curve balls!? What gives, p5js?

p5js is giving us floats, that's what gives. You know this one: a float is a number that has a decimal point in it. But if we

wanted to make a game that guessed your lucky number, we probably wouldn't want those crazy fractions. Unless your lucky number happens to be 9.28875200600597.

For the rest of us normal, boring people, we want numbers without decimal points. We can force p5js to convert this number into a whole number, or integer. This is called "casting", as in, "You cast the incoming number as an integer." This terminology is not unique to p5js or Javascript, and as a general programming term.

So to "cast as an int" (all the cool kids say int instead of integer), you would wrap our random call around the int() function like so:

```
console.log(int(random(10)));
```

Now lets move that line up into the setup function, which only runs once. Before you run the code, you think of your lucky number between 0 to 10. Your press the run button and… you didn't get your lucky number!

Unless you did. But most of you probably didn't, and maybe for a brief second you thought I was reading your mind. Nope. Just playing the numbers. Also, if your lucky number was 10, you definitely didn't get it. That's because when we cast a float as an int, we are simply discarding everything after the decimal point. No rounding up or anything like that. Try putting the following into your sketch:

```
console.log(int(9.9));
```

You'll see "9" come up in your console. The .9 is simply discarded. So, to compensate for this, we could change our line to:

```
console.log(int(random(11)));
```

We'll see some 10s come through on the console. But also some zeros. And if your lucky number is zero, I commend your edgy gothness. However if we were framing our question as, "Pick a number 1 through 10", we would want to make a range. In this case, you add an extra argument to the random function.

```
console.log(int(random(1,11)));
```

This picks a random number starting at 1, leading all the way up to 11 as the limit that never gets touched. So, you could get 10.999999999, but because you cast it as an int it still turns into 10.

But this isn't good enough for you. You don't sit through life waiting for your lucky number to come through. You are a programmer. You make your own luck. But how, you wonder?

Random Arrays

Try this:

```
function setup() {
  createCanvas(400, 400);
  oneThroughTen = [1,2,3,4,5,6,7,8,9,10];
}

function draw() {
  background(220);
  console.log(random(oneThroughTen));
}
```

No floats, no int casting, no 11's. Otherwise, similar functionality of the last number picker. But it looks different to you. What's going on?

The p5js function random() doesn't just pick random numbers. Nor is it limited to picking random numbers within a range. It also picks randomly from an array of numbers. This means that you can weight the scales in your favor. Try this:

```
oneThroughTen = [1,2,3,4,5,6,7,7,7,7,7,7,7,7,7,8,9,10];
```

There are nine other numbers besides 7, so if you put in nine 7's you should be seeing a 7 show up around 50% of the time.

"How can I use this besides create an extremely unethical gambling startup?" you wonder. Well, today is your lucky day. And if it isn't, just throw a few more 7s into that array. Because random() doesn't just pick randomly from an array of numbers, but can pick randomly from an array of anything.

Take colors for instance. Let's start our program like this:

```
//color choices
var color1;
var color2;
var color3;
var color4;

//an array to hold all of the colors
var colors;

function setup() {
  createCanvas(400, 400);

  color1 = color(255,255,255);
```



```
  color2 = color(0,255,255);
  color3 = color(255,0,255);
  color4 = color(255,255,0);

  equalColors = [color1,color2,color3,color4];

  mostlyColor2 = [color1,color2,color2,color2,
                  color2,color2,color2,color2,
                  color2,color2,color3,color4];

  mostlyColor3 = [color1,color2,color3,color3,
                  color3,color3,color3,color3,
                  color3,color3,color3,color4];

  mostlyColor4 = [color1,color2,color3,color4,
                  color4,color4,color4,color4,
                  color4,color4,color4,color4];

  colors = equalColors;
}
```

You have four colors: white, cyan, magenta, and yellow. You've made four arrays with randomness in mind: even odds, mostly cyan, mostly magenta, and mostly yellow. Now you can add the second half of the program:

```
function draw() {
  background(75);

  for(i=0;i<4000;i++){
        fill(random(colors));
        noStroke();
        rect(random(width),random(height),10,10);
  }
}
```

```
function keyPressed(){
        if(key==1){
          console.log(“keypressed 1”);
          colors = equalColors;
        }
        if(key==2){
          console.log(“keypressed 2”);
          colors = mostlyColor2;
        }
        if(key==3){
          console.log(“keypressed 3”);
          colors = mostlyColor3;
        }
        if(key==4){
          console.log(“keypressed 4”);
          colors = mostlyColor4;
        }
      }
```

Here we are drawing 4,000 squares on the screen every frame. The squares are 10 pixels in size and placed in random locations within the bounds of the width and height of the sketch. Press the 1,2,3, and 4 buttons on your keyboard to cycle through the different modes of probability.

Random Anything!

And it doesn’t stop at colors. You can have randomized text as well, because you can put strings into an array just as easily.

You’ve done a good job following along so far. So I’m going to give you an example sketch to play around with instead of talking you through everything again, except with text. Needless to say, you’ll see some of the same strategies being employed. Take this time to change the variables, colors and text to something that you like and has your artistic vision.

```
//color choices
var color1;
var color2;
var color3;
var color4;

//an array to hold all of the colors
var colors;

var displayText;

var textChoices = [‘good ‘,’smart ‘,”really hoping this
will be useful.”+
“Or, not useful, but maybe just something that turns
out to be inspiring.”+
“Something that resonates, ya know?”+
“I keep having this compulsion to make art with
technology.”+
“But how great has it been, *really*?”+
“Maybe this will be it, or the start of the it, that gets
```

```
me to that level.”+
“Or, place.”+
“I should say place.”+
“Level implies something else.”+
“\n”+
“Just like the word useful implies something else.”+
“Hierarchy, merit, productivity.”+
“Thats not why I wanted to make things with
 computers.”+
“But… maybe if I had focused on the more practical
things, I’d be a better coder.”+
“And then I wouldn’t have to look to books to
 help me.”+
“Why didn’t I pay more attention in math class?”+
“Why didn’t I have parents that pushed me too hard
to be good at math?”+
“\n”+
“But then some of those math people say the same
thing to me about knowing other things.”+
“Now that I think of it, most thoughtful people I know
will at least occasionally wonder things like that.”+
“Grass is greener on the other side, I suppose.”+
“Its just a shame to think that my friends would
think that their talents aren’t “,’good ‘,’smart ‘,’good
‘,’smart ‘,’good ‘,’smart ‘,’\n\n’];

function setup() {
  createCanvas(400, 400);

  color1 = color(255,255,255);
  color2 = color(0,255,255);
  color3 = color(255,0,255);
  color4 = color(255,255,0);
  equalColors = [color1,color2,color3,color4];
  mostlyColor2 = [color1,color2,color2,color2,
                  color2,color2,color2,color2,
                  color2,color2,color3,color4];

  mostlyColor3 = [color1,color2,color3,color3,
                  color3,color3,color3,color3,
                  color3,color3,color3,color4];

  mostlyColor4 = [color1,color2,color3,color4,
                  color4,color4,color4,color4,
                  color4,color4,color4,color4];

  colors = equalColors;

  displayText = “I am “ + random(textChoices) +
“enough”;
}

function draw() {
  background(75);

  for(i=0;i<4000;i++){
     //fill(color1);
     fill(random(colors));
    noStroke();
     rect(random(width),random(height),10,10);
      }

  textSize(100);
  textFont(‘Helvetica’);
  stroke(255);
  strokeWeight(5);

text(displayText, 25, 25, width, height);
}
```

```
function keyPressed(){
  if(key==1){
    console.log(“keypressed 1”);
    colors = equalColors;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
  if(key==2){
    console.log(“keypressed 2”);
    colors = mostlyColor2;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
  if(key==3){
    console.log(“keypressed 3”);
    colors = mostlyColor3;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
  if(key==4){
    console.log(“keypressed 4”);
    colors = mostlyColor4;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
}
```

Actually *Using* Randomness

After showing people the previous chapter and getting some feedback, people asked me what ways we could use this approach to randomness in their creative coding practices. This isn't just a fair question for me to answer, but for many tutorials and example code. When you are already working on a project and just need an answer to a specific technical question, you will take what you need from the lesson and continue on your merry way. However, when you are learning for a general mastery, it may be hard to imagine how or why these things may be put into practice.

Which is actually a bit of an issue when you are learning, in my opinion. I can tell you that you can use the random function on more than just numbers, and you may say, "... ok." I could tell you to copy and paste what I have, and then challenge you to replace the numbers with strings. You might also say, "...ok…" and then proceed to haphazardly throw in some gibberish, perhaps a swear word or two. You stop the sketch, change some of the text to double check your competency, and when the program runs as you expect you will proceed to the next coding exercise.

I've been there. My problem was, when it actually came time to be creative with code… the blank white screen made me realize I didn't know how to use what I knew. Figuring out ways to actually use the examples in a man-

ner that was interesting to me, resulted in me using those concepts more. Which not only helped me make interesting programs (ok, ok, interesting for me), but helped me solidify those concepts in my head.

This is a guide for creative coders.

But being creative can be hard. Especially when brought into the technical realm. Everything seems possible, which is too much to think about. But a beginner-to-mid level skill level slows you down and limits the scope of your ambition, which is discouraging. These are not fertile conditions for the creative mind. Which could be just enough to drive people away from creative coding altogether.

And what kind of programming guide would this be if you don't come out the other side ready to make awesome programs? No kind of guide at all, I say.

This means that I am going to try and be like that cool history teacher in college who told you, "You won't need to memorize dates!". I'm going to eschew the quizzes, challenges and pure technical focus that other programming guides have. Don't get me wrong, they have their place. But here we are going to try and pair technical practice and information with creative practice and information. This is in the hope to make you a better creative coder, not just programmer.

What Next?

So how can we take the concepts and code from part one and kick things up a notch? Well, since I'm on such a kick about randomness, maybe randomness will help me with some inspiration.

From Brian Eno's online shop:
https://www.enoshop.co.uk/product/oblique-strategies.html

What do I do? What next? What does the future hold? Randomness has helped answer the big questions throughout the history of humanity. Runes, I-Ching, Tarot cards. A certain kind of artistic divination was developed by Brian Eno in 1975, with a card set called "Oblique Strategies". Oblique Strategies is a set of cards with instructions for people who are running up against a creative block or looking for new inspiration.

How do I use it? Example: as a complete amature, non-professional writer, I am currently experiencing writer's block. Totally at a loss as to what to type next. What to do? I might pull a card from an Oblique Strategies deck and use it as a creative prompt or design constraint. Sometimes they are more specific, other times very ab-

stract. The card I pulled was:

“Try faking it!”

Hmmm. Not sure why I would want to fake anything. Being the commensurate professional writer that I am, I shouldn’t have to. With all these ideas brimming in my head, just waiting to burst out on the page?

Maybe you can try a card and see what comes up. You can find my deck here:

https://alpha.editor.p5js.org/dbarrett/sketches/ryXGukwFz

First, a Little Story

I had found the text file for Oblique Strategies online. I put it into a txt file, knowing that p5js would be able to parse it. But I don’t know how it works off the top of my head, so I went to:

https://p5js.org/reference/

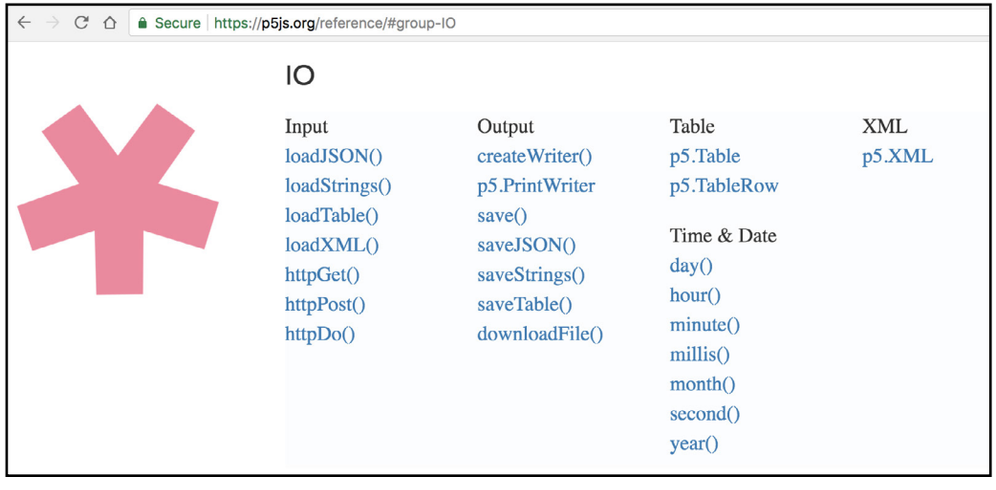
If I’m coding in an environment or language, I always have the reference handy. First, I clicked on the “IO” section link. Why did I think IO would give me the information I needed, and not something like the “Data” section? “IO” stands for input and output. Yes, a text file contains data. But more importantly, from the perspective of documentation, it is an external file that needs to be loaded. Hence “Input”.

Then inside of “IO”, I clicked on “loadStrings()”. Why did I think that had what I needed? Because I’m dealing with a txt file. This is typically the most basic way that we are going to store characters on a computer. At the end of the day, things like JSON, XML, CSV, are just a specific way of organizing text. P5js, and plenty of other languages and environments will give you built in functionality to parse JSON, XML, and CSV.

But the basic file type for text is txt, and the basic data structure for text is a string. I have enough experience to expect that the function “loadStrings()” is what I’ll be needing.

You most likely know most of these facts by now if you have completed an intro class or book. Well, that is to say, you’ve been exposed to them. Perhaps this jogged your

memory. But the more you are actually using these things in your practice, the more you'll be digging into your programming reference documentation. And mental linkages like what I've outlined above will start to become second nature. You will build this instinct as well, over time.

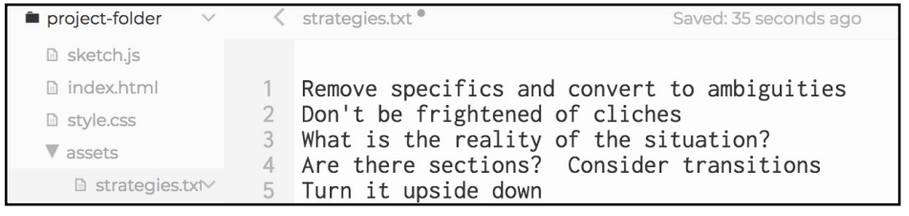


Reference. Use it!

The Plot Thickens

I'm currently using the alpha version of the online p5js editor. I uploaded the file into my sketch directory. However, when I clicked on the file, I didn't see any of the text inside of it. All blank. What gives?

To be honest, I don't know. I had seen some behavior like this before, and knew it might be a while before the glitch was fixed. I simply opened up a copy of the file on my machine, and pasted the contents into the blank text file inside of the online editor.



Then I output the entire files contents into a console log to make sure it worked, and up came… HTML? My txt file didn't have any HTML in it. Something had gone wrong. I saw "amazon" and "aws" tucked away inside of the HTML. This must be where the online editor is hosting the txt file, and things had gone screwy on the backend. Another glitch.

So what to do? I looked at the example code from the loadStrings() reference entry. All of the lines loading the txt files would put them in a folder called "assets" first; 'assets/test.txt'. Perhaps this assets folder was some kind of requirement. So I created an assets folder, created a new txt file inside of it and pasted my content into it.

I used the loadStrings function, and it worked like a charm. Then in an effort to document this bug in order to show it to you, I attempted to recreate it. I made a text file in the root directory again, put content in it, and attempted to read from it. And succeeded. No HTML, no error.

I have absolutely no clue why this works now.

WOW what a boring and tedious story!

I couldn't agree more! But it illustrates an important point. Sometimes when we're coding or building circuits or making anything amazing with technology, you hit hurdles like these before you even get started. You updated your development environment and now the code that worked yesterday is broken. Re-installing a library doesn't work for some mysterious reason. The microphone input works on Android browsers, but not iOS.

All these tasks are things that have to get done before you can start "working", even though this "not work" is costing you plenty of time. Sometimes this is called "yak shaving"; all the little dumb tasks and problems that have to get solved in order to solve your bigger problem. You can and will be able to solve these problems. They will happen again and again however, no matter how experienced a programmer you become. Perhaps even when you are writing a programming tutorial, for example.

Take solace knowing that the pros are toughing it out in the trenches, throwing error messages into Google just like the rest of us.

Let Us Consider

Imagine Claude Monet, looking out his window when he is suddenly struck by inspiration. He excitedly opens his drawer of paintbrushes, only to have a screw pop out of the sliding hinge and have the entire drawer crash onto the ground. Of course, the only brush he was actually looking for was the one that broke. Right in half.

He puts on his coat and walks down to the art supply store. "That thing was creaking, should have fixed it before something happened. Or at least been more gentle opening the drawer up. Come on Claude, you're better than this!" He finishes silently scolding himself as he arrives at the door of the store. A "Closed" sign dangles on the other side of the glass.

"Damnit! They close Mondays I always forget… what to do what to do…"

The hardware store isn't too far away. Maybe some heavy duty tape will make do as a temporary fix. He goes into the store, and grabs some tape. Oh! And while he is there, why not get some new screws with fresh threads on them so he can fix the drawer? Up and down the aisle he goes, doubling back again to find the display of screws he missed. But the only box of phillips heads left is a pack of 100. Ugh, come on.

So he finds a store employee and asks if they have any packs of phillips heads with less than a hundred screws. The employee goes back with Claude to the screw section and takes a look at the stock shelves above the customer display.

"We got a 24 pack, but they're flat head screws."

"No phillips heads?"

"No, but the flat heads are the same thread and length. You have a flat head screwdriver?"

Claude sighs and closes his eyes. He thinks. Does he have a flat head screwdriver? He should, but he can't really remember for sure. Phillips head? Totally. No question.

But flat head, flat head, flat head...

“...no… no I don’t think so.”

“Well we got a flat head screwdriver for sale right here, if you want that and the 24 pack.”

“nah... together that’d cost more than the hundred pack of phillips head screws. Is there any way I can break up the hundred pack? Honestly I only really need three or four.”

“Sorry sir, can’t do it. Store policy.”

