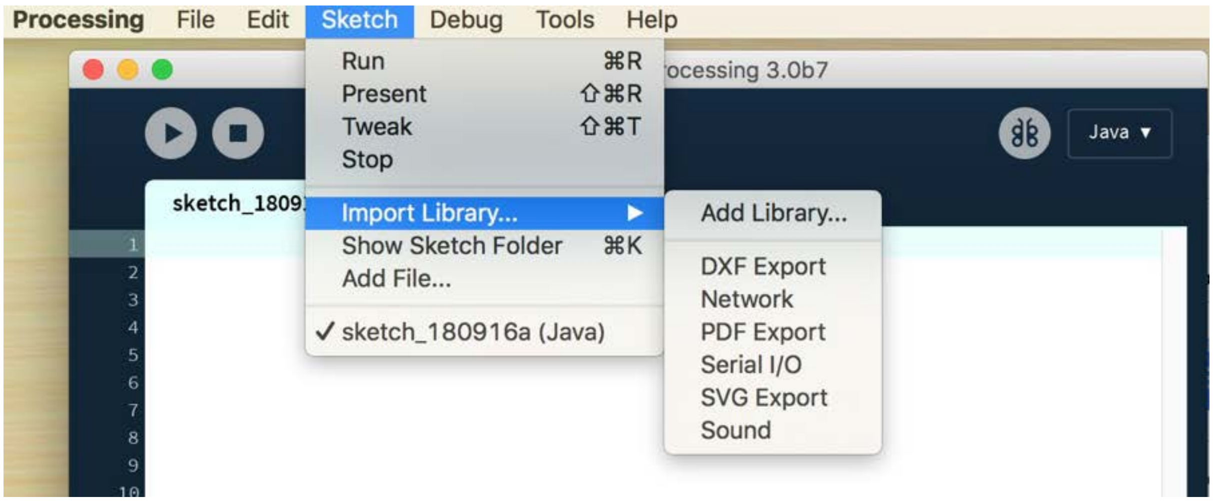
Week 8: p5.js and libraries

It's time for a sketch UPGRADE! We are ready to start adding libraries - which means lots more cool things! Libraries are a collection of additional functions that we can join onto our sketch to make our sketch more awesome.

Today we'll be focusing on two main things you may want to play with: Microphone Input and Sound output .



Importing a library into Processing is easy peasy: Sketch > Import Library > Add Library
Once you become a bit more comfortable with libraries, you can also branch out to Contributed Libraries that other people have made.

Although the libraries in Processing work, it's a lot easier to handle sound and video if we switch over to p5.js for this instance. This is because p5.js already comes with built in functions and libraries for this. **This class has introduced you to a myriad of Processing-related bits and pieces so here is a summary(in timeline order):**

Processing	Processing.js	openProcessing	p5.js
Created in 2001, as a language for learning how to code within the context of the visual arts.	Created in 2008 and has since been sidelined(works but has noone to maintain or improve it)	Created in 2009 as a method of sharing your sketches and creating an online Processing community with visuals!	The newest of the bunch, released in 2014 and has the most work and effort going into it.
Desktop	Web	Web	Web
Programming language based on Java. This is the OG we know and love.	Made to run Processing sketches in the browser. Is also used as a base to run Processing code inside of openProcessing.	Online community where you submit your sketches to a class and can upload and share your sketches	p5.js is a Javascript library based on Processing.

There is a time and place for all these things and as you start to build out your sketch, you can investigate what is appropriate especially in regards to Processing vs. p5.js.

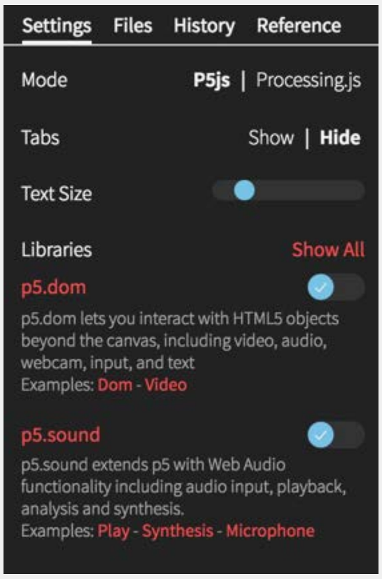
Introducing p5.js

p5.js

p5.js is a Javascript library based on Processing which means, it’s way more friendly when you’re trying to do things in the browser because Javascript is one of the core technologies of the big, bad, world wide web.