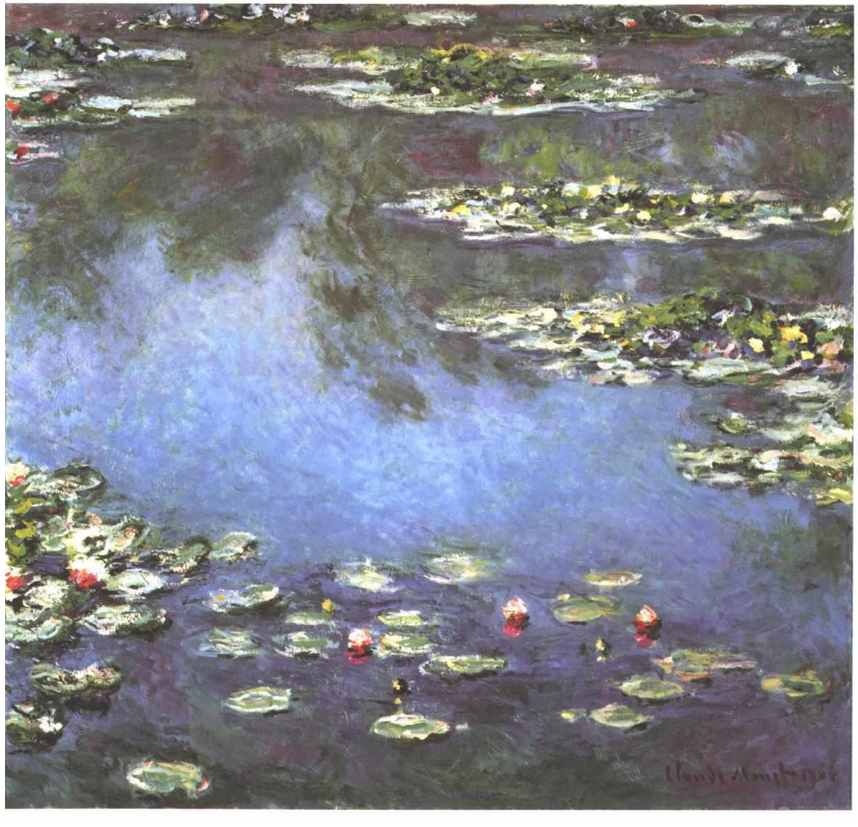
“Not even if I bought like, fifty?”

“No, sorry.”

“It’s fine, had to ask. I guess I’ll take the 100.”

And so, Claude Monet bought a roll of that cloth-type gaffer tape and an absurdly large box of phillips head screws. He walked down the street, past the closed art store, back to his home. The wreckage of the spilled drawer lay on the ground and he got to the task of cleaning the mess. One by one he put each brush back, until he came across a flat head screw driver and he thought, “At least I didn’t buy another one”.

We don’t think about this when we look at Monet’s painting “Water Lilies”, but maybe we should.



Yak shaving.

Alright

We have had our own “Monet’s trip to the hardware store” moment, and can finally get down to the business of starting. It could be easy to be in a sour mood having spent so much time, but perhaps also a feeling of relief. You’ve finally arrived to your medium, ready to move forward. What do you have for us, Oblique Strategies?

I ran the program and came up with: “Spectrum analysis”

Hmmmm, ok.

Brian Eno made Oblique Strategies with music at the front of his mind, so there are going to be some entries that may be more oriented specifically to music and sound. You don’t have to use the first card you draw if it doesn’t make sense to you (I promise, I won’t tell anyone). But this isn’t a half bad prompt. I have an idea.

Use Some Audio!

Alright
We have had our own “Monet’s trip to the hardware store” moment, and can finally get down to the business of starting. It could be easy to be in a sour mood having spent so much time, but perhaps also a feeling of relief. You’ve finally arrived to your medium, ready to move forward. What do you have for us, Oblique Strategies?

Lets analyze the audio spectrum.

Copy and run the following code in the p5js online editor:

```
var mic;

function setup() {
  mic = new p5.AudioIn();

  // start the Audio Input.
  // By default, it does not .connect() (to the computer
speakers)
  mic.start();

  createCanvas(400, 400);
}

function draw() {
  background(220);

  var vol = mic.getLevel();
  console.log(vol);
}
```

This is a basic use of the AudioIn functionality in the p5.sound library. You may need to give your browser permission to access the microphone. If everything is set, you should be seeing some floats scrolling down in your console. This is the measured volume of what your microphone is hearing.

If I’m adding a new feature into a program, I like to have a separate running example like what we have above. I can make sure it works, test out solutions to issues, and start yak shaving again without messing with the pretty code that works. Then, once I know it works (is the mic working? Is it on? Do I even have a microphone??) I will do a transplant into my main project. Today, we’re doing an ear

transplant so our p5js sketch can hear.

Lets flex our muscles a bit, and perhaps establish a bit of a rapport with one another. I'm going to describe how you should perform this code transplant surgery, but not show you the actual code.

"But you said there would be no programming challenges!"

I know, I know. Maybe it is better to think about this as a stretching exercise. You aren't a total beginner, but we just need to get a little flexible and start the blood flowing.

I would like you to take the program from the end of part one (found here) and add this example microphone code into it. How?

> The mic variable should be created outside both functions because it is a global variable.
>
> Assigning the mic variable as a new AudioIn(), and the start() method should go in the setup function.
>
> Get the mic level by assigning it to a new variable called 'vol',
>
> and console logging 'vol' should happen in the draw loop so we can see the changes.

Did it work? Great! Did it not work? It happens. Maybe take a look at the errors you get and try moving things around. Did you copy and paste things that weren't code? Did you not copy and paste the entire code snippet? Did you miss the last } at the end? If you're still having issues, you can use this code:

```
var mic;

//color choices
var color1;
var color2;
var color3;
var color4;

//an array to hold all of the colors
var colors;

var displayText;

var textChoices = ['good ','smart ,"really hoping this
will be useful."+
"Or, not useful, but maybe just something that turns
out to be inspiring."+
"Something that resonates, ya know?"+
"I keep having this compulsion to make art with tech-
nology."+
"But how great has it been, *really*?"+
"Maybe this will be it, or the start of the it, that gets
me to that level."+
"Or, place."+
"I should say place."+
"Level implies something else."+
"\n"+
"Just like the word useful implies something else."+
"Hierarchy, merit, productivity."+
"Thats not why I wanted to make things with comput-
ers."+
"But… maybe if I had focused on the more practical
things, I'd be a better coder."+
```

```
"And then I wouldn't have to look to books to help
me."+
"Why didn't I pay more attention in math class?"+
"Why didn't I have parents that pushed me too hard
to be good at math?"+
"\n"+
"But then some of those math people say the same
thing to me about knowing other things."+
"Now that I think of it, most thoughtful people I know
will at least occasionally wonder things like that."+
"Grass is greener on the other side, I suppose."+
"Its just a shame to think that my friends would think
that their talents aren't ",'good ','smart ','good ','smart
','good ','smart ','\n\n'];

function setup() {
  createCanvas(400, 400);

  mic = new p5.AudioIn();

  // start the Audio Input.
  // By default, it does not .connect() (to the computer
speakers)
  mic.start();

  color1 = color(255,255,255);
  color2 = color(0,255,255);
  color3 = color(255,0,255);
  color4 = color(255,255,0);

  equalColors = [color1,color2,color3,color4];

  mostlyColor2 = [color1,color2,color2,color2,
                  color2,color2,color2,color2,
                  color2,color2,color3,color4];

  mostlyColor3 = [color1,color2,color3,color3,
                  color3,color3,color3,color3,
                  color3,color3,color3,color4];

  mostlyColor4 = [color1,color2,color3,color4,
                  color4,color4,color4,color4,
                  color4,color4,color4,color4];

  colors = equalColors;

  displayText = "I am " + random(textChoices) +
"enough";
}

function draw() {
  background(75);

  var vol = mic.getLevel();
  console.log(vol);

  for(i=0;i<4000;i++){
        //fill(color1);
        fill(random(colors));
    noStroke();
        rect(random(width),random(height),10,10);
  }
  textSize(100);
  textFont('Helvetica');
  stroke(255);
  strokeWeight(5);
        text(displayText, 25, 25, width, height);
```

```
}

function keyPressed(){
  if(key==1){
    console.log(“keypressed 1”);
    colors = equalColors;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
  if(key==2){
    console.log(“keypressed 2”);
    colors = mostlyColor2;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
  if(key==3){
    console.log(“keypressed 3”);
    colors = mostlyColor3;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }
  if(key==4){
    console.log(“keypressed 4”);
    colors = mostlyColor4;
    displayText = “I am “ + random(textChoices) +
“enough”;

  }

}
```


Alright, now we have our random visuals and we’re detecting sound. How can we incorporate these sound values into our sketch?

Before we dive in

This can be a very open ended question. There are tons of other variables that we can replace with our ever changing "vol" variable, resulting in a reactive piece of programming art. My broader instinct is that I want to play, and try out a bunch of different things. Color, size, position, and on and on.

I'm immediately thinking about how I am changing the volume number into a new number that makes sense for any of these potential fields. Easily and quickly. Our sketch size is 400 by 400, or rectangles are 10 by 10, and rgb color values go from 0-255. The volume returned to us is a floating point number between 0 and 1.

It would be easy to simply multiply our volume value by any of these maximum numbers. Half volume could yield a position on the canvas that is somewhere in the middle. A quiet microphone could create a darker background color. But you don't always get input values from 0-1, and sometimes you want to change the way the translation works. What if we wanted our sketch to get brighter as it got quieter? Then instead of 0 input:0 color and 1 input:255 color, it would be 0:255 and 1:0. And what if we're kind of close to what we want, but we want to tweak the details just a little bit? Maybe not too bright, or just a hair bigger, perhaps scoot it to the left a bit more.

In order to facilitate an open ended discovery, I instinctively use the map() function. This takes a number, that numbers range, and then a new range. You have probably used this before. And in other languages, sometimes it can be called something different, like "scale". But it will make this kind of experimentation much quicker.

We have our vol, and we can add a new variable called mappedVol.

```
var vol = mic.getLevel();
var mappedVol = map(vol,0, 1, 0, 255);
console.log("Vol____:" + vol);
console.log("Mapped:" + mappedVol);
```

This could be used for brightness correlating to noise. But if we wanted the inverse? Just flip your mapping

```
var mappedVol = map(vol,0, 1, 255, 0);
```

Playing around with the mic example (you are playing around with it, right? Always take a moment to make funny noises when you are testing noise detection code) you may notice that vol never really seems to hit "1". You could always just change the 255 to something higher. If you wind up going past 255, the color code will still be treated as if it was 255. If its 256, 257, 5000, whatever. You're good.

Using map() lets us quickly test these linkages we want to make between our input and what that input effects. I don't want you to technically know map() and use it, but I want you to think map(). I want you to feel map() and look through the world with map() tinted glasses.

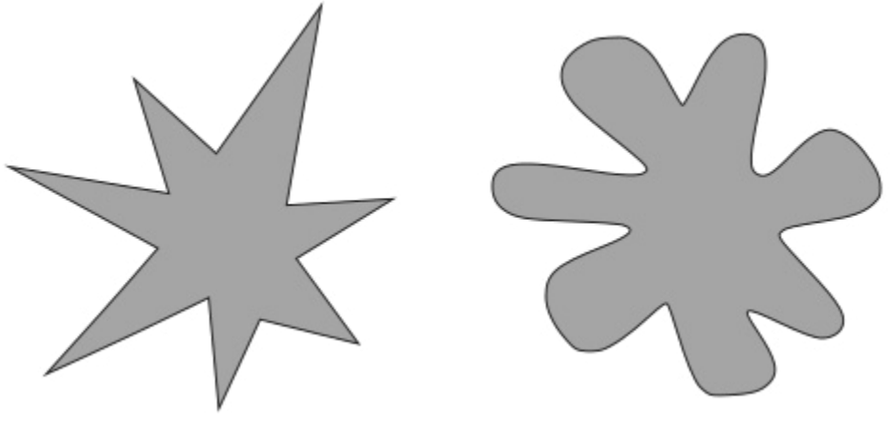
Synesthesia and other strategies

I will always encourage plugging one input into another output just for curiosity's sake. Why not? Who knows what strange system will result from it? If you get something that works, roll with it.

But if you are failing to come up with good random linkages, what else can you draw upon? Synesthesia is a condition experienced by some people where two senses can be linked via specific experiences. A common example is experiencing some musical notes as specific colors. Other senses and topics can be tied; numbers, space, months or days of the week. People experience many variations of these, though the specific pairing can change from person to person.

This general concept can be a jumping off point for coming up with interactive concepts. You may have thinking this way already. But in the case of people who experience synesthesia, each relation between senses is different from one person to the other. A subjective experience amongst specific people.

Let's also consider the phenomenon that there is a human tendency to naturally associate certain things with each other. This can be seen in the Bouba/Kiki effect.



https://en.wikipedia.org/wiki/Bouba/kiki_effect

Question: these two forms each have a name. One is "Bouba" and the other is "Kiki". Which shape has which name?

Vast majorities of people agree that the rounder shape is Bouba and the sharper shape is Kiki. Across multiple cultures and languages, from children as young as 2 ½ years old. Experiments in 1929, 1947 and 2001 replicated the phenomenon.

Even if you didn't pick the same names, these are points of leverage when we think about our audience. We could play into people's expectations, or perhaps subvert them, in order to make a compelling experience. A performance, an interactive sculpture, a UX experiment, or interface.

So I encourage you to think in this manner if something doesn't leap to mind immediately. Taking a subjective linkage that is interesting, or investigation a widely held association. Sometimes these type of linkages will even be your first thought, a sort of obvious choice that your mind immediately leaps to. I was thinking about noise volume and size.

```
var vol = mic.getLevel();
var h = map(vol, 0, 1, 0, 20);

console.log(vol + " " + h);

for(i=0;i<1000;i++){
   //fill(color1);
   fill(random(colors));
   noStroke();
   rect(random(width),random(height),10,h+1);
}
```

What if the “static” blocks we drew every frame were very thin, and only grew to their full size when there was a loud noise? Conceptually, this may play into a straight, “one to one” style mapping. A louder, thing is larger. Feels right, doesn’t it? But there also may be another way to consider the metaphor; a visual noise that grows when there is more sonic noise.

Side note: TV static, noise, and the case against Kiki and Bouba

A bit of a diversion, but bear with me. My instincts about “noise” are perhaps generationally biased. I grew up with analog TVs, broadcasting over UHF/VHF frequencies. If you tuned into a station or frequency that wasn’t broadcasting, you would see the “snow” of visual white noise and hear an auditory white noise.

https://en.wikipedia.org/wiki/Noise_(video)

Even if you had cable instead of over the air antennas, perhaps you might have switched input channels on your TV set to your VHS tape machine or your Nintendo video game console. Blank input channels would give you white noise.

This noise does exist today when you are attempting to tune into digital HD broadcasting on the ATSC or DVB standard. However it is less prevalent. “Static-y” signals with poor reception now visually glitch in unexpected ways, instead of fading in the visual noise on top of your desired programming. Tuning into channels without a strong signal may be intercepted by your HDTV tuner and given to your TV screen as clean, “No Signal Found” user interface messages. And all of this is avoided by the prevalence of cable TV, which is also passed over for increasing preference towards streaming video internet services like Netflix. Static, snow, or noise of the 21st century is looking more like a ‘404’ page than black and white dots.

I thought of “noise”, considered a double meaning of word and ran with the concept. But my aesthetic mapping might be less “Kiki and Bouba” and more “Analog TV”. Something subjective that would not cross a cultural or generational boundary, but not necessarily as unique and specific as someone who experiences synesthesia. Not that there is anything inherently wrong with one approach over the other, but it is good to meditate on when you are building these links in your head.

A conceptual leverage point. A cultural touchstone. A future obscure reference. Use these things, but be aware of what they are. Are they universal, or are they telling little bits about who you are?

And we’re back

Ok, so lets play with this noise.

I’ve incorporated the following into my version of the code:

```
var vol = mic.getLevel();
var h = map(vol, 0, 1, 0, 20);

console.log(vol + “ “ + h);

for(i=0;i<1000;i++){
   //fill(color1);
```

```
    fill(random(colors));
  noStroke();
    rect(random(width),random(height),10,h+1);
}
```

Running this on different machines, sometimes it was smooth and on others there was some lag. Adding the mic input and drawing 1000 squares to the screen per frame might be harder on certain machines. Feel free to change the amount of squares drawn in your for() loop for whatever runs best.

Initial impressions: I like where this is going. Resizing the squares into small little lines makes the text more readable. Though now I'm thinking about TV static and snow after my tangent above. Its stuck in my head, can't get it out. My nostalgia is pulling at my brain: TV static never looked this sharp. Lets blur things.

```
//old background
//background(75);

//new, blurry background
background(75,75,75,25);
```

Adding two more integers to our background function lets us use color instead of black and white. A fourth integer specifies the alpha channel, making the color more see through. So you can see a bit of the previous frame as well as the new one. As the background draws over and over again, the previous frames will fade. But the combined effect creates a kind of motion blur.

Try playing around with the alpha value to see what feels good to you. One thing that I noticed is that with smaller alpha values (lots of blur), the scene looked like it had way more static than before. Now I can feel free to drop the amount of squares that I am drawing in my for() loop, if so desired. Feeding my nostalgia not only scratched an aesthetic itch but also introduced a possible way for my sketch to run a little faster. When these types of moments occur, take a moment to appreciate the artistic synchronicity. Maybe you're onto something.

What next?

If you have some inspiration by now… go off! Be free! Those moments don't just come any old time, you know. This tutorial will be here waiting for you after your wild artistic excursions. Good luck, and take plenty of screenshots!

However, you might be sitting at your computer wondering what you would even want to do next. Or more specifically, looking at what I've given you and thinking "...so what?". I feel this way sometimes about example code I see online. Perhaps you'll think it looks cool, sure. But how are you really going to use it? How do I make this piece into a "thing"?

Habit guides me towards putting in some kind of music as the sound source and making a visualizer. But that feels a little… eh, I don't know. Been there, done that, right? What is something that could give me something new, right now? Something that might be mine?

I'm going to propose a kind of conceptual framework for the rest of this series. When I'm trying to push an idea forward into something more fully realized, I'm going to

encourage you to put a part of yourself into the piece. This is easier said than done, so I'll put my money where my mouth is and "walk the walk" with you along the way.

I've done plenty of creative projects that have been tastefully removed from my own person. Simple shapes, straightforward color pallets and clean lines. Abstract geometry. Bleeps and bloops, clicks and whistles. It's all good fun, and I will most certainly draw from that well again. But it isn't directly "me", is it? And to be honest I've been a feeling a little uninspired lately. Haven't you? Or maybe just a bit worn out.

Also, I have to admit sometimes I feel like I have a bit of a chip on my shoulder about this whole "art with computers" thing. Once, in a conversation with a painter, a mutual friend of ours mentioned that I was an artist. He asked what kinds of things I painted, and I said my medium was the computer.

A quizzical look crossed his face. "I use pixels, not paint," I clarified, feeling pretty clever with myself having made the phrase up on the spot. "Ah, alright." His facial expression read, "oh, you're one of those". That ended the conversation pretty quickly.

And it doesn't help that technology is starting to feel a bit cold and impersonal. Or that people outside of our art and technology bubbles only seem to be drawn to the whiz-bang spectacles without thinking of the potential humanity behind them.

Something needs a shake up. Feels like now is just a good a time as any to incorporate ourselves into our work.

We need noise. So let us use our voices.

If we are going to speak, perhaps we will want to use a script. Good thing we're already putting text on the screen. The size is a little big for longer pieces of text, so we'll need to go smaller. Text in the string variable will get cut off if there isn't enough space for it on the screen. So always double check your visual output to make sure all your expected text is there.

Now because we're making a script for ourselves, we won't want random text. We want text in a sequence. Good thing our array can be used either way. There may have been other ways to write the code that we have here, but I've set myself up for this kind of flexibility and experimentation. Random text? Ask for a random array entry. Changed my mind, I want something sequential. Then step through the array incrementally. Backwards? Instead of ++, use --. Like my map() habits, I'm setting myself up to improvise and quickly iterate.

Maybe you'll find different structures that make more intuitive sense to you. In any case, invest time into making sure you can keep coding when inspiration strikes.

In order to track our progress through our script, we will need a global variable. Also, since we're changing the functionality and making a new version of our program, it might be a good time to rename some of our existing variables. Instead of "textChoices" as our text array name, I'm going to switch it to "textScript". That way we can wind up with lines of code that look like this:

```
displayText = textScript[scriptProgress];
```

Do you need to rename these things now? No, I guess not. But if you are trying to advance your practice, these little organizational things are going to go a long way. Tomorrow when you come back to this program, you won't confuse it with the previous version that does random text. If you want to show the code to a friend, they'll be able to read it a little easier and not ask you as many questions. And next year when you look back on all of your amazing progress, you'll maybe, kind of, almost remember how and why the code works.

Our new input

I've changed my keyPressed() function to the following:

```
function keyPressed(){
  if(keyCode===37){
    console.log("left key detected");
    if(scriptProgress==0){
      scriptProgress=textScript.length-1;
    }
    else{
      scriptProgress--;}
  }
  if(keyCode===39){
    console.log("right key detected");
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;
  }

  if(key==1){
    console.log("keypressed 1");
    colors = equalColors;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;
  }
  if(key==2){
    console.log("keypressed 2");
    colors = mostlyColor2;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;

  }
  if(key==3){
    console.log("keypressed 3");
    colors = mostlyColor3;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;

  }
  if(key==4){
    console.log("keypressed 4");
    colors = mostlyColor4;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;

  }
}
```

Just checking in:

Adding a global variable. Renaming variables. Changing a function. Have I lost you? If you don't feel fluent yet, just take each instruction one by one. Make the change, run the program and analyze any errors that come your way. I'll be giving you a full program later to reference, contrast and compare. But give troubleshooting a shot.

Being able to read these types of instructions isn't just

about throwing a pop quiz at you. Knowing this kind of language will let you digest more advanced material quickly, parse out those cryptic “help” posts on programming message forums, and pass your knowledge along to the people you will someday teach.

ASCII Keys

The first noticeable change in our new function is using keys that aren’t 1, 2, 3, or 4. The left and right keyboard keys appear to be 37 and 39. These are the ASCII key codes. You may have gone over this in your introductory programming material. I don’t actually have to use the key codes, as p5js builds in constants for you. Meaning, this line of code:

```
if(keyCode===37){
```

Should operate the same way as this line of code

```
if (keyCode === LEFT_ARROW) {
```

There are several other constant names that should work this way. You can find them in the p5js reference. Which you have open… right?! Right.

If you want to use these constants instead, feel free to do so. I like to keep my mind in ASCII land, just in case I decide I want to use a keyboard key that doesn’t have one of these constants. The “Home” key, for instance. So lonely, the Home key. Depending on the keyboard you are using, you might not even have one. And I don’t get in impression that people use it very much, even when it is available.

Strikes me as a little depressing, actually. An abandoned Home, permanently locked into the ASCII neighborhood. While Unicode Consortium pushes forward with all kinds of shiny new emojis, no one wants to go to the older parts of town anymore.

Seems like an artistic opportunity to me. So many potential heartfelt metaphors! Did you know you can search for idoms online? I just went to https://idioms.thefreedictionary.com to look up phrases that use the word home. Oh man guys, there are so many good ones: Home is where the heart is. Nothing to write home about. A man’s home is his castle. Hammer something home. Home away from home. You can’t go home again.

There is poetry lurking inside of our machines, I tell you.

These are all free art ideas, by the way. Maybe I’ll get around to doing something with this concept. For right now, it will stay in my notes app on my phone as an entry stating, “Home Key is depressing”. There is a moderate chance I will totally forget what that means by the time I look at it next. You should have a similar note taking system that logs all of your bizarre art ideas. Anyways, feel free to take this idea and run with it. I won’t show up to your gallery opening after you get famous, causing a scene, all yelling about how you stole my genius. No, I’m classier than that. The free wine will suffice. I’ll make do with a wink and a knowing nod as we catch each others glance from across the room.

The Home key ASCII code is 36.

Triple Equals

The triple equals sign could also be a bit of a head scratcher. You know the difference between = and ==, but what's the deal with ===? This is a strict check. This makes sure that not only is the value on both sides equal, but also the type. For example, booleans are not the same as integers. But boolean conditions can be interpreted as true;1 and false;0.

This means that the statement

```
if(true==1)
```

Evaluates to true, while

```
if(true===1)
```

Evaluates to false, because these are two totally different types of things (the essence of truth itself, and numbers).

You don't see this all the time, but pay attention when it comes up. If it is present in any code you come across, it usually means it is very needed. I'm using it here because that is how it was written in the reference documentation.

Not too big of a deal, but I didn't want you to be confused or thrown for a loop if you didn't recognize it.

Modulo

Oh, modulo. That's what we call the % sign in our code. Honestly, I'm a huge fan of modulo. Modulo is the best. I'm thinking I might need some kind of modulo branded clothing, a pin or sticker.

Modulo gives you the remainder of the division of two numbers.

That's it? That's what all the commotion is about? Yes. But it lets us do really useful things, and I have fully incorporated it into my mental programming toolkit. You see it repeated in these lines:

```
scriptProgress++;
```

scriptProgress=scriptProgress%textScript.length;
Say we come up with a textScript array with 4 entries. If we want to keep track and advance our progress through the script, we want that number to go from 0 (because array index locations start at 0) to 3 (the last entry in the array). Trying to reference an entry outside of the length of the array will either yield nothing, or crash your program depending on what your program is doing. Don't want that. To make everything work nicely, you could do something like this:

```
if(scriptProgress==textScript.length){
    scriptProgress=0;
    }
else{
    scriptProgress++;
    }
```

But the modulo method does the same thing, in fewer lines. Cleaner, perhaps a bit more readable. How is it working? When progress variable is 0, the returned remainder is 0. When the progress is 1, returned remained is 1. In fact, whenever your first modulo number is smaller than your second modulo number, you will be given your

first number back. This is true for our length of 4, or 100, or 1,000.

When the numbers are the same, you get zero. 4/4=1, with no remainder left over. 100/100, no remainder. 1000%1000? Zero.

Next is when the number on the left is larger than the right. 5%4 will return 1. 6%4 returns 2. 7%4 returns 3.

Lastly, 8%4 returns 0. Because four goes into 8 twice with no remainder. 9%4 returns 1, and so on.

This means that modulo is a good, quick way to develop conceptual structures that increment, sequence, and/or loop. And further, these loops start at 0 as opposed to 1. Making them a perfect pair with arrays, which also start at 0.

However, my modulo crush might be a little over stated. I still had to use the if/else structure for the left button decrementing functionality. Try using my modulo method with a -- instead and see how the program breaks. Don't worry, I won't take it personally.

Keys 1 through 4

You'll notice I kept the 1 through 4 key press detection. They all seem to advance the slides just like the right arrow key does. Seems a little redundant. But each one of them has the additional functionality of changing the color, each in their own way.

This is more of a conceptual example. When we're building interfaces, sending and receiving messages, and designing interactivity, there is opportunity in using multiple ins and outs while throwing in little bits of variation.

What if, when you write your script, you want to have a passage about diving underwater? "SPLASH!" is the next piece of text in your array. You can then press 2 to have your splash arrive with a cool blue wash over the screen.

But then you go to perform your piece at a tech poetry reading in Chelsea, and the vibe is all different than you expected. The bright blue and yellow suddenly strikes you as a bit gauche. And water paired with blue, isn't that a little on the nose? It all felt so different rehearsing at home. Magenta, that's the one for this room. A bit darker, more mellow. But warm. A playful, dim romance. Press 3.

These are all programming questions and technical scenarios. If you want your art to seem less mechanical, if you want your performances to have more room for improvisation, consider introducing these little differences to identical functionalities. The act of walking "naturally" is a repeating of steps, but the speed, stride and direction can all vary in little ways. Even with something a simple as words, colors and shapes, we can introduce little bits of our artistic gait.

Make your own

Now is the time to make your own piece. Perhaps this can be projected ten feet wide, while you stand in front of it. Or you could just do a screen recording while you speak the words. Or, maybe you can just write something for yourself, say it once right here and now, and never repeat it every again. In any case, if you're struggling for something to say, tell the program something about yourself.

If you are having technical issues with the code, or just want to see what I came up with, you can look at or use the code below:

```
//color choices
var color1;
var color2;
var color3;
var color4;

//an array to hold all of the colors
var colors;

//text we are looking at
var displayText;

//our microphone
var mic;

/*
You can use this:
“+
“


or this:
‘+
‘
To break up long strings of text, in order to make things more readable
They will work exactly the same, but they will fit all on a visible screen
        instead of flying off into the endless whitespace of the right hand side.

*/
```

```
var textScript = [‘You probably could just say “lalalalala” into the microphone and test the functionality ‘+
‘of the code just fine. But you don’t. For whatever reason you decide to speak the words ‘+
‘on the screen aloud, like a speech. Perhaps even if just under your breath. At first this ‘+
‘spares you from feeling funny about appearing strange, blurting out some rant into the ‘+
‘multicolored mess on the screen. However, you start to wonder how much your soft muttering’+
‘ is really affecting the height values of the squares being drawn.’,
‘You draw closer to your microphone, hoping for more visible results without risking any ‘+
‘more of your ego. Now closer to the screen, you notice you might be making some difference.’+
‘ Though getting closer to the microphone has also brought your face closer to the screen as ‘+
‘well. The previously invisible black grid in between the pixels vaguely starts to emerge.’,
‘You are squinting now, face lit up by the neon slurry of the screen. There is no escaping it,’+
‘ you’ll need to speak louder. Gradually, you raise your voice. There is a noticeable’+
‘ difference. Or maybe that is just in your head. You are expecting the changes to happen’+
‘ so maybe you are more primed to think they are happening (even when they are not). The’+
‘ pauses between words seem to help. Abruptly raising your voice at the beginning of each’+
‘ word seems to make the static pop to life. A kind of strange rhythm emerges in your’+
‘ speaking, bouncing up and down between muttering and recognizable words.’,
‘“Thats me,” you think to yourself. Perhaps a bit of a journey,
```

```
but yes, you feel like the ‘+
‘statement is right. Out of your mouth, into the micro-
phone, through the hardware, into the ‘+
‘operating system, flying through layers of drivers and
system utilities, landing into your ‘+
‘browser. Its you. You are looking at yourself, in a way.
Certainly a strange mirror. But ‘+
‘they aren’t really your words. Should you write your
own?’,
‘You were a shy child. Well, at first. Shy until certain
occasions came up where your family ‘+
‘and school mates were surprised to see you perform
so boldly in front of so many people. ‘+
‘Book reports, dance recitals, school plays. But after
these outbursts, you would return to ‘+
‘a more introverted and calm disposition. As these so-
cial surprises became more common, the ‘+
‘unexpected became expected and people thought of
you differently. You had changed, though ‘+
‘not entirely sure where or when. Or how much.’,
‘Eventually you stopped performing from scripts and
started making your own things. Perhaps ‘+
‘some music. A painting. Bad poetry. Oh wow, so much
bad poetry. In fact, as you became an ‘+
‘adult you started looking back at all the things you
made with a little bit of embarrassment.’+
‘ Its ok, though. You have to start somewhere. Practice
makes perfect. But now you always ‘+
‘consider how what you make is going to look, going to
sound. Why make something you’ll be’+
‘ unhappy with tomorrow? Best to think a bit first. Es-
pecially when so many people you know’+
‘ are making and doing so many amazing things.’,
‘Of course, you have indulged this habit a bit too much
lately. Thinking only a bit turned’+
‘ into thinking a lot. A whole lot. Inspiration comes in fits and
starts, usually away from’+
‘ your gear and during busy occasions where you can’t spare
a moment. You DO want to write’+
‘ your own words into the tutorial. But you simply don’t have
any good ideas. You want to be’+
‘ able to write something amazing, something real. Some-
thing you wouldn’t feel strange about’+
‘ speaking out loud, at full volume.’,
‘“Maybe later,” you think. A moment passes. “I’ll come back
to it,” you half heartedly insist.’+
‘ You’re tired, anyways. And the flashing colors and static
have made your eyes a little sore.’+
‘ How long have you been squinting for? What time is it? You
roll your shoulders backwards’+
‘ and straighten out your spine. Aching back and neck, again.
You close your eyes for an’+
‘ extended moment, seeing the multicolored snow dancing
on the inside of your eyelids.’,
‘“That’s enough for now,” you confidently decide. You eat a
snack while watching some TV, then’+
‘ take a shower before going to sleep. You dream of filling up
ice trays with cubes that take’+
‘ the shape of words and letters you can’t quite make out.
Tray after tray after tray you pour’+
‘ water and load into the frosty machine. But your freezer
doesn’t seem to run out of space.’+
‘ The water from the sink is never turned off, and becomes a
singular source of white noise.’+
‘ A moving, rushing blankness. It feels good. You delve into a
deeper, dreamless sleep.’,
‘end of the array’,];

//our current location in the script
var scriptProgress;
```

```
function setup() {
  createCanvas(400, 400);

  mic = new p5.AudioIn();

  // start the Audio Input.
  // By default, it does not .connect() (to the computer
speakers)
  mic.start();

  color1 = color(255,255,255);      //white
  color2 = color(0,255,255);        //cyan
  color3 = color(255,0,255);        //magenta
  color4 = color(255,255,0);        //yellow

  equalColors = [color1,color2,color3,color4];
  //equal chances of picking 4 colors

  mostlyColor2 = [color1,color2,color2,color2,
                  color2,color2,color2,color2,
                  color2,color2,color3,color4];
//more likely color 2 is picked

  mostlyColor3 = [color1,color2,color3,color3,
                  color3,color3,color3,color3,
                  color3,color3,color3,color4];
//more likely color 3 is picked

  mostlyColor4 = [color1,color2,color3,color4,
                  color4,color4,color4,color4,
                  color4,color4,color4,color4];
//more likely color 4 is picked

  colors = equalColors;
  //we'll start with the first option

  //textScript = ['part1','part2','part3','part4'];

  //starting in the array position zero, first part of our script
        scriptProgress = 0;

  //put the current script as the text that will be displayed
  displayText = textScript[scriptProgress];
}

function draw() {
  //background has four arguments: the fourth is alpha
  //having this not be 255 allows for bluring effect
  background(75,75,75,25);

  //get the level of the microphone volume and re-map the
values
  var vol = mic.getLevel();
  var h = map(vol, 0, 1, 0, 20);

  //for testing mic values in the console
  //console.log(vol + " " + h);

  //five hundred times per frame
  for(i=0;i<500;i++){
        //We create a square

    //that pulls randomly from our currently selected colors
array
    fill(random(colors));
    noStroke();
```

```
    //placed in a random location,
          //with a height determined by the mapped vol-
ume value
          rect(random(width),random(height),10,h+1);
  }

  //put the current script as the text that will be dis-
played
  displayText = textScript[scriptProgress];

  //display that text
  fill(255);
  textSize(18);
  textFont('Helvetica');        //because helvetica
  stroke(255);
  noStroke();
  //strokeWeight(1);
          text(displayText, 25, 25, width-35, height-25);

}

function keyPressed(){
  if(keyCode===37){
    console.log("left key detected");
    //if the left key is pressed

    //and we are at the beginning of our script
    if(scriptProgress==0){
      //loop back around to the last script entry
      scriptProgress=textScript.length-1;
    }
    else{
      //otherwise, just go back one in the script array
      scriptProgress--;}
  }

  if(keyCode===39){
    console.log("right key detected");
    //if the right key is pressed

    //go to the next location in the script array
    scriptProgress++;
    //run modulo on the script progress variable, ensuring
          //that we never run over the maximum length of the
array
    scriptProgress=scriptProgress%textScript.length;
  }

  if(key==1){
    console.log("keypressed 1");
    colors = equalColors;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;
  //like the right button, but setting a specific color option
  }

  if(key==2){
    console.log("keypressed 2");
    colors = mostlyColor2;
          scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;
  //like the right button, but setting a specific color option
  }

  if(key==3){
    console.log("keypressed 3");
    colors = mostlyColor3;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;
  //like the right button, but setting a specific color option
  }
```

```
  if(key==4){
    console.log("keypressed 4");
    colors = mostlyColor4;
    scriptProgress++;
    scriptProgress=scriptProgress%textScript.length;
 //like the right button, but setting a specific color option
  }
}
```

A Brief Review of Objects

If you want to create more advanced programs as a creative technologist, you will want to be comfortable with making your own objects. Not physical (though that will come later), but digital. Objects inside of your programs. Instead of relying on things like strings, integers and floats, you'll be making your own.

Which is something you will have most likely covered in your introductory programming classes. Lets review. Making your own "Ball" object class might include basic features:

- Position in space
- Size
- Speed
- Ability to move
- Ability to be seen
- I like to think of these as adjectives and verbs.

Some things describe the state of the thing: It's big, it's slow, its off in the distance. These are adjectives. They are made by defining variables of an object.

This may start out looking like:

```
function Ball() {
this.x = random(width);
this.y = width/2;
this.diameter = random(100, 400);
this.speed = 1;
```

X, Y, diameter, speed. This is the start of our Ball constructor. And in Javascript, we use the “this.variable” convention when making our blueprints for a new object inside of our constructor.