It is also because openProcessing leans on “Processing.js” (see above) to render our Processing code in the browser and *this* library is no longer being maintained or worked on. So you would’ve noticed some incompatibilities. Most of the things we will cover in class will work across both Processing and p5.js but when they don’t, they are more likely to work with a p5.js/openProcessing combo.

So let’s first make the switch to p5.js in openProcessing.

	**1. Switch modes to P5js(this is generally the default)** **2. Switch on the p5.dom library** **3. Switch on the p5.sound library**

Settings Files History Reference

Mode **P5js** | Processing.js

Tabs Show | **Hide**

Text Size

Libraries Show All

p5.dom ✓

p5.dom lets you interact with HTML5 objects beyond the canvas, including video, audio, webcam, input, and text
Examples: Dom - Video

p5.sound ✓

p5.sound extends p5 with Web Audio functionality including audio input, playback, analysis and synthesis.
Examples: Play - Synthesis - Microphone

For the rest of this worksheet, we will be writing Javascript code using p5.js and you will see there are a few differences but a lot of similarities.

Making the first move: Processing to p5.js

As p5.js is based on Processing, it is very similar minus a couple of small things. You still have all your handy functions like `rect()` and `ellipse()` and `keyPressed()` but we will have to change a couple of things.

1. Functions are simply “function” followed by its name.

Processing	**p5.js**

`void setup(){`
`}`
 	
`void draw(){`
`}`
	`function setup(){`
	`}`
	
	`function draw(){`
	`}`

2. `createCanvas(w,h)` replaces `size(w,h).`

Processing	**p5.js**

`void setup(){`
`  size(900, 900);`
`}`
 	
`void draw(){`
`}`
	`function setup(){`
	`  createCanvas(900,900);`
	`}`
	
	`function draw(){`
	`}`

3. Variables are declared as `var` no matter their data type. However, always be sure to keep track of what each variable’s data type is because this is still important.

Processing	**p5.js**

`int number = 10;`
`color[] col = new color[10];`
 	
`void setup(){`
`  size(900, 900);`
`}`
 	
`void draw(){`
`  for (var i = 0; i < col.length; i++){`
`  }`
`}`
	`var number = 10;`
	`var col = [];`
	`var cLength = 10;`
	
	`function setup(){`
	`  createCanvas(900,900);`
	`}`
	
	`function draw(){`
	`  for (var i = 0; i < cLength; i++){`
	`  }`
	`}`

4. Sometimes things don’t load in order for Javascript so you’ll need to start using the function preload for preloading files before your sketch runs.

Processing	p5.js

`PImage img;`
 	
`void setup(){`
`  img = loadImage("file.jpg");`
`  size(900, 900);`
`}`
 	
`void draw(){`
`  image(img, 0, 0);`
`}`
	`var img;`
	
	`function preload(){`
	`  img = loadImage("file.jpg");`
	`}`
	
	`function setup(){`
	`  size(900, 900);`
	`}`
	
	`function draw(){`
	`  image(img, 0, 0);`
	`}`

5. p5.js is in itself a library and therefore comes with some functions that Processing does not have(yay!). For instance, we now have touch events in addition to mouse and key events! This means we can get our sketches working on our mobile.

Processing	p5.js

`PImage img;`
`int number = 10;`
`color[] col = new color[10];`
 	
`void setup(){`
`  img = loadImage(file.jpg);`
`  size(900, 900);`
`}`
 	
`void draw(){`
`  for (var i = 0; i < col.length; i++){`
`  }`
`}`
 	
`void mousePressed(){`
`  //do something`
`}`
	`var img;`
	`var number = 10;`
	`var col = [];`
	`var cLength = 10;`
	
	`function preload(){`
	`  img = loadImage(file.jpg);`
	`}`
	
	`function setup(){`
	`  createCanvas(900,900);`
	`}`
	
	`function draw(){`
	`  for (var i = 0; i < cLength; i++){`
	`  }`
	`}`
	
	`function touchStarted(){`
	`  //do something when you press the mouse OR touch a mobile screen`
	`}`