This can be seen again as we continue on to the verbs. These are things that each instance of the object can do: it exists, it moves.

```
this.move = function() {
  this.x += this.speed;
};

this.display = function() {
  ellipse(this.x, this.y, this.diameter, this.diameter);
}
```

To make the ball even be seen, we will wind up doing things like this when we are inside of our draw loop:

```
myShinyNewBall.move();
myShinyNewBall.display();
```

A tour of my personal object

Creating the concept of a ball, and then making a shiny new ball for yourself. It reminds me of being a kid. You realize that there is such a thing as a ball. You find out what it is and what it can do. And then you want one for yourself. Naming an object, defining its qualities, and then making a variable with it.

But we know children will grow bored of their toys. No matter what, it seems that they always lose their luster. Or maybe it is us that changes, our way of looking. I’m not sure I always see the shine like I used to.

Things have changed, for better and worse. I would love to take a look at the world through those young eyes again, but only for a while. Who would really want to be a kid again permanently? It’s just concerning that the further forward you go, the more likely you are to forget what it was like back then. Can I go forward, and still remember how I felt way back when? I don’t want to be enamoured with the shiny ball, trapped by its allure. But I do want to remember the way it made me feel.

These are not the feelings I see in usual object constructors. Usually we see traditional taxonomies: A flock contains a number of birds, a bird has a color and speed and number of feathers, and so on. But our objects, their variables, and their functions can be named whatever we like. Maybe we can communicate more about ourselves instead.

What would you like to do?

Casual Activity:

Below is a piece that is a representation of my life, as I feel about it in this moment. Feel free to take a look at the code to see how the object is created, used, and then modified by various functions.

Curious Activity:

If you would like to tell your own life story, take a look at how the diameter, speed, color, outsideColor, and ourWalls properties are used. Feel free to make your own version by changing the variables to suit your story. Let me know if you'd like to share it with me to use for my documentation, or simply just describe the thoughts behind your life objects to someone else here.

Creative Activity:

Or, change the whole narrative! Isn't there more to life than just age? Why is time described as going from left to right? Do you feel like more of a square than a circle? Feel free to modify, remove, or add as much as you need to tell your life story.

My story can be found over the next few pages. You can look at the live running sketch at the following URL:

http://alpha.editor.p5js.org/dbarrett/sketches/HJrgI6Uqz


```
/*
Originally inspired by
https://p5js.org/examples/objects-objects.html
*/


var me;  // Declare variable

function setup() {
  createCanvas(800, 400);

  // This variable type is that of the object we will
create
  me = new You();
}

function draw() {
  background(10, 10, 100, 5); //R,G,B,A, with low
alpha for blurring

  //We move. We exist.
  //(These are defined in the constructor after our
draw loop)
  me.move();
  me.display();

  //We go through life.
  life(me);
}


//Creating our object
//a you
function You() {
//
```

```
  this.diameter = random(10, 30);

  //Life has to start somewhere
  //In this case, the left side of the screen
  this.x = 0;
  //fixed to middle height, to lead a balanced life
  this.y = height/2;

  //What is our color?
  //No, not on the surface. Deeper.
  this.color = color(255,255,255);
  this.outsideColor = color(255,255,255);
  this.ourWalls = 1;

  //The longer you live, the faster life seems to go
  //But time doesn't change, you do. How fast are you
going?
  this.speed = 1;

  //And where are you headed?
  this.move = function() {
    this.x += this.speed;     //move x
    this.x = (this.x%width);     //limit x to size of the
canvas
  };


  //It seems funny to me that "existing" needs to be a
verb
  //But hey, sometimes life is more difficult than we like
  //Haven't you ever felt the most basic things were an
effort?
  //Or "not all there?"
```

```
  this.display = function() {
    strokeWeight(this.ourWalls);
    stroke(this.outsideColor);
    fill(this.color);
    ellipse(this.x, this.y, this.diameter, this.diameter);
  }
}



//Life.
//Life Function
//A thing that you go through
//You come in one side and come out the other
//It changes you



function life(people){
  //I'm thinking about a life time: Youth, Adolesence,
Adulthood
  //If our position in life is in the first third, we are
children
  //Second third, teenagers
  //Last third, adults

  if(people.x < width/3){
    youth(people);
  }
  else if (width/3<=people.x && people.
x<2*(width/3)){
        adolesence(people);
  }
  else
  {
```

```
    console.log("We should be in adulthood. Right?
What else is there, I guess...");
                adulthood(people);
  }
  //maybe these distinctions are out dated
  //or maybe they fit, but our concept of them should
change

  //Having trouble with life? Try these debugging mes-
sages:
  //console.log("what is life telling us?");
  //console.log("Our position in life: " + people.x);
}



//But life isn't just "LIFE", ya know? Its made up of
different things
//So each of those things needs to be defined


function youth(people){
  if(people.diameter){//are we dealing with a person?
     people.diameter = 10;

     people.speed = 0.66;
     people.outsideColor=color(220,220,0);
     people.ourWalls = 0.5;
     return;}
  else{//if we're not dealing with a person, we don't
use this function
       //this prevents errors, as we're setting qualities
unqiue to the object type "you"
       //The number 17 didn't have a childhood, so this
function doesn't apply to it
       //Which sounds kind of sad for the number 17,
but that is a bit of a distraction for right now
  return}
}



function adolesence(people){
  if(!people.diameter){//if we aren't dealing with a
You class
       //the "!" in this if statement means "not"
       //Same type of results as above, we're just
defining things in the negative
       //Which sounds very adolesent to me
       console.log("this isn't you, is it?");
     return}
  else{
    people.outsideColor=color(175,20,255);
    people.ourWalls = 2.5;

    //Our color on the inside. Our true color.
    people.color=color(22,22,100);

    people.diameter += 0.2;
    //I grew over time


    return people;
  }
}
```

```
function adulthood(people){
  //and all of a sudden I was an adult before I knew it

  if(people.diameter){//are we dealing with a person?
     people.diameter = 125;
     people.speed=1.66; //where does the time go?

     people.color=color(188,188,255);
     people.outsideColor=color(255,11,175);
     people.ourWalls = 1;

     return;}
  else{

  return}
}
```

Colors. Sound. Considering how we can break down our concepts into discrete elements and objects. This may have been a review for some of you, but hopefully there have been some novel ways of approaching creativity that are useful for everyone.

Adding MIDI

So far I've talked about how to spice up simplistic examples you have come across in your introductory materials. There has been some discussion about "mapping", "scaling" or general translating of values for various purposes. I've done a brief review of objects. I've talked about the importance of reading documentation, encouraged you to write and read code comments, and tried to push you to copy/paste/modify without strict step-by-step instructions.

I'm now going to recap these themes with another small program. This should be similar to something you may have made already in your intro programming courses. Now we're going to make bouncing balls. In the previous review of objects, we might not have really needed to make the ball an object. But this time we definitely need one.

We want to make lots of bouncing balls. We want things to happen every time each individual ball bounces. We want them to each to live a certain amount of time, and then disappear. We want gravity to affect them. We never expected some bouncing balls to demand so much! This is a classic use case for an object.

Instead of going through piece by piece or building this from scratch again, I'm just going to give you a fully functioning sketch. The comments will serve as a guide to what is going on:

```
/*
duplicated with much love from the bouncing ball
example
in my Introduction to Computational Media class with
Allison Parish

re-worked, extended, and otherwise mangled the ball
object to add;
constructor, sounds, lifespan, etc.
*/

var balls = [];   //array to contain ball objects
var gravity = 0.1;   //gravity

function setup() {
  createCanvas(400, 300);

  //start with a single ball
  //i've set up a for loop for you in case you want more

  for (var i=0;i<1;i++){
        balls[i] = new Ball(random(width),ran-
dom(height),24);
    //a sized 24 ball is placed in a random position
  }
}

function draw() {
  background(0);

  /*every frame, we need to:
                1) show
                2) move
                3) check for a bounce
                4) decide whether to remove the ball
        For every ball in the array of balls
  */

  for (var i = balls.length-1; i>=0;i--){
    balls[i].display();
    balls[i].move();
    balls[i].bounce();

    if(balls[i].isFinished()){
        console.log("finished: removing a ball");
        balls.splice(i,1);
    }

  }
}

function mousePressed(){
  //When we press the mouse button

  //we add a new ball object to the balls array
  //at the location of the mouse, with a width of 24
  balls.push(new Ball(mouseX,mouseY,24));

  //How many balls do we have?
  console.log("length="+balls.length);
}



//The Ball object
function Ball(_x,_y,_size) { //<-- "_x,_y,_size" are
construction arguments
  //in order to create a ball, these three things have
to be defined
```

```
  //Position, size, and speed of the ball
  this.x= _x;
  this.y= _y;
  this.size= _size;
  this.speed= 0;    //some object properties can be set
by default
  ///(they don't have to be required constructor argu-
ments)
  //in this case, all balls start with a speed of zero

  //How long should the ball exist
  this.lifespan=10*60;
  //Time will be measured in frames
  //assuming 60 frames per second
  //10*60 = 10 seconds

  //volume of the sound
  this.volume=1;
  //level of transparency
  this.alpha=255;

  //the sound file
  this.ballSFX = loadSound('piano.mp3');

  //the speed at which we will play back the sound file
  //this number is linked to where the ball is inside of
the sketch:
  //bounces on the left side are deeper, slower
  //bounces towards the right are higher, shorter
  this.sfxSpeed=this.x;
  this.sfxSpeed=map(this.sfxSpeed,0,width,0.01,4);

  //show us the ball (will be called every frame)
  this.display = function() {
    //the alpha transparency will fade until the ball is
destroyed
    this.alpha=map(this.lifespan,0,300,0,255);
    fill(255,this.alpha);
    ellipse(this.x, this.y, this.size, this.size);

    //the speed of the sound playback
    //is assigned to the sound file
    this.ballSFX.rate(this.sfxSpeed);
  };

  //move the ball (will be called every frame)
  this.move = function() {
    //The ball moves and is effected by gravity
    this.y = this.y + this.speed;
    this.speed = this.speed + gravity;

    //the volume of the sound effect gets quieter as is
fades out
    this.volume=map(this.lifespan,0,this.lifespan,0,1);
    console.log("mapped vol:" + this.volume);
                this.lifespan--;
  };

  //bounce the ball (will be called every frame)
  this.bounce = function() {
    //is it time to bounce the ball?
    if (this.y > height) {
      //slow down the speed of the ball over time
      this.speed = this.speed * -0.97;

      //play the sound effect when we bounce
      this.ballSFX.play();

      //change the volume of the sound
      this.ballSFX.setVolume(this.volume);
```

```
    /*In theory, we could call this every frame, right?
      Though, we are only hearing the difference in
volume
      every time the ball bounces.
     So why make changes to each object more
times than we have to?
      We will set it here in the bounce function.
    */

    console.log(this.volume);
  }
 };

 //is it time for the ball to go?
 //this will be checked every frame
 this.isFinished = function(){
  if (this.lifespan<0){
   return true;
  } else{
    return false;
  }
 };

}
```

You can find this p5 sketch online here:
http://alpha.editor.p5js.org/dbarrett/sketches/rkqd5uT9M

If you want to re-create this yourself or run it locally, you will need to have a sound file named “piano.mp3” in the same directory as the sketch.

Things you can do:

Casual:

Read through the code, then run it. Does it do what you expected it to do? Can you copy and paste the code somewhere else and have it run? In the online editor? What about the offline editor? Have you tried using text editors (Atom, Sublime, etc) to develop p5 sketches? Now could be a good time to practice.

Curious:

Sure are a lot of variables. You could change all of them! And I made the balls plain ol’ white. How boring. You certainly would want to change that. And what’s the deal with the piano noise? Surely something else could go in there.

Creative:

You’ve changed the variables. Always fun. How about adding variables? What if you made each ball a different color by giving them their own color variables? Or change the functions. Could the creation of a ball and the destruction of a ball trigger new functionality? Maybe add a function. Setting multiple variables at once could done with one function (new color, size, and sound all in one command).

Challenge:

But you said there’d be no challenges! Well, there are no required challenges. But I know some of you. You want to be pushed. You want something specific. All this artsy feel good mumbo jumbo just isn’t doing it for you. Ok ok, I get it.

How about the following as a challenge? I probably could have made the lifespan a variable that was set in the Ball

constructor, like the x, y, and size variables. Make lifespan a required variable when creating a ball object. Then, when creating a ball, give it a random lifespan. And how would you double check that the lifespans were actually different?

The Next Step

You want to make more involved and innovative programs. You are a creative technologist. An artist. You want software, you want hardware. You want performance, interaction, synchronicity. Magic.

The optimism and excitement of a good idea, or even just a fresh sense of motivation, is a special thing to have. Next I'll be outlining a multi-step process that will perhaps start off a little dry, and could sap some of that excitement and motivation. There is arcane technical history. There is installation of new libraries. And configuring system settings. Not exactly spicy stuff.

But what I'm going to be sharing is a setup and approach that helps me quickly and easily iterate on new ideas. It allows me to work on multiple platforms. It allows multiple programs, made with different languages, to talk to each other. It helps me use cheap, pre-made hardware to interact with my programs. It helps me make my own hardware that can easily work, with no fussy installation, with the programs I make.

First there are the boring things, like wrestling with system settings and application preferences. No one likes that stuff. However, this is an investment. It has helped me get to a place where I can make stuff fast, and be flexible enough that I can improvise easily when inspiration strikes.

What I'm doing here may or may not be your perfect way of making. But the broader lesson is that you should invest in programming languages, systems, libraries, hardware, software, and so on, that let you create as seamlessly as possible. This may involve a bit of research. Just knowing about what kind of technology is out there can be hugely beneficial. Technology is your artistic canvas. Knowing your materials and using what works best for you will make you a better artist.

So even if this doesn't wind up being "your way", the following exercise will help you practice "a way".

Adding another library

This far we have been working with p5js, which is a library for Javascript. A library is really just other people's code. Sometimes it's Javascript, other times C++, or any other programming language. The important part is that someone made something for you. They made a bunch of functions and objects. If they're really nice, they also documented those functions and objects kind of well. If they're really, really nice they documented them very well and given you good examples of how to use them.

Thanks Lauren McCarthy! Thanks Processing Foundation! You made my life easier.

I have another library that I have simply been loving lately. It's called WebMidi.js:

https://github.com/cotejp/webmidi

It lets you use the MIDI communication protocol in your browser based programs. In order to use it, we need to in-

clude the library in our sketch. We are going to build off of our previous bouncing ball object sketch. If you go to your standard p5js index.html file, you'll see the following lines:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.2/p5.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.2/addons/p5.dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.2/addons/p5.sound.min.js"></script>
```

This pulls the most recent version of the p5 library, the p5 DOM library, and the p5 Sound library from a web server. If you needed to use a local copy of the library (like if you were running a browser based program on a machine without internet), you would do something like this:

```
<script src="p5.min.js"></script>
```

And then you would make sure that p5.min.js file (or whatever it is you want to run locally off of your machine) was in the directory that your other index.html and sketch.js files were in. I want to add my new favorite library, which will look like this:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.2/p5.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.2/addons/p5.dom.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.2/addons/p5.sound.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/webmidi"></script>
```

After that, we're going to initialize the library inside of our setup function. The setup function will look like this:

```
function setup() {
  createCanvas(400, 300);

  //start with a single ball
  //i've set up a for loop for you in case you want more

  for (var i=0;i<1;i++){
        balls[i] = new Ball(random(width),random(height),24);
    //a sized 24 ball is placed in a random position
  }

  ////
  //Adding MIDI functionality
  ////

  //Enable the WebMidi Library
  WebMidi.enable(function (err) {

    if (err) {
      console.log("WebMidi could not be enabled.", err);
      //What happens when there is an error
    } else {
      console.log("WebMidi enabled!");
      //What happens if there isn't an error
    }

    console.log("---");
    console.log("Inputs Ports: ");
    for(i = 0; i< WebMidi.inputs.length; i++){
      console.log(i + ": " + WebMidi.inputs[i].name);
      //Listing all of our MIDI input ports
    }

    console.log("---");
    console.log("Output Ports: ");
```

```
    for(i = 0; i< WebMidi.outputs.length; i++){
      console.log(i + ": " + WebMidi.outputs[i].name);
      //Listing all of our MIDI output ports
    }
          });

  var output = WebMidi.outputs[0];

}
```

If everything has gone right, when you run your program it will work as usual. Except that you will see something like this in your console log:

```
WebMidi enabled!
---
Inputs Ports:
---
Output Ports:
```

What is MIDI?

https://en.wikipedia.org/wiki/MIDI

MIDI stands for Musical Instrument Digital Interface. You may have heard about it in different contexts. Maybe you think of old school Nintendo soundtracks as "MIDI music". Or perhaps there is a dusty MIDI keyboard in your parent's closet. You may have seen a MIDI cable, or downloaded a MIDI file.

Kind of confusing. MIDI is described as a "technical standard", which consists of a standardized communications protocol, and physical standard for interfaces, connections and cables.

Created in 1983, it was mainly a way to get different pieces of musical gear to talk nice to each other. One machine might say to another machine, "I want to send this note, for this long, right now". The first machine is controlling the second. The second machine has a voice that creates the sounds according to the MIDI message, while the the first machine has no voice at all. It just sends commands.

The piano keys that you may see on a synthesizer are the interface to the machine that makes the sounds. Just like our computer keyboards are not one and the same as our computers, MIDI controllers (like a musical keyboard) are not the same as the things that make the noise (a synthesizer). With this new standardization, one musical keyboard was able to work with synthesizers from all kinds of different companies. Additionally, one keyboard could send the same message to multiple synthesizers at the same time. You use MIDI cables to connect this hardware to each other.

And then the messages could be manually programmed and played back if desired, making a kind of digital sheet music. These are MIDI files. And since the live MIDI mes-

sages and MIDI files are digital, all of this technology was implemented into computers not too long after MIDI was released. Composition of musical scores could be played back on computers with the right hardware and software, which is what you might associate with simplistic “video game MIDI music”.

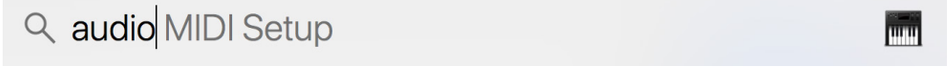
All of this continued to be implemented in new musical gear, new computers, and new operating systems as the years went on. Most importantly, virtual versions of the physical elements were made. A program that generates MIDI messages can send those messages to a program designed to receive them and makes sound accordingly. The computer has a virtual port that facilitated a connection between the MIDI output of one program to the MIDI input of another. You can think of this like a virtual cable that connects the two.

It is this virtual nature that is going to help us create some really cool stuff.

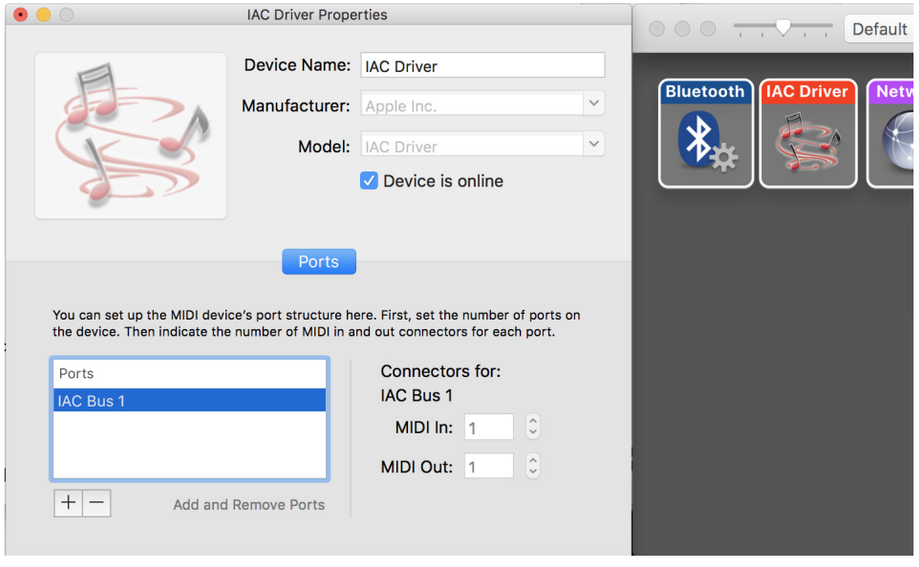
...but where are we virtually plugging in?

Enabling Virtual Ports

MacOS has the ability to make virtual MIDI ports using system settings. Do a spotlight search on your machine for the application “Audio MIDI Setup”.



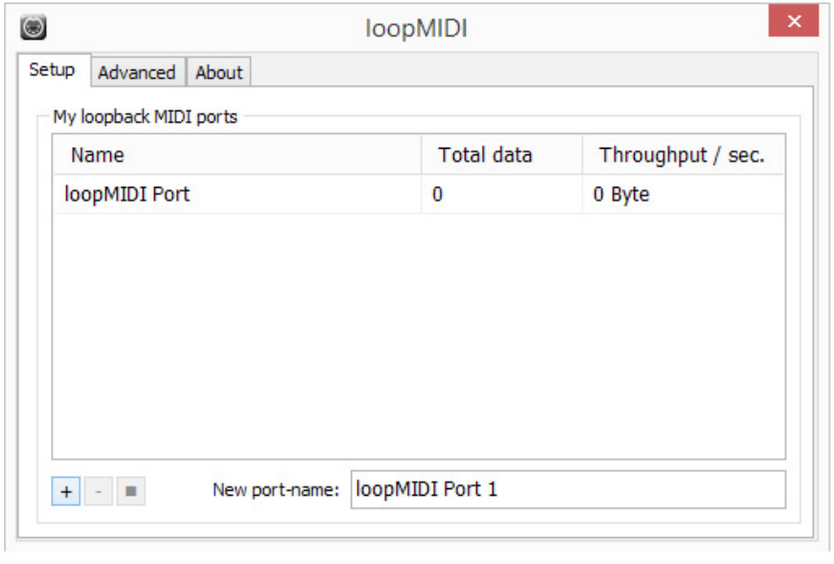
Inside of Audio MIDI Setup go to Window>MIDI Studio. Double click on the IAC Driver box. A properties window will open.



Click on the Plus icon to add a “MIDI Bus”. You can come back here to add more later, but for now one is fine.

Windows does not have this ability natively. However, there is an excellent free program that has worked well for me called loopMIDI:
https://www.tobias-erichsen.de/software/loopmidi.html

loopMIDI functions and feels very similar to the MacOS’s MIDI setup. Once running, you can add or remove MIDI buses using the “+” and “-” buttons in the lower left hand corner.



Once you have made a virtual MIDI port, run our bouncing ball program again. You should see something like this in your console log:

```
WebMidi enabled!
---
Inputs Ports:
0: loopMIDI Port
---
Output Ports:
0: loopMIDI Port
```

That means that our program can send and receive on this port.

But, send and receive… what?

A single MIDI port can send lots of things, but to keep things simple we're going to focus on only a few specific elements.

Notes

A piano keyboard has different keys you can play. A musical score can have notes printed on a staff. A MIDI message can tell a MIDI instrument to play a note in the same way. There are two kinds of notes: a note on and a note off.

Velocity

Imagine hitting that piano key as hard as you could. Imagine a violinist tenderly pulling their bow across the strings of their violin. A single note can be hard, or soft. Generally, this affects the volume of a particular note. Every note is paired with a velocity. In many instances where a velocity is not given, the default maximum value (127), is used.

Control Changes

Control Change messages are usually referred to as CC messages. This is a kind of "general purpose" communication. Imagine a synthesizer with lots of knobs and sliders. Maybe they don't send note messages, or the velocity of the notes. But they could control other aspects of the sound or the function of the system. Things like bending the pitch of a note can be assigned to these types of controls.

Channels

Within any given MIDI connection, we might be sending note ons, note offs, all kinds of velocities and throwings all sorts of CC messages around. Imagine all of that communication going through a MIDI cable, and then multiply it by 16. There are 16 channels available on any given MIDI

connection. These channels can be dedicated to things like multiple instruments. For example, a piano on channel 1, channel 5 could be guitar, and usually channel 10 is drums. However, this isn't set in stone and the channels can go to wherever you want.

Let start by sending a note message. Add the following code inside of your Ball object's "bounce()" function:

```
//Setting the MIDI output of the Ball object
this.output = WebMidi.outputs[0];
//play the note on that output
this.output.playNote("C5");
```

This will mean that a C5 note is sent through the webmidi output 0. When we saw "0: loopMIDI Port" or "0: IAC Driver IAC Bus" or whatever you named your virtual MIDI port, this is what we are referring to. The inputs and outputs are arrays, and 0 is the first output that is available.

Now we're going to save this sketch, and set it aside for a moment.

Next, we're going to make a whole new sketch. Don't despair! It's a quick, easy little guy. You can find it here:

http://alpha.editor.p5js.org/dbarrett/sketches/HkUhtS7PM

With the following code:

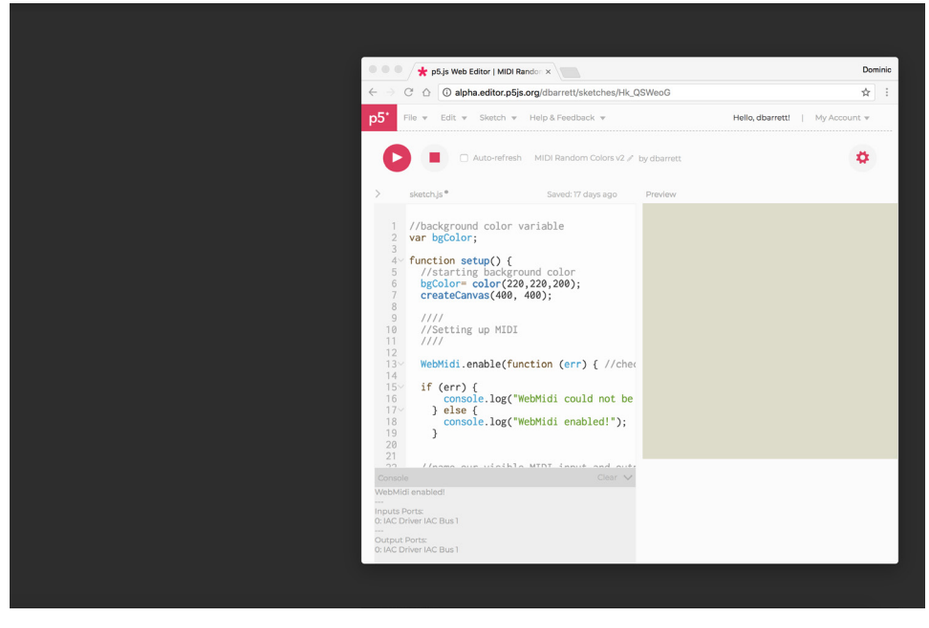
```
//background color variable
var bgColor;

function setup() {
  //starting background color
  bgColor= color(220,220,200);
  createCanvas(400, 400);

  ////
  //Setting up MIDI
  ////

  WebMidi.enable(function (err) {
  //check if WebMidi.js is enabled

  if (err) {
      console.log("WebMidi could not be enabled.",
err);
    } else {
      console.log("WebMidi enabled!");
    }


  //name our visible MIDI input and output ports
  console.log("---");
  console.log("Inputs Ports: ");
  for(i = 0; i< WebMidi.inputs.length; i++){
    console.log(i + ": " + WebMidi.inputs[i].name);
  }

  console.log("---");
  console.log("Output Ports: ");
  for(i = 0; i< WebMidi.outputs.length; i++){
       console.log(i + ": " + WebMidi.outputs[i].name);
```

```
  }

  //Choose an input port
  inputSoftware = WebMidi.inputs[0];

  ///
  //listen to all incoming “note on” input events

inputSoftware.addListener(‘noteon’, “all”,
        function (e) {
        //Show what we are receiving
        console.log(“Received ‘noteon’ message (“ +
e.note.name + e.note.octave + “).”);

         //change the background color variable
        var randomR = random(0,255);
        var randomG = random(0,255);
        var randomB = random(0,255);
        bgColor = color(randomR,randomB,randomG);
        }
  );
    //
    //end of MIDI setup
    //
        });

}

function draw() {
  //draw background with background color variable
  background(bgColor);
}
```

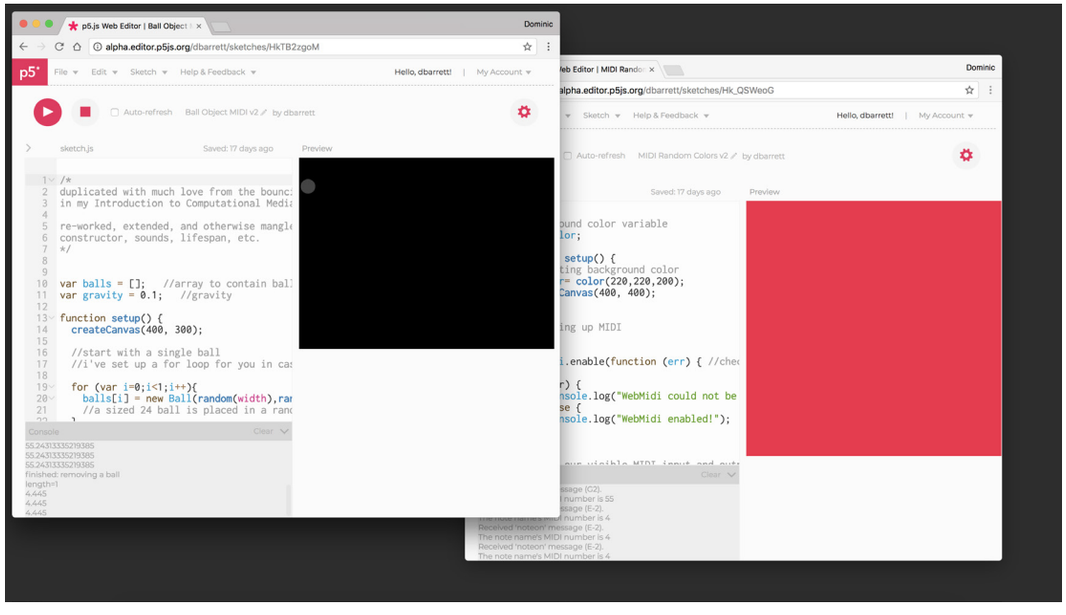
What this says is: whenever I receive a “note on” MIDI message on input port 0, I am going to console log information about the message I received. Then I am going to change the background color variable to something random. Then when I draw to the screen, I’ll use that background color.

Now run this sketch. In the console log you will hopefully see the successful initialization messages. If you got errors instead, and you are copying and pasting this code into a new document… did you remember to include the library in your HTML?

If everything is working, you’ll just see a pale square. Keep that sketch running in one window.



Now go back to your MIDI enabled bouncing ball sketch. Run that as well, where you can see both windows. Now click to create a ball.



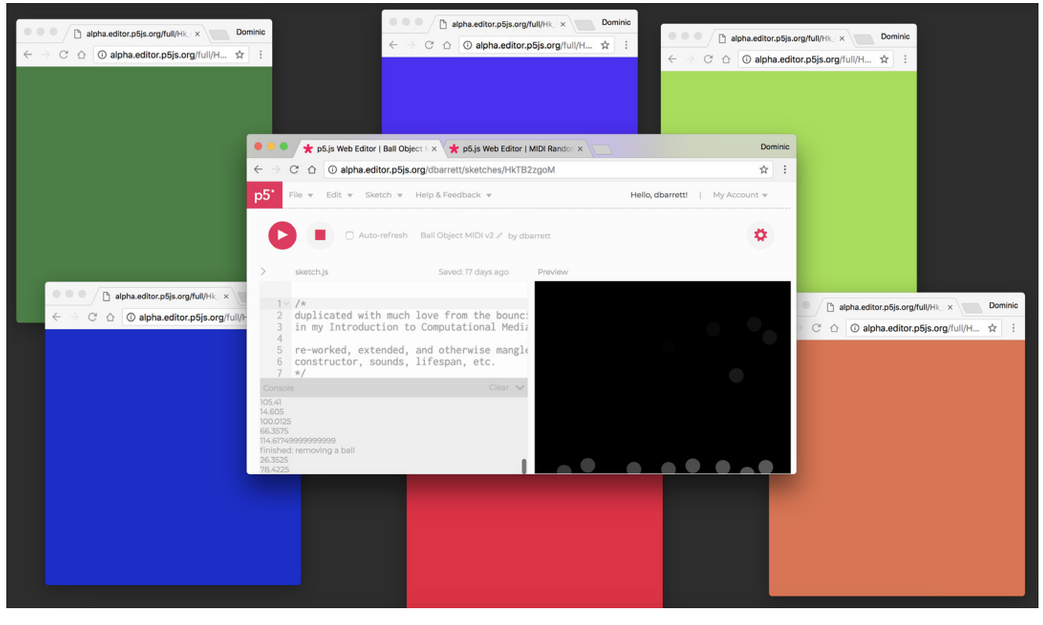
When the ball bounces on your first sketch, you should see the color change in your other sketch.

Next, copy the URL for the color change sketch and open a new browser window. In the p5 online editor, you can go to File>Share and copy the “Fullscreen” link if you want to only see the sketch output and not the code. If you bounce a MIDI ball again, you’ll see that it changes both windows.

MIDI is being sent out of one program; the bouncing ball with MIDI p5 sketch. Another program is listening for MIDI messages on the same port; the random color p5 sketch. Multiple instances of the same program can hop onto one MIDI port. These pairs of MIDI inputs and outputs can sometimes be called a “MIDI Bus”.

The great thing about MIDI Buses is that lots of things can hop on the bus, sending and receiving to their heart’s content. And this is all built into the way that MIDI works. You don’t need any servers to set up or coordinate the messaging. It just works.

Try opening three browser windows of the random color sketch. Four. Five. Fill the screen with small versions of them. Plug in a monitor, drag over another instance of the sketch and full screen it. They’ll all receive the MIDI messages from the ball sketch, all at the same time.



But That's Not All

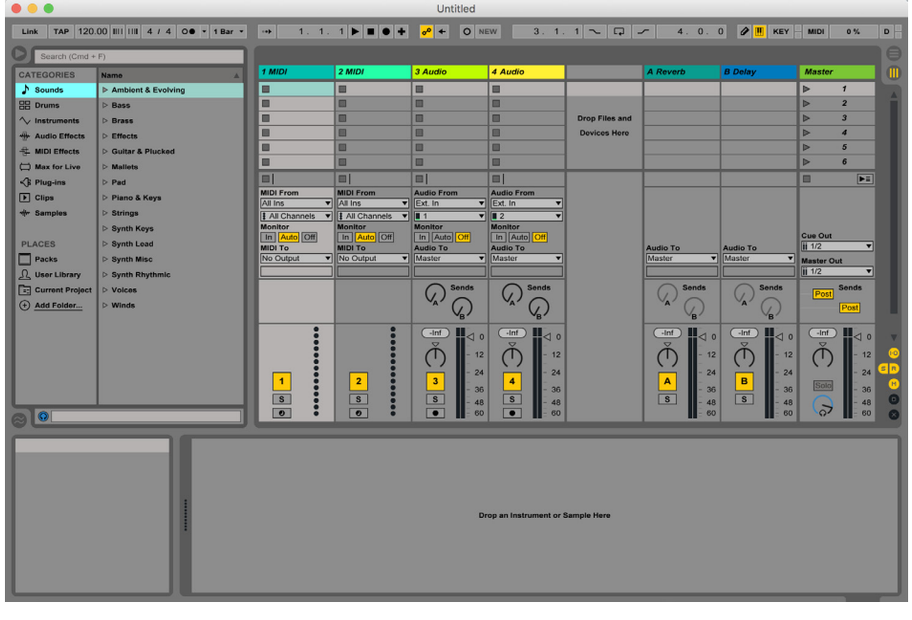
There are all kinds of different programs that can work with MIDI messages. You have just made a browser based interface that can interact with them. Simplistic, perhaps. But certainly a start. Imagine being able to divide a collaboration up into two parts: you work on a visual half while a musician works on the audio half.

Totally possible. But how would we go about testing it before handing it off to your collaborator?