6. This also means however that some functions, although called the same name, accept a different format or quantity of parameters. For instance, if you’re changing colours in p5.js with openProcessing…

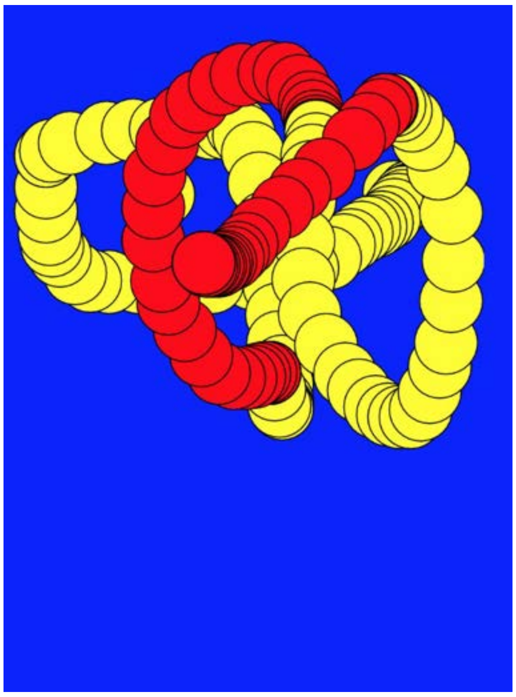
Processing	p5.js

`color c = color(200,100,200);`
 	
`void setup(){`
`  img = loadImage(file.jpg);`
`}`
 	
`void draw(){`
`  size(900, 900);`
	`var c = “rgb(255,255,0)”;`
	
	`function preload(){`
	`  img = loadImage(file.jpg);`
	`}`
	
	`function setup(){`
	`  createCanvas(900,900);`

`  fill(c);`
`  for (var i = 0; i < col.length; i++){`
`  }`
`}`
	`}`
	
	`function draw(){`
	`  fill(c);`
	`  rect(0,0,100,100);`
	`}`

This list of specific differences could get relatively large so let’s just try an example and see how we go.

Time for an example!



```
boolean s;
color c = color(255,255,0);

void setup(){
  background(0,0,255);
        size(375,600);
  s = true;
}

void draw() {
  fill(c);
  ellipse(mouseX, mouseY, 50,50);
}

void mousePressed() {
  if (s) {
    c = color(255,255,0);
    s = false;
  } else {
    c = color(255,0,0);
    s = true;
  }
}
```

Convert this Processing sketch into Javascript with p5.js. Get this sketch working both in openProcessing on your Mobile and Desktop.

** UPDATE(from Week 7) **

Let’s take a look at last week’s exercise and edit it together.

Original Processing sketch:

```
PImage originalImage;
//Create a new Array of 10 colours
color[] col = new color[10];

void setup(){
  background(255);
  size(1200,120);
  originalImage = loadImage("assets/Daedalus.png");
  originalImage.loadPixels();

  //pixels[] Array has (width x height) amount of pixels in image
  int dimension = originalImage.width * originalImage.height;
  //Pick a colour and add it to col[]. Run loop 10 times.
  for (int c = 0; c < col.length; c++){
    col[c] = originalImage.pixels[int(random(dimension))];
  }
}

void draw(){
  for (int i = 0; i < 10; i++){
    noStroke();
    fill(col[i]);
    rect(i*120,0,120,120);
  }
}

void keyPressed(){
  int dimension = originalImage.width * originalImage.height;
  for (int c = 0; c < 10; c++){
    col[c] = originalImage.pixels[int(random(dimension))];
  }
}
```

First, the basics:

1. Every `void` turns into `function`.
2. `size()` turns into `createCanvas()`
3. Any data type declarations(`PImage, float, col`) turn into `var`.
4. Use `function preload(){}` for any files.
5. And, as a nice touch - let's use `touchStarted();`

```
var originalImage;
var col = new color[10];

function preload(){
  originalImage = loadImage("Daedalus.png");
}

function setup(){
  background(255);
  createCanvas(1200,120);
  originalImage.loadPixels();

  var dimension = originalImage.width * originalImage.height;
  for (var c = 0; c < col.length; c++){
    col[c] = originalImage.pixels[int(random(dimension))];
  }
}

function draw(){
  for (var i = 0; i < col.length; i++){
    noStroke();
    fill(col[i]);
    rect(i*120,0,120,120);
  }
}

function touchStarted(){
  var dimension = originalImage.width * originalImage.height;
  for (var c = 0; c < col.length; c++){
    col[c] = originalImage.pixels[int(random(dimension))];
  }
}
```

Done?! Well, not quite - there are some errors we need to address.