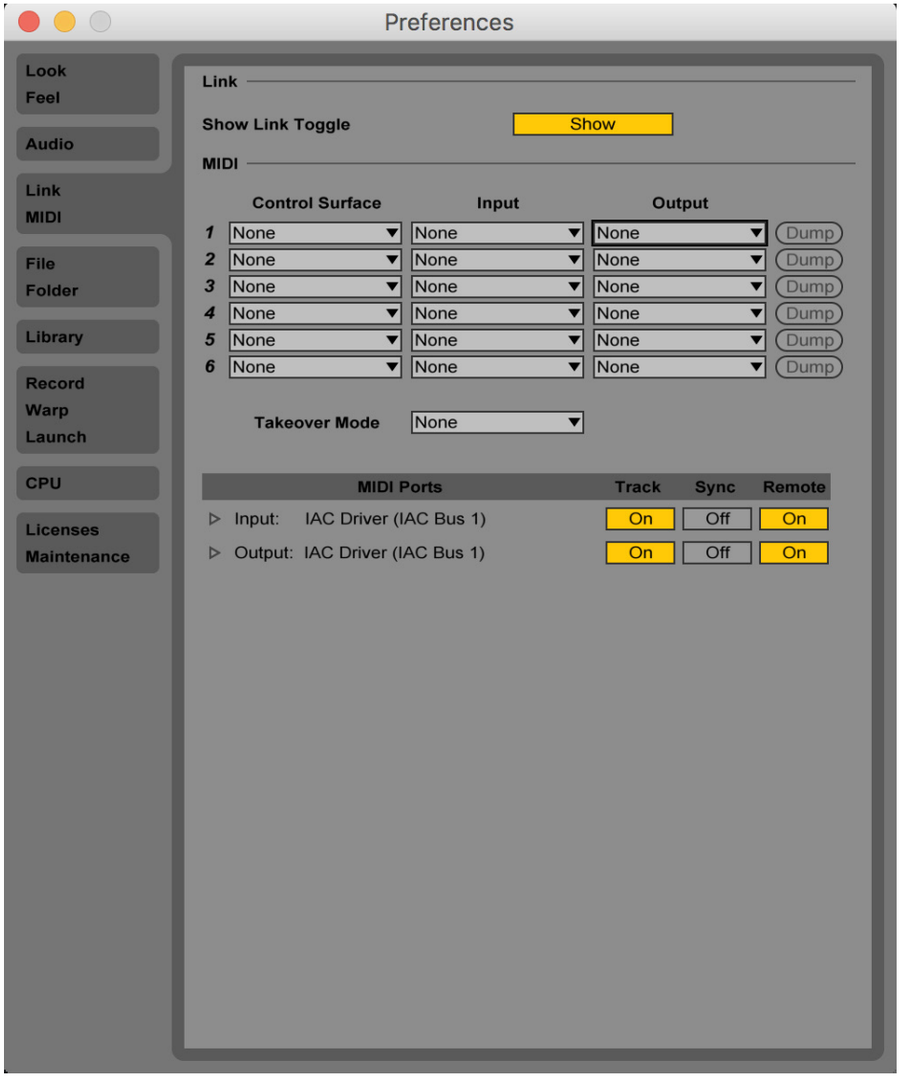
Ableton Live is a popular music making application, and it uses MIDI! You can install it here:

https://www.ableton.com/en/trial/

When you install Ableton and open it, you will be presented with a window that looks like this:

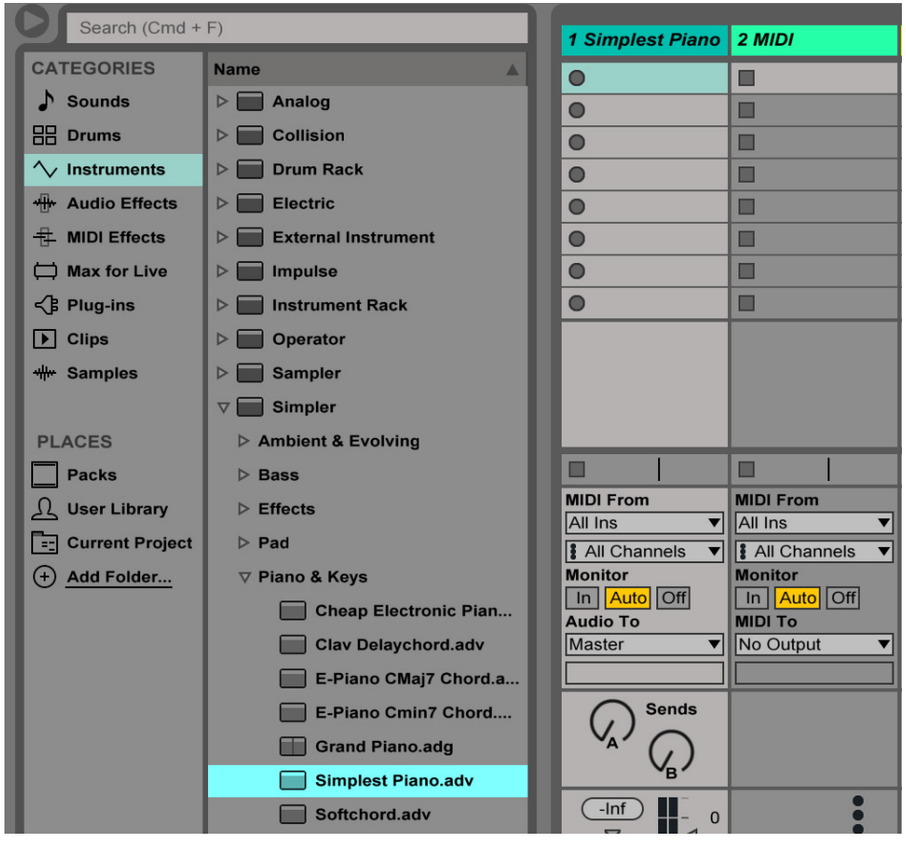


If you then go to File>Preferences, you'll see different settings for how Live works. If you click on the "Link MIDI" tab on the left, you will see the current MIDI setup for the program.

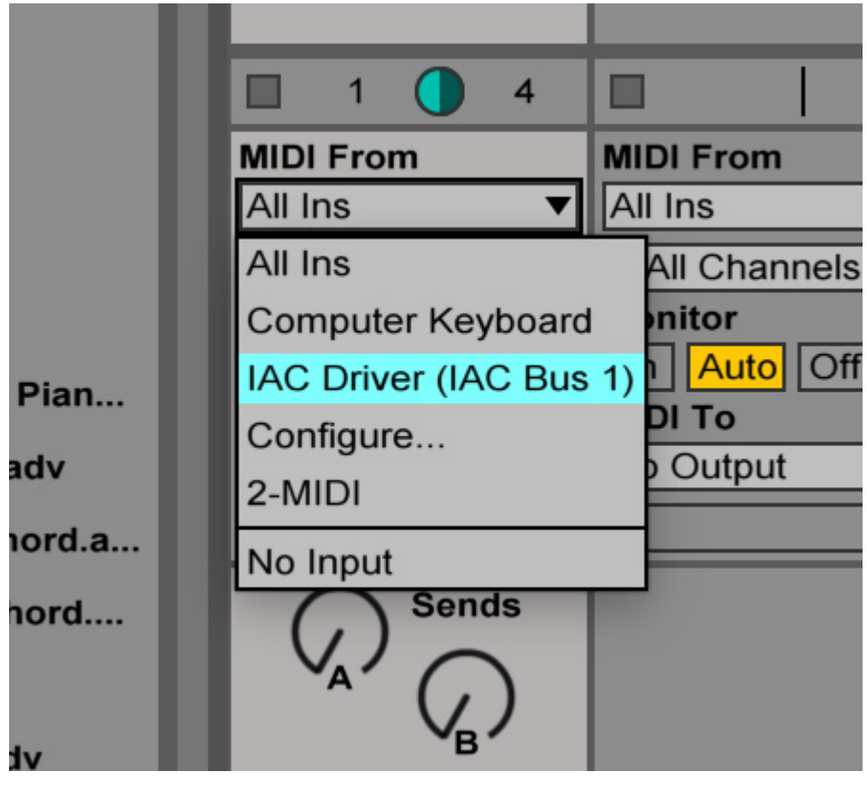


Oh, look! It's the MIDI bus I made! Click "On" for Track and Remote for both Input and Output MIDI Ports. This will allow Live to receive MIDI messages that are sent to this MIDI bus, and also send messages out to that MIDI bus.

Once you have your MIDI bus enabled inside of Ableton, you will want to add a MIDI instrument. If you use the browser on the left hand side of the Ableton screen, you can select Instruments>Simpler>Piano & Keys and find a piano instrument that you like. Click and drag it to the work area to the left, onto the "MIDI 1" instrument slot. Now the slot will be re-named to whatever instrument you chose, like "Simplest Piano".



Now, to get our Simplest Piano to be able to listen to our p5 sketch, we will need to make sure that the specific instrument is listening to the MIDI bus we have made for ourselves. For every MIDI instrument in Ableton, you will see a "MIDI From" section with a drop down menu. On your new piano, choose MIDI From your new MIDI bus (in my case right now, "IAC Bus 1").



Now when you use your original ball bouncing sketch, the MIDI instrument should detect the "bounces". Out of the browser, into Ableton Live. Amazing. But you can see in our ball's bounce function that it is just sending the same note, "C5" over and over again.

Lets replicate the functionality of the pre-made piano sound we were using before.

First, to clear things up let's remove the sound from the p5 sketch. Comment out your play and set volume commands inside of the bounce function.

```
//because we're moving to MIDI, lets
//disable sound playback inside of p5

//play the sound effect when we bounce
//this.ballSFX.play();

//change the volume of the sound
//this.ballSFX.setVolume(this.volume);
```

The way that the sketch was making lower and higher pitched sounds was by affecting the speed of the sound playback. You may have noticed that the lower notes stretched on for longer. Making the left side of the screen play lower tones and the right side play higher tones was accomplished with the following code:

```
this.ballSFX = loadSound('piano.mp3');
...
this.sfxSpeed=this.x;
this.sfxSpeed=map(this.sfxSpeed,0,width,0.01,4);
…
this.ballSFX.rate(this.sfxSpeed);
```

This can be read as, "The rate of the playback of the file is set to a variable called sfxSpeed. This variable is set by determining the ball's X position in the sketch. If the ball is all the way left the rate is 0.01, if it is all the way right the rate is 4. All the X positions inbetween are mapped accordingly."

This means a ball in the middle will be played at a rate of approximately 2, where as a ball one quarter away from the left side of the sketch should play at a rate of approximately 1. This is fine when we are working with numbers. But "C5" isn't a number. How can we map for MIDI values?

Luckily, each specific note value has a corresponding number. The lowest note C0 can also be referred to as note number 0, and the highest note is G10 is 127.

127

We will see this number a lot when we are working with MIDI. Why? A MIDI event can be sent as a byte, which is 8 bits. The way the MIDI standard is implemented, the first bit is used to identify the kind of MIDI message the byte describes. This leaves 7 bits left to describe a value. Considering a binary counting system, 2 to the power of 7 is 128. Meaning the range of a sent MIDI value is 0-127, since we will start at zero instead of one.

You don't need to know all this. There are plenty of great resources online if you want to get into the nitty gritty of how MIDI messages are created at the lowest level. But for our purposes, not necessary. Just didn't want you to think we picked 127 out of thin air. And you'll be seeing it come up repeatedly when working with MIDI.

A new map() for a new territory

Our old playback rate was mapping our ball's position on a scale from 0.01 to 4.0. Now we need a new map for using MIDI values instead.

```
//creating a mapped MIDI note instead of a mapped audio playback rate
this.midiNote=map(this.x,0,width,0,127);
```

And then in place of triggering a playback event with our audio, we want to playback MIDI

```
//because we're moving to MIDI, lets
//disable sound playback inside of p5

//play the sound effect when we bounce
//this.ballSFX.play();

//change the volume of the sound
//this.ballSFX.setVolume(this.volume);


//Setting the MIDI output of the Ball object
//using the this.midiNote number as determined by ball position

//set output
this.output = WebMidi.outputs[0];

//play note
this.output.playNote(this.midiNote);
//this.output.playNote("C5");

//console.log(this.volume);
console.log(this.midiNote);
```

If you now use the ball sketch, you'll hear different notes being played by the piano inside of Ableton. There is a bit of a quirk, however. Might be hard to notice, depending on what kind lovely piano you have loaded up. When we use WebMidi.js's "playNote" function, we are creating a midi "note on" event. But a note on is a whole different beast than a note off. It makes sense, when you think about it. Maybe you want to hold down a piano key shorter or longer, depending on how you want to play.

When we send a lot of note on events without corresponding note off events, sometimes our MIDI enabled software can act a little funny. A note on that stays past its welcome might be called a "stuck" or "hung" note. With the bouncing ball sketch, it didn't really matter. In Ableton, it might result in unpleasant sound or glitches. If you are experiencing a stuck note in Ableton, click the "stop" button at the top of the program.

Stopping and Softness

Sending a corresponding note off message would be a good idea, just to keep things clean. While we're at it, we can set a note velocity (a "volume" of sorts). And that way that WebMidi.js works, will also require us to set our MIDI channel.

```
this.output.playNote(this.midiNote, 1, {duration: 100, velocity: .95});
```

All that work in one line?! WebMidi.js you gotta be kidding me that is amazing. Thank you!

In one line we: Sent a midi note number (this.midiNote, determined by ball position) Set the channel this message is to be sent out on (1) Scheduled a note off event in the future (100 milliseconds after the note on) Set the velocity of the note (on a scale of 0 to 1)

The note off stops the current note from playing, and the velocity describes how hard we are hitting the note. This affects the volume of playback.

Volume of playback, volume of playback… hmmmm. Aren't we already working with that for our old sound file? We are! What is that again?

```
this.volume=map(this.lifespan,0,600,0,1);
```

It is already being mapped on a scale of zero to one. Nice. Lets just drop it in for our velocity value:

```
this.output.playNote(this.midiNote, 1, {duration: 100, velocity: this.volume});
```

You should now be able to hear pianos playing not only while bouncing, but also the notes will get softer and softer as the bouncing ball disappears.

If you have made it this far, congrats! I told you there'd be some fairly boring stuff to get through. All those preferences, configuration screens, setup windows. Yeesh! So many tedious little things just to get to this point.

But now if you have this working, you'll be able to do a lot of interesting things with a bunch of different programs. Thank you for being patient with me. Before we dive into more things we can do, you should take a break. Try selecting some new instruments from the left hand side of the Ableton Live window. Pianos, so traditional! Did you see there are tons of crazy synthesizers inside of this program? So cool. They sound great. Load up some new instruments in your first MIDI track and then play with the bouncing ball sketch.

Jam out for a bit. You've made an instrument, after all. When you try new sounds, take note of what works and what doesn't. Why are certain instruments not as satisfying for use with the bouncing ball sketch as other ones are?

In fact,

now that I think of it,

How about we hold up for a second.

Later on, I want to outline how to implement the other parts of the MIDI protocol I talked about: CC messages and using different channels. I'd even like to push you a little bit by introducing some new programs and languages: Processing and Max MSP. But what about using what we

have? What about our bouncing ball? What about the Note On and Off messages? And don't forget Velocity!

It is really easy to get caught up in ALL the things that a program or language can do. And why not? If you want to learn it, you want to know the whole thing so you can use it to your advantage. How are you going to make something awesome if you aren't leveraging all of these amazing features?

It is that last sentence where I have a bit of a problem. There are plenty of cool and compelling things you can do with very simple features of programming languages and programs. And afterall, we're not just programmers- we're artists! We're designers. "Creatives". Constraints are great for creating new work. How many amazing images have you seen that have been drawn just with charcoal? What about Bauhaus? *WHAT ABOUT DEITER RAMS?!*

Sure, there is more to learn. But what if what you have right now is enough for you to make your perfectly elegant minimalist browser based MIDI enabled debut in the world of creative coding? I know, I know. You aren't ready for that. You wouldn't know what to make. You don't even know about CC messages yet! (It's not that hard, I promise)

It is true that there is more to learn. But it isn't true that you know nothing.

I've aimed this tutorial series at people who have already done some introductory programming work. When was the last time you thought about the things you made when going through your old beginner material? The bouncing balls, the buttons, the sliders. Color, shape, time. Letters, words, weights. Mice, microphones, webcams. Image, video, sound. Perhaps an API call here and there. Or even linking it to hardware, like an Arduino.

With what you have learned in previous Javascript classes or books, you can now take any one of those things you have made and have it send or receive a note or notes, knowing the difference between on and off, each with varying velocity values. One p5 sketch in one browser window can send to multiple other browser windows. Multiple browser windows can each send notes to a single window. And then all of that, back and forth, in and out, can also include Ableton Live.

When was the last time you played with your old programming projects? You left them in a folder, somewhere. What did you name it? "Projects"? "Programming?" How deep down inside of the Documents folder did you hide them? Or they are backed up on some cloud storage somewhere. Google Drive. Or Dropbox. Did you ever share it with someone? Or did you keep it private? Perhaps they are still sitting on an online editor. What was the login? The password? Maybe it will autocomplete when you go back. Oh, these small miracles.

All that work you did. Sure, a lot of example code. But you worked through it, its your example code. You changed the colors, the sizes, the speed. It didn't go to waste, and it wasn't "just" tutorial stuff. It has been waiting, diligently, to be picked back up again. Don't worry, your programs weren't loney. They don't have feelings. But they've been ready to be picked back up this whole time. Ready to be re-worked, tweaked, and incorporated into something new.

The mouse X, Y tutorials. The "onClick()" function examples. They all can have MIDI inserted into them.

I propose we take a technical break, and pick up an artistic task.

Go into your back catalog of work, or re-create an example from one of your previous programming tutorials, and "MIDI"-fy it. Make a MIDI version of and old piece. Incorporate the WebMidi.js library and the Note and Velocity MIDI messages in the way that I've done to the bouncing ball sketch. This can be a good technical review for you to do on your own, without explicit guidance:

- Add the library
- Initialize it in your program's setup
- Setup to listen for incoming MIDI notes, and/or
- Send MIDI notes

There will be other useful programming life lessons, as well. You'll look through your old code and wonder what you were thinking. How does this thing even run, anyways? Get reacquainted with your program, and this time write detailed code comments before you forget again. Your future you will thank your past you.

But even more so, this is a chance to meditate on simple interactions and concepts for a piece. They could be complete on their own, or perhaps satisfying fodder for something bigger. How many different ways can you use MIDI?

What is obvious? What is the opposite of that? What is weird or nonsense? What is best suited as a "sending" program and what is best suited as a "receive"? What could work as both? Do you like using Ableton Live with these? Do you like the music it makes? Record that and listen to it for a few days.

Instead of plowing through the rest of the MIDI specification and throwing more code at you, how about we take some days to think of how we can use what we already have? Every day this week, come up with a new idea for a small MIDI program idea. Simple stuff that you can jot down in your smartphone's note app. Right now we have one: "Ball bounces, makes sound and changes other window's color". So you can't take that one. What else can be done?

You don't always have the motivation to program. So I highly encourage this method of writing down strange, vague project ideas into your smartphone. Also, bizarre diagrams with weird labels, penciled inside of a small notebook is also a very good look. When you actually do have time to code a bit, after work, after the laundry, and after doing your dishes… you'll have something to start on. Instead of wondering, "...what do I do now?" You'll open your phone and see your half-baked prompt from Wednesday: "Magenta Triangles - growing / tones"

And with any luck, you won't remember what you even meant at the time. But you'll somehow be motivated with a new idea, nonetheless.

GOING

FORWARD

Interlude

Over the past few months, you have started to get into a steady rhythm with your work. Not that everything is easy for you, or that your schedule is reliable. Sometimes you have the motivation to do things during your spare time, other times not so much so. But the down times, the "unproductive weekends", don't seem to bother you as much as they once did.

You have a thing or two that you have been working on. Plucking away at the code with the occasional binge session (sometimes planned, sometimes not), you are steadily approaching a possible end goal. Or at least, an "end" enough to where you may want to show it to other people.

The "ideas" section of your notes app has exploded. Your friends are now used to you abruptly snatching ideas from thin air. It doesn't even interrupt conversational flow anymore. You will never, ever get to complete all of these ideas. It now resembles more of a library of your mind. You will pull something off the shelf when you need it, or when you are bored. Some recent entries include:

"Gold foil and dirt, random number generator. USB HID device, electrodes"

"Zork, but on the Arduino via the serial port. Password manager? Geocaching?"

"Computer info with medical graphic design: HD space is cholesterol, electricity usage is EKG, etc"

And finally, you have gained a new habit of reading about all of this technology you are working with. It started with bouts of restlessness. Should you be switching to another programming language? New environment? Different platform? Github pages, blog announcements, Reddit comment threads. But now, where there was once restlessness there is now a straightforward curiosity. Little light bulbs go off in your head.

You start a separate file in your notes with URLs to go back to. Someone made a visual programming environment for Processing code. There is a different OSC library that someone recommended in a forum. This Arduino shield can play MP3s and WAV files off of an SD card, but it can't play *multiple* sound files simultaneously.

As a result, you leave browser tabs open. Lots of them. Some of them just simple Google searches to follow up on later: "polyphonic mp3 playback arduino", "fading EL wire", "control Canon camera from Processing". The wikipedia page for "Visual programming language" has been open for the better part of a month.

It is all research, even if you don't feel like it is. Now it is just part of your normal web surfing habits. On your lunch break, you go to obscure electronics blogs. You read the release notes to a new version of a library you've never used. You spend 15 minutes investigating detented potentiometers. Your knowledge usually follows from what your projects need at that moment, starting as desperately utilitarian. But now you are starting to have an instinct in regards to the specific domain you have made for yourself. It isn't expertise per se, though you notice these thoughts are saving you time, preventing headaches and dead end google searches. "This thing doesn't work with that thing." "Yeah, that looks powerful but it isn't documented well." "Seems like a great program, but it doesn't work on this operating system."

Internal Pull-up Resistors

One of these fun little tricks you have developed lately is using the internal pull-up resistors in your Arduino. Kind of odd to call it a “trick”, since it really is just a feature waiting to be used. But it feels kind of like you are getting away with something. One day, while gliding over the Arduino examples for inspiration, you were reminded of their presence. Something clicked.

Internal pull-up resistors are resistors that are built into some of the input/output pins on an Arduino. You still have a little trouble with all this “physical computing” stuff, and are constantly re-reading the instructions for basic things you’ve already done before. But as far as you can remember, a pull-up or pull-down resistor is used in a circuit in order to make sure that there is either steady voltage (pull-up to positive) or no voltage (pull-down to ground) going to a specific component. They are commonly used in microcontrollers to make sure that your input pins don’t “float” between high/low readings. You see them often with things like buttons or switches.

The *internal* part just means that they are built into the Arduino itself. You need to specify when you want to use them in the code you upload to the Arduino.

Instead of:

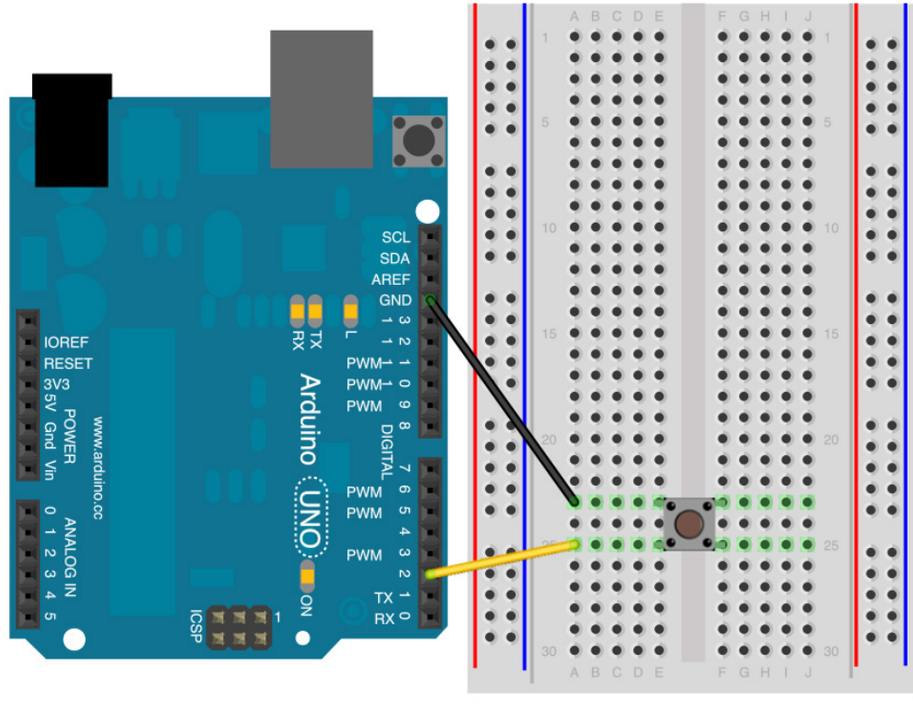
```
//Normally doing an input for a button like this
pinMode(pinNumber, INPUT);

//you would instead say
pinMode(pinNumber, INPUT_PULLUP);
```

And then when taking readings, you see that the “off” state of an input is now 1 instead of 0, and the “on” is 0 instead of 1.

This didn’t strike you as particularly interesting the first time you came across it. Then over time, you would switch from projects only on the computer to projects using your Arduino. You had to go through all the intro material again: Which resistor? What colors are those? How do I wire up a button again? I’ve done this before, why can’t I remember?

Eventually, you figured it out. But then your affair with “physical computing” was distracted by a different project. Maybe a Twitter bot, or a data visualization about fonts. And sure enough, when you came back to making a physical thing again, all memories of your push button example had been purged. Pushed out by fresh knowledge of API keys and kerning.

On one of these occasions, you happened to come across the “Input Pullup Serial” example in the Arduino IDE and it’s webpage on the Arduino site:



https://www.arduino.cc/en/Tutorial/InputPullupSerial

It's just the Arduino, two wires and a button. It occurs to you that you could probably just strike the exposed ends of the wires together to test if it works. The image in your head reminds you of those scenes in movies where a car thief tinkers under the steering wheel, and rubs two frayed cords together as some sparks fly and the engine roars to life.

This has been a huge revelation for you.

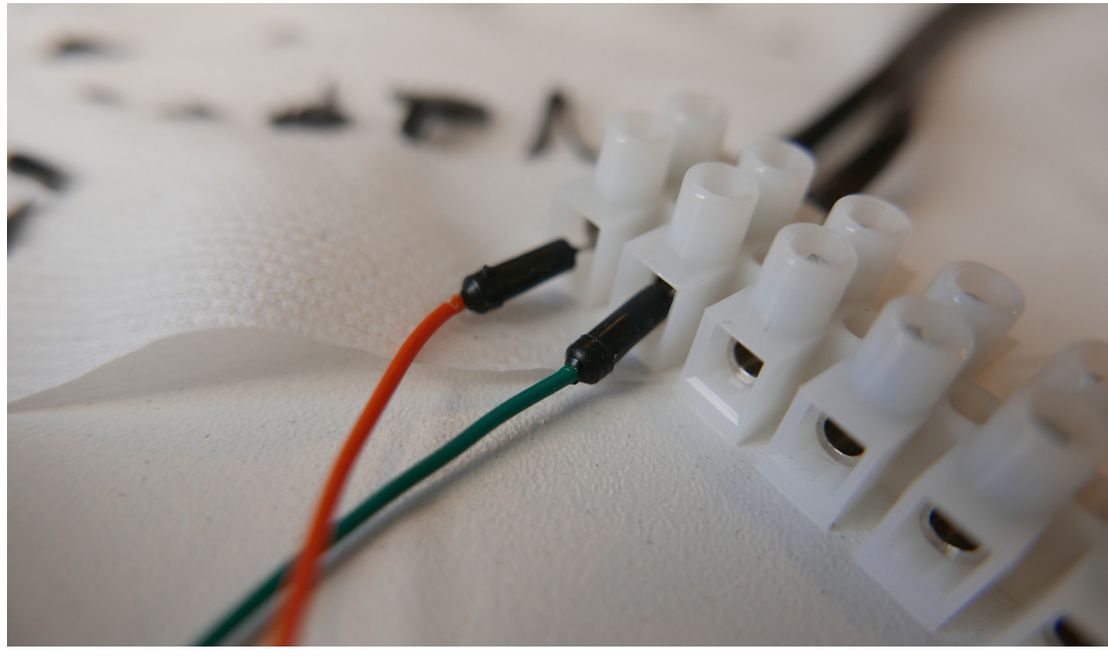
Instead of thinking about the bread board or how things will go into it, you are thinking about the possibilities of interaction. Afterall, a button is just a way to temporarily bridge the gap between two pieces of conductive material.

Now you are assaulting different items in your house with two jumper cables attached to an Arduino, creating "buttons" out of anything nice enough to give you a little conductivity. You aren't thinking about the code, or the circuit, or the math that could be involved with any of it. That all fades away as you enter an extended flow state, considering all of the ways you can bridge a gap, make a connection.

You start looking at copper tape online, considering the options. Some are conductive on both sides, where others are only conductive on one. When a friend asks you why you have multiple rolls of seemingly identical copper tape on your Amazon wishlist, you point out that they come in different widths.

Terminal blocks become a vital tool. On one side, your Arduino and two jumper cables that are screwed securely into the terminal block. On the other side are wires. Oh, the wires. First it was a logical necessity: "...what if I need to connect something farther away than the longest jumper cable can reach?"

But then it turned into a deeper appreciation. Sure, you could find extreme lengths of wire at cheap prices online. You stripped the ends and screwed them into the other side of the terminal block, and were delighted that your pull-up resistor Arduino program worked just the same.



jumper wires in terminal block

But you wondered about the wire. The color, the thickness, the rigidity. How easy was it to strip? Different variants were found online and ordered for two day delivery. Now you know the difference between stranded and solid core wire. You quickly became a budding wire and cable connoisseur, weighing the character of different cables in your hand when you noticed them. Your headphones, the charger for your laptop, the extension cord on the floor.

Then the realization that certain kinds of cables were just bundles of wires. Your headphones (a wire each for left, right, and ground), ethernet cables (8 wires), hdmi (19). And today you have arrived at the logical conclusion. There are tons of old, broken or simply unused cables out there waiting to be repurposed.

old wires waiting to get thrown away

You used to look at places like these with confusion, perhaps a bit of disgust. Why keep any of this stuff? What is it about dirty, sad electronics? The inhumanness of technology combined with the grime of decaying history. The smell of dust and old plastic. Today is different, though. You want to dive in, even if you still plan on washing your hands afterwards.

You start pulling, picking, untangling. Cables are evaluated (VGA video cable, 15 wires), but you aren't even sure what you are looking for.

You aren't even sure what you are looking for.

Why are you doing this again? This started as a way to make things easier for yourself. Now you're wire obsessed, digging through old egg crates at thrift stores and asking your roommates not to throw out their broken phone chargers. You got distracted by wires, and lost the thread.

You take a moment. You think about buttons. Interaction. Touching. Making a connection. What can you do with the simplest examples you started out with? Maybe one day you'll do something fancy, learn more, be so good that you are able to properly explain a pull-up resistor to someone at a cocktail party. But what can you do today? Now?

Make a connection. Complete a loop. Bring it together.

You stare at the cable bin.

You breathe deep.

Dust. Old plastic

You bring it to the bathroom to wash it. You pump the handsoap a few times and slather the cable in suds. You do a few more hits of the soap, just to be sure. Faint scent of lavender fills the room.

a dense nest

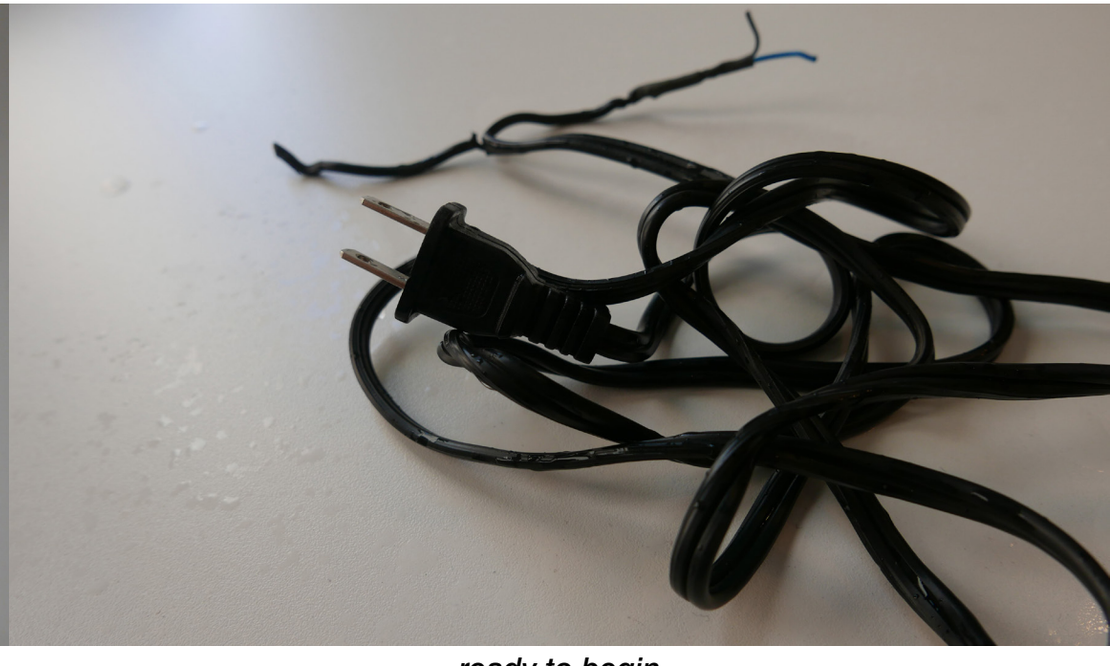
a rescued cable

You reach in. It takes you awhile, but you finally find one. A black, two prong power cable. There is some kind of sandy, light colored dirt on it. A decent amount of length, without being inconveniently long.

This is it, it is your cable now.

Hot water, starting to steam. Enough of the cable occludes the drain for the water to backup, miniature mountains of bubbles growing slowly. The white noise of the running water soothes you, a quasi-trance state. You put your hands straight through bubble mountain, finding the cable underneath the soap suds. You wash the cable with your hands, and each hand washes the other.

a calming ritual



ready to begin

Rinse.

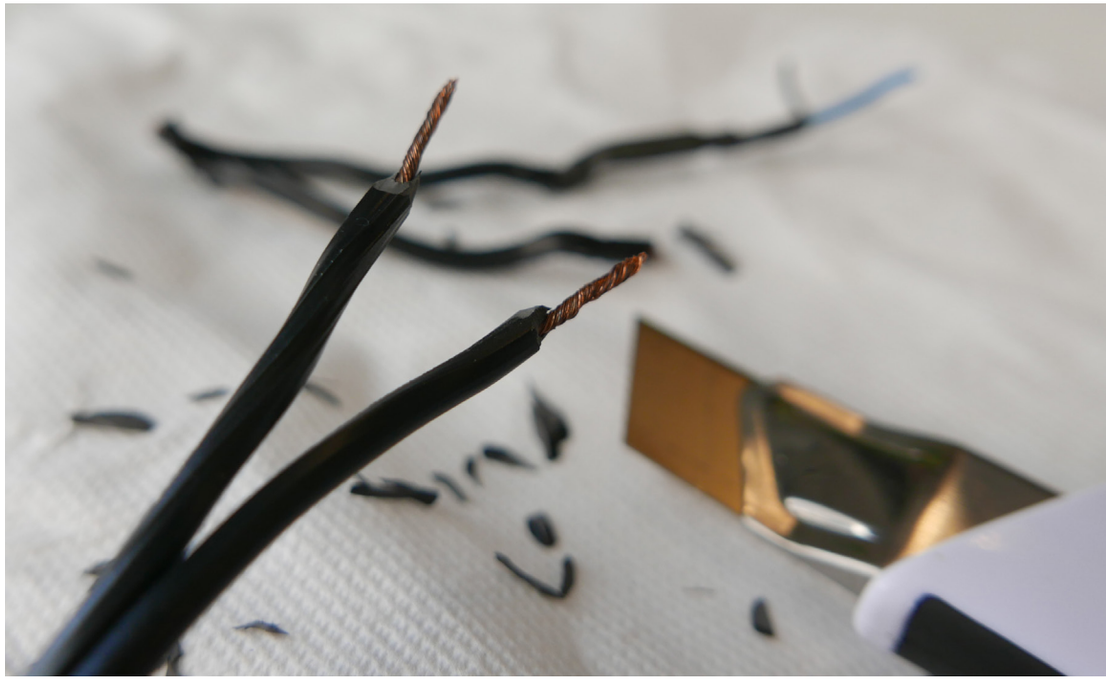
Brief drying with a paper towel, but the cable still drips as you walk it to your workstation.