We should now see these two errors:

```
mySketch, Line 2: color is not defined
mySketch, Line 11: Cannot read property 'length' of undefined
```

> Line 2: color is not defined:

```
var col = new color[10];
```

If you haven't already noticed, p5.js does not need you to tell it how many items you are going to have in your list ahead of time. Therefore you can simply make an empty Array in the following method:

```
var col = [];
```

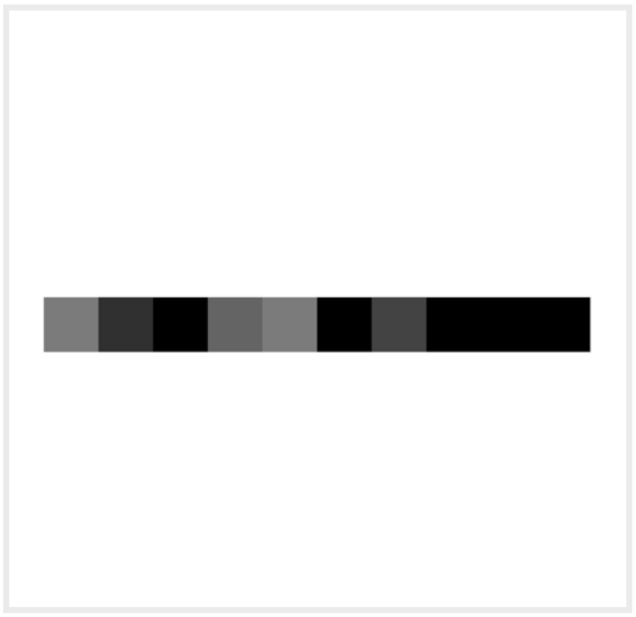
> Line 14: Cannot read property 'length' of undefined

```
for (var c = 0; c < col.length; c++){
}
```

A related issue that is shown is that if you have created an empty Array, the "length" of the Array is undefined. Let's make this a separate value that we define at the top of our sketch. Let's replace all other mentions of `col.length` with `paletteNum`.

```
var paletteNum = 10;

for (var c = 0; c < paletteNum; c++){
}
```

Debug champion! 🏅
Now, which line is next?!



Hmm.. not the best result... they're different shades... but they're not the right colours...

Let's look at our code again! The first thing to double check here are the values in our Array `col`.

> Double check what you think you know! Add the following line underneath the first time we put pixel colours into our colour palette.

```
function setup(){
  ...
  var dimension = originalImage.width * originalImage.height;
  for (var c = 0; c < paletteNum; c++){
    col[c] = originalImage.pixels[int(random(dimension))];
  }
}
```

```
print(col);
```

You will see in the console now that actually, we have a bunch of numbers in our palette(so the computer is drawing the right coloured rectangles) but not in the colours we want.

```
//In the console, you will see the number zero.
0,7,214,110,0,0,0,0,107,0
```

Grr.. so if the computer is right, what have we done?!
Now wait for it… this is **crucial**.

CONSULT THE REFERENCE!

If in any doubt, always consult the reference. So, the first thing we will check is the reference for pixels[] in p5.js

p5.js: pixels[]

Uint8ClampedArray(an array limited to integers between 0-255) containing the values for all the pixels in the display window. These values are numbers. **This array is the size...of the display window x 4**, representing the R, G, B, A values in order for each pixel.
And compare this to pixels[] in Processing...

Processing: pixels[]

An Array containing the values for all the pixels in the display window. These values are of the **color datatype.** This array is the **size of the display window.**

So here lies the difference.

Processing	**p5.js**
Returns pixel as a color datatype	Returns pixel as four values with the datatype int between 0-255

`e.g. An array of 10 pixels`
 	
`{ color(72,125,201), color(84,65,50), color(192, 135, 180), color(255,255,255), color(219,77, 37), color(255,255,255), color(5,107,59), color(12,110,59), color(98,106,168), color(212, 70, 32) };`
	`e.g. An array of 10 pixels`
	
	`{ 72,125,201,255,84,65,50,255,192,135, 180,255,255,255,255,255,219,77,37,255,255,255,255,255,5,107,59,255,12,110,59,255,98,106,168,255,212,70,32,255 };`

Investigating further, I think our easiest option(even though we can solve this lots of ways) is to draw `image` to the canvas and use another pixel function `get(x,y);`

> Draw Image, loadPixels of canvas, use get(x,y) to store random pixel colours.

```
createCanvas(1200,120);
originalImage.loadPixels();
var dimension = originalImage.width * originalImage.height;
for (var c = 0; c < paletteNum; c++){
  col[c] = originalImage.pixels[int(random(dimension))];
  }
```

```
  createCanvas(900,900);
  image(originalImage, 0, 0, 900, 900);
  loadPixels();
  for (var c = 0; c < paletteNum; c++){
    col[c] = get(random(width), random(height));
  }
```

Oh my lord! It works. Now let's replace the following inside of `touchStarted();` and if we don't want that image in the background, call background at the top of `function draw();`

```
function touchStarted(){
  var dimension = originalImage.width * originalImage.height;
  for (var c = 0; c < paletteNum; c++){
    col[c] = originalImage.pixels[int(random(dimension))];
  }
}
```

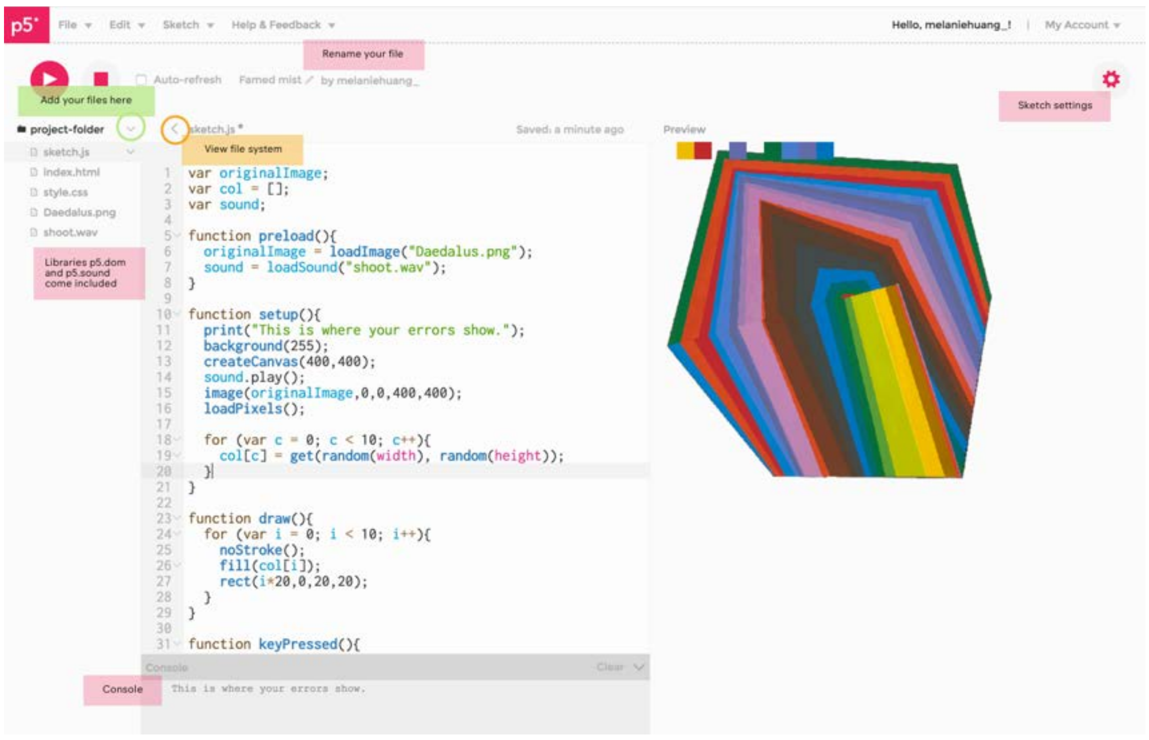
```
function touchStarted(){
  image(originalImage, 0, 0, 900, 900);
  loadPixels();

  for (var c = 0; c < paletteNum; c++){
    col[c] = get(random(width), random(height));
  }
}
```

Phew! Was that a mission but in all honestly, this is what coding is about. Break down the problem into smaller problems. Solve them one by one with the information you have. Now you have really earnt your Debug Champion badge.