You take account of everything you need:

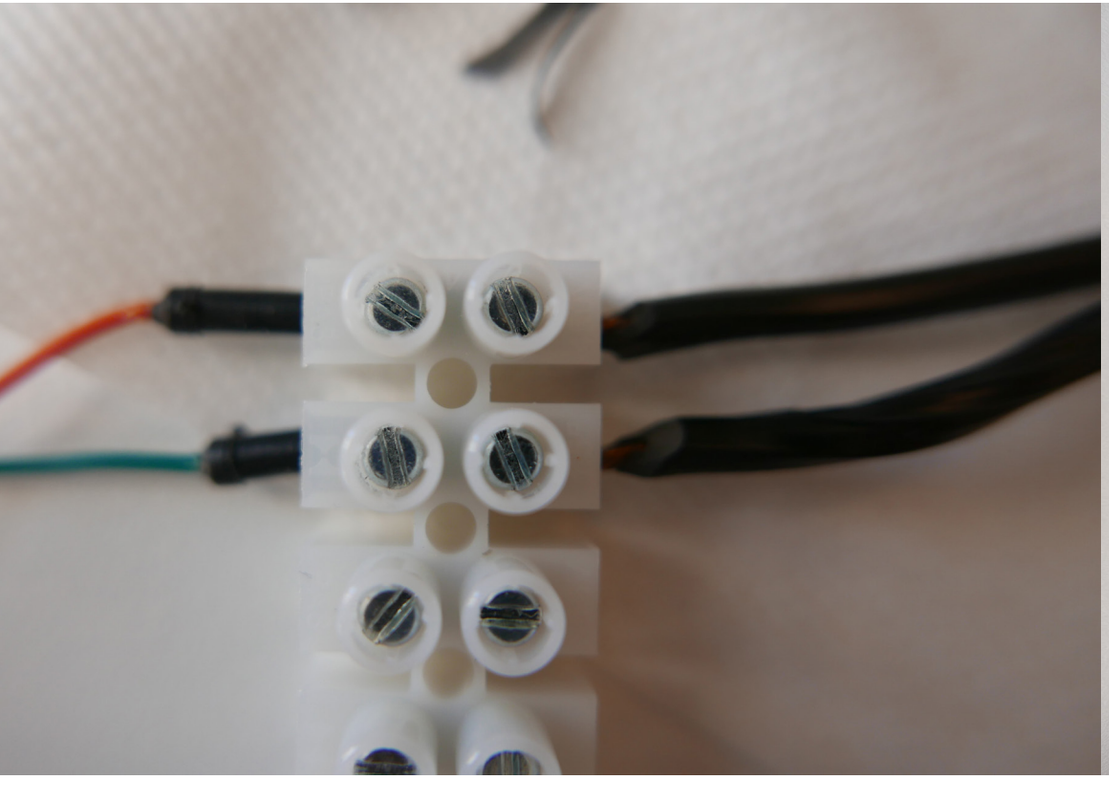
- Your computer
- Arduino, with USB cable
- Three jumper cables
- Terminal block
- Precision flat head screw driver
- A drying, two prong power cable

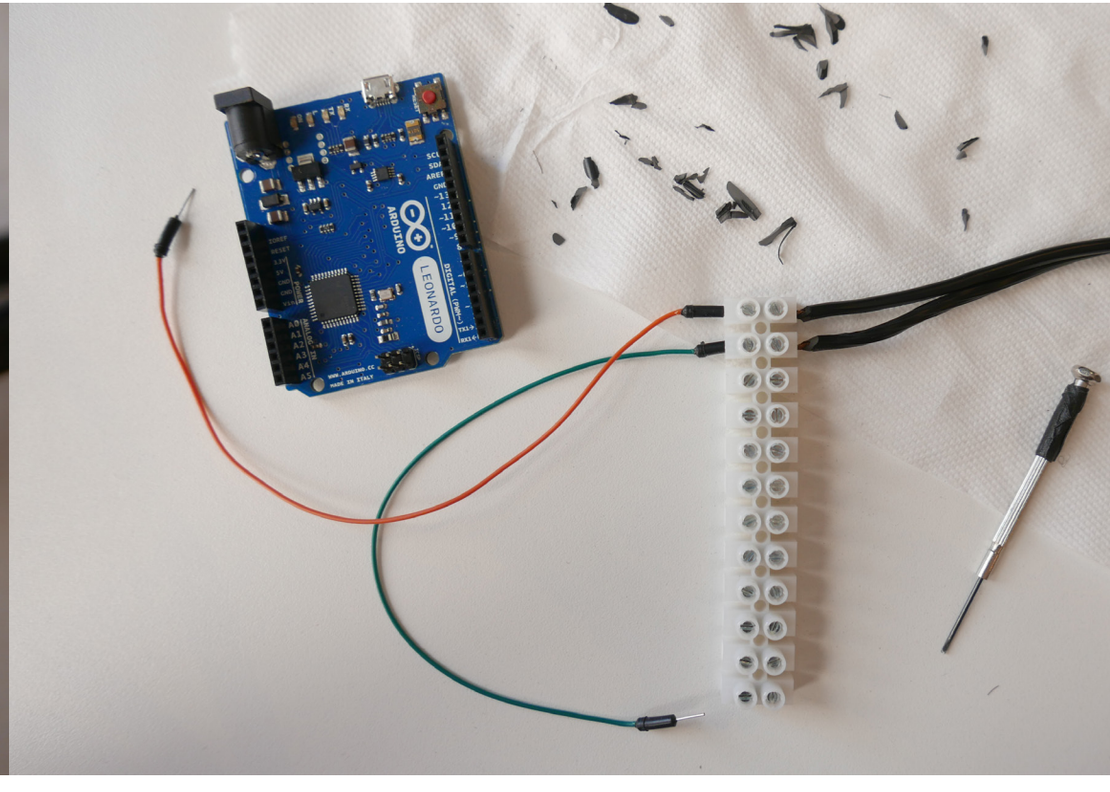
You look at the diagram for the internal pull-up resistor example. One jumper wire in the ground pin, one jumper wire in pin 2. Then for your version of this circuit, the other ends are inserted into the terminal block, secured by tightening the screws.

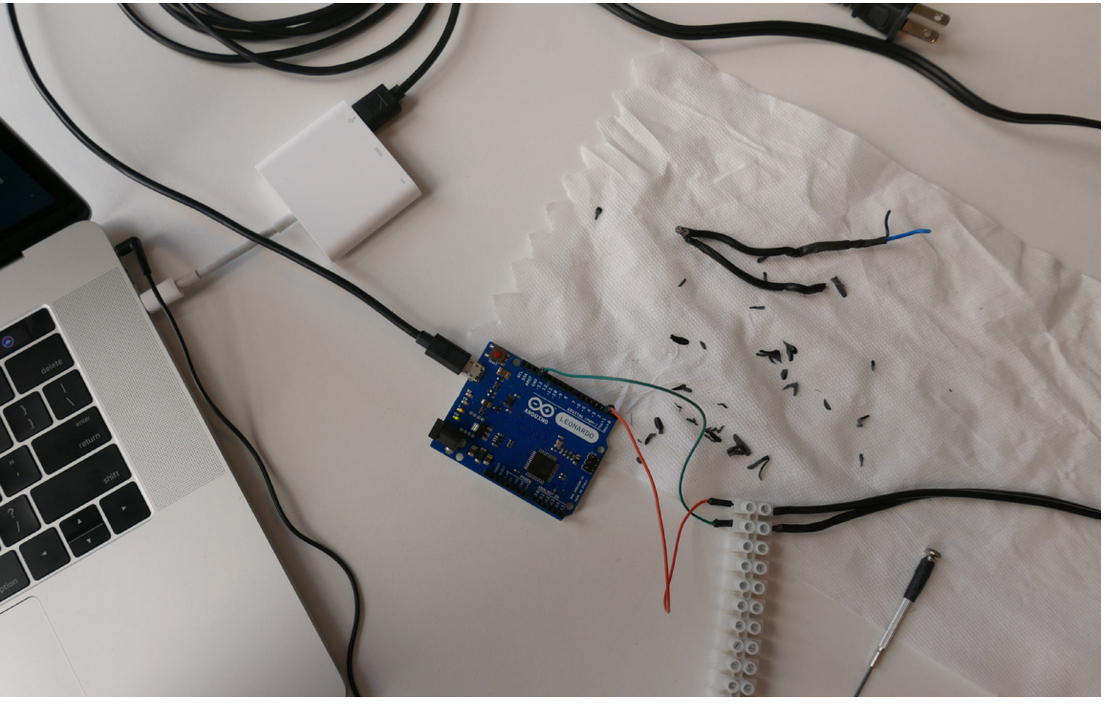
You could use wirestrippers for your salvaged power cable. But since it is only two relatively thick wires, you are content to grab a box cutter and whittle away at the ends. It reminds you of your grandpa, though he didn't seem to keep up with wood carving for very long. You wonder where those little sculptures went.



Inside are lots of small, thin copper wires. They had spread open as you were carving away the wire insulation. You take them in between your thumb and your forefinger, and twist. Tight, tighter, tight. The wires are strong enough to hold the shape. You insert the two pieces of exposed power cable wire into the screw terminals that match the two jumper cables. You tighten the screws on the power cable side. The two metal screw terminals each join one jumper wire to one cable power wire.




ARDUINO
LEONARDO



You plug in your Arduino, load up the Arduino IDE software. You load up the example code.

File>Examples>2.Digital>DigitalInputPullup.

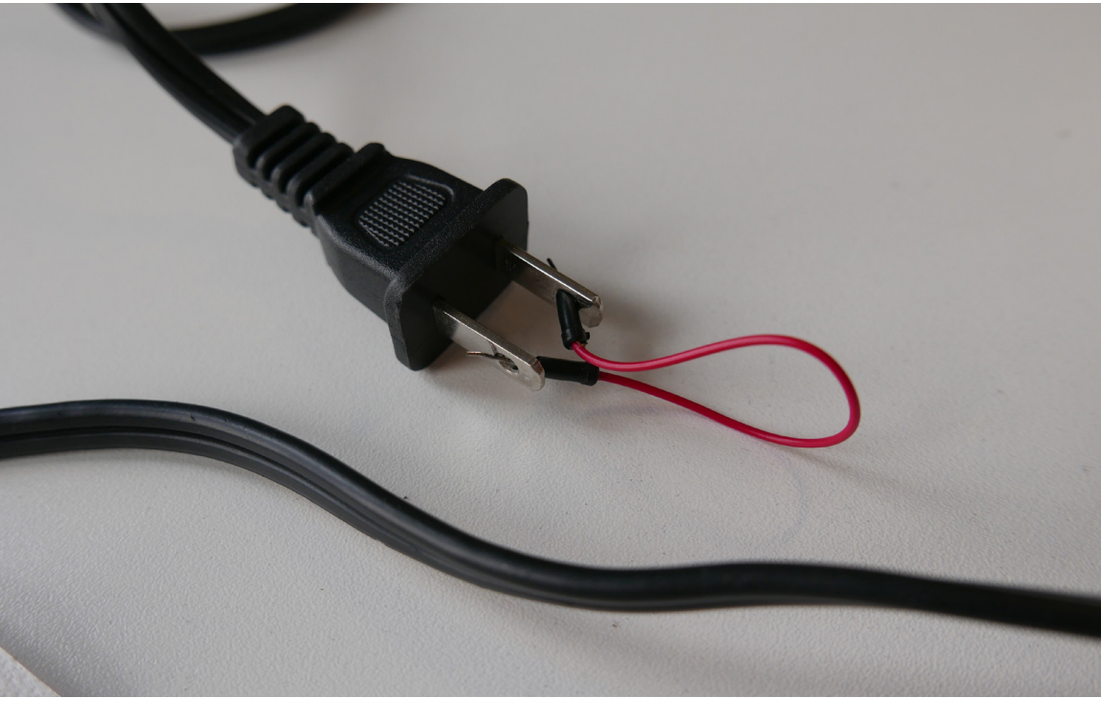
You upload the code to your board. It doesn't work. You double check the standard stuff.

Tools>Board is correct.

Tools>Port… is not selected. Who knows why? You enable the correct port and then upload again. This time, nothing complains.

You click the Serial Monitor icon in the upper right hand corner of the window. A new window opens up. After a brief moment, in which you think things are broken again, a stream of "1"s pour down. It works. This is the resting state of the program.

You take your third jumper wire, and wedge the two pins into the circles on the prongs of the plug. It creates a neat little bow.



Power out of pin 2,

through the jumper cable,

across the terminal,

flowing up one wire inside the power cable,

crossing over your little bow jumper cable,

into the other wire of the power cable,

across the other terminal,

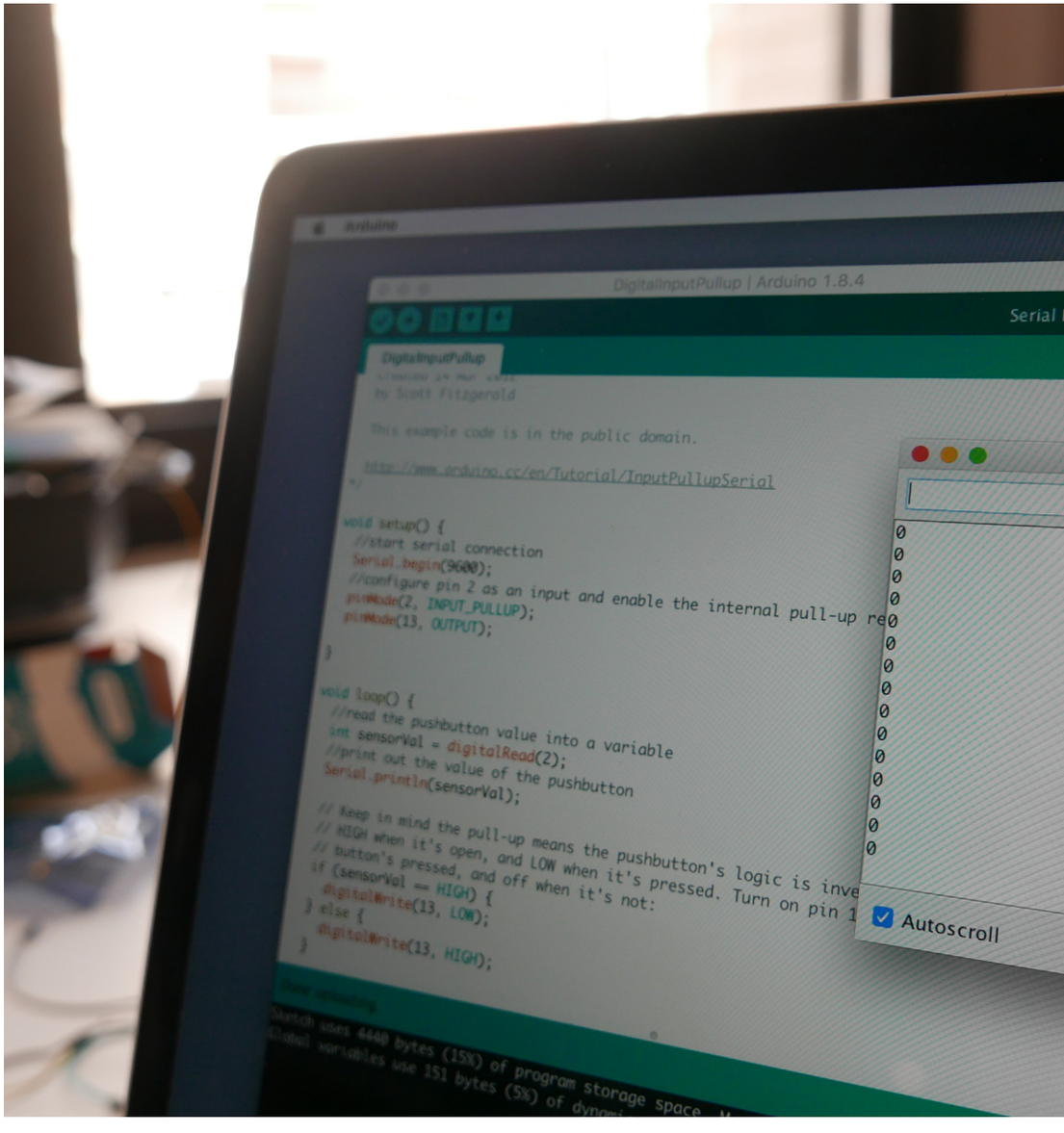
through the other jumper cable,

into ground,

completing the circuit.

The Serial Monitor shows a stream of 0s. You undo the bridging jumper cable and tap the ends to the power plugs. The Serial Monitor dances between 1s and 0s, in time with your beat.

It works. You take a moment, look out the window, look back at the mess you've made. Plastic shavings. Crumpled, half-damp paper towels. It's not that bad. The stray drops of water have dried by now.



Holding up the plug to eye level, the metal shines as you twist it in the soft daylight. You consider the two prongs. Close, and waiting. How many ways can you bring them together?

You stick out your tongue.

You put the metal prongs on your tongue.

Five volts doesn't do any harm, by the way. In fact, you don't feel anything at all. You look at the Serial Monitor. Still a stream of 1s. This isn't working. If there was a connection it would read 0s. Something is wrong. You briefly ponder if your tongue is abnormal, somehow. What a way to find out. "Doctor, its my tongue. It's not conductive!"

You release the plug, holster your tongue. You test the plug with the third jumper cable again, see the successful 0s on the Serial Monitor. So the circuit still works. You breathe, you think.

You stick out your tongue again. It is dry from your previous attempt, your mouth salivates to compensate. You press the metal prongs harder onto your tongue this time, not shy about holding them tight against your flesh.

The Serial Monitor reads 0s. You are a button.

It doesn't feel like anything. How can that be? You thought when you succeeded there would be some kind of sensation. Instead, things just work. Though, when you close your eyes and keep holding... you begin to wonder if maybe you feel something.

You know that the electrons are pursuing the path of least resistance, making a break for it across the dimple in the middle of your tongue. But you wonder how true that is. Is the pooled saliva mildly electrifying the entirety your mouth? Could a stray electron jump the gap somewhere, ride a nerve ending back up into your brain?

You don't have to know all the specifics for it to work. You knew enough to make it happen, to make the connection.

You imagine a jumble of electrons swimming around in your head, bouncing around like popcorn. Their density and frenetic pace increasing into a dense buzz of chaos. They form shapes, turn into visions.

Where did the cable come from? How many lives did it lead? How much did you do to get to this point? Learning, re-learning, experimenting. Without you, the cable would still be in the bin. Gathering dust. Without the cable, you would still be separate from your work, your tools. You've mashed it all up, now. What is plugged into what? You're in it, a part of it.

Who will you talk to with your new electric mouth? Will your words be more sharp and clear? Will your next kiss spark like a static shock?

It's changed your brain, changed the way you look at things, changed the way you think.

How are you going to explain this one? Once you release the cable, cut off those electrons, walk away. You might not need to say anything at first. You're alone. But eventually, at some point, someone will be the first one to ask you those familiar old questions at some social engagement. "What do you do?" "What have you been working on?" "Productive weekend?"

Sure, you could play it straight, you suppose.

“I make things with technology. I like to call it art, a lot of the time. I’ve been licking cables lately. Yes, yes, I made a big breakthrough on Saturday.”

This isn’t normal. Follow up questions will be asked.
And you aren’t even sure why you did it, what it could be, and all those other things to consider.

“Does it hurt?”

“No, its only five volts.”

“So… what does it feel like?”

Yes. Yes, that it was they will ask. What does it feel like?

The plug is still against your tongue. As you consider the question, you feel a wave of calm followed by a surge of energy. A kind of giddiness, even. It feels good. A confirmed instinct. A successful improvisation. A flow state. A discovery. A path forward. Motivation. These moments are hard earned, and aren’t permanent. But they are a kind of magic you’ve always wanted. There will be more of them, it’s just hard to see that sometimes. Or feel it.

But the computer, the five volts, the wire?
What does *that* feel like?

It feels like metal.
It looks like madness.
It tastes like hand soap.