🏅

Now see if you can translate your Week 7 sketch to work in p5.js.

Note: Due to some updates in openProcessing, you may find it easier to use the p5.js web Editor https://editor.p5js.org. This is newly released and comes with p5.sound and p5.dom libraries included.



```
var originalImage;
var col = [];
var sound;

function preload(){
  originalImage = loadImage("Daedalus.png");
  sound = loadSound("shoot.wav");
}

function setup(){
  print("This is where your errors show.");
  background(255);
  createCanvas(400,400);
  sound.play();
  image(originalImage,0,0,400,400);
  loadPixels();

  for (var c = 0; c < 10; c++){
    col[c] = get(random(width), random(height));
  }
}

function draw(){
  for (var i = 0; i < 10; i++){
    noStroke();
    fill(col[i]);
    rect(i*20,0,20,20);
  }
}

function keyPressed(){
```

```
This is where your errors show.
```

Everything you need is already on screen.
Use the left arrow circled in orange to reveal the “file system”.
Use the down arrow circled in green to create a folder or upload a file.

Let’s add some sound!

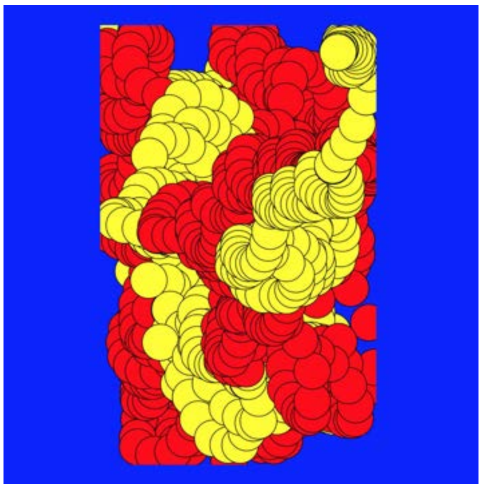
Okay, so that was a bit of a detour but we have climbed the mountain and come out on top. And an important mountain at that; one to get you more confident in exploring the Reference docs as this is something you will continue to do regardless of your skill level.

Let’s play a sound. Make sure your p5.sound library is turned on in Settings.

Here’s some sound files we can play with:
https://drive.google.com/open?id=1yGbLdaNUiJPWfddCUsxTkLkEEpKYZBL-

Attribution required:
/*
Sound from LittleRobotSoundFactory
(https://freesound.org/people/LittleRobotSoundFactory/)
https://freesound.org/people/LittleRobotSoundFactory/packs/16681/
*/

Play a sound
play();



```
/*
Sound from LittleRobotSoundFactory
(https://freesound.org/people/LittleRobotSoundFactory/ )
https://freesound.org/people/LittleRobotSoundFactory/packs/16
681/
*/

var isRed;
var c = "rgb(255,255,0)";
var sfx;

function preload(){
  sfx = loadSound("game-jump.wav");
}

function setup(){
  background(0,0,255);
  createCanvas(375,600);
  isRed = false;
}

function draw() {
  fill(c);
  ellipse(mouseX, mouseY, 50,50);
}

function touchStarted() {
  sfx.play();
  if (isRed) {
    c = color(255,255,0);
    isRed = false;
  } else {
    c = color(255,0,0);
    isRed = true;
  }
}
```

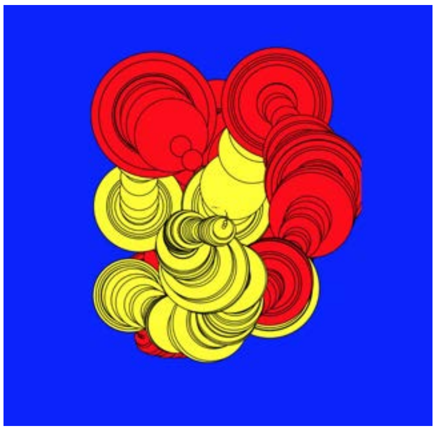
Map the amps to the circle size

p5.Amplitude

Amplitude measures volume between 0.0 and 1.0.

Let’s look at an example: https://p5js.org/reference/#/p5.Amplitude

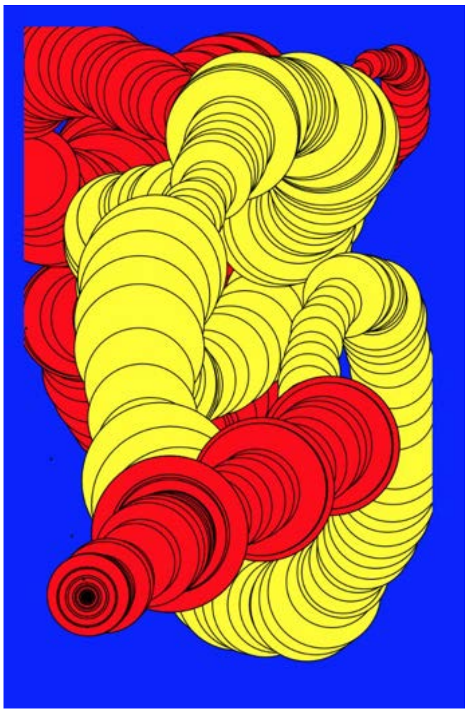
Let’s load a longer sound byte.



```
/*
Sound from LittleRobotSoundFactory
(https://freesound.org/people/LittleRobotSoundFactory/ )
https://freesound.org/people/LittleRobotSoundFactory/packs/16681
/
*/

var isRed;
var c = "rgb(255,255,0)";
var sound;
var a;

function preload(){
  sound = loadSound("lose.wav");
}

function setup(){
  background(0,0,255);
  createCanvas(375,600);
  isRed = false;
  a = new p5.Amplitude();
}

function draw() {
  var eSize = a.getLevel();
  fill(c);
  ellipse(mouseX, mouseY, eSize*400, eSize*400);
}

function touchStarted() {
  sound.play();
  if (isRed) {
    c = color(255,255,0);
    isRed = false;
  } else {
    c = color(255,0,0);
    isRed = true;
  }
}
```

What else can we do?
Up, Up and Delaaaayyyyyy();

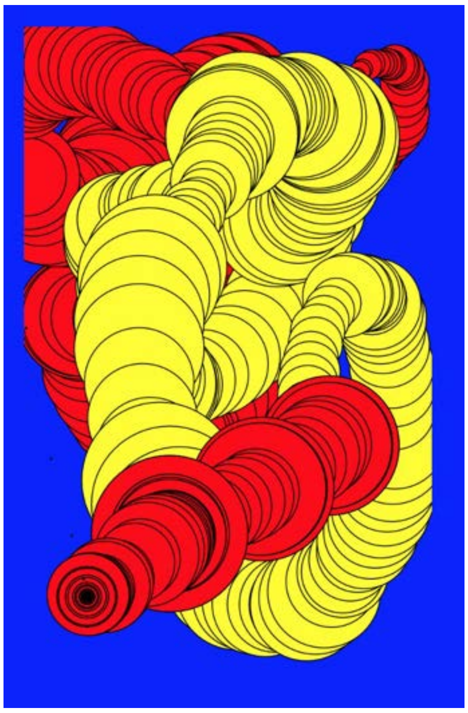
p5.Delay
https://p5js.org/reference/#/p5.Delay



```
/*
Sound from LittleRobotSoundFactory
(https://freesound.org/people/LittleRobotSoundFactory/ )
https://freesound.org/people/LittleRobotSoundFactory/packs/16681
/
*/

var isRed;
var c = "rgb(255,255,0)";
var sound;
var a;
var d;

function preload(){
  sound = loadSound("lose.wav");
}

function setup(){
  background(0,0,255);
  createCanvas(375,600);
  isRed = false;
  a = new p5.Amplitude();
  d = new p5.Delay();
  //source, delayTime, feedback, filter frequency
  d.process(sound, .20, 0.8, 3000);
}

function draw() {
  var eSize = a.getLevel();
  fill(c);
  ellipse(mouseX, mouseY, eSize*400, eSize*400);
}

function touchStarted() {
  sound.play();
  if (isRed) {
    c = color(255,255,0);
    isRed = false;
  } else {
    c = color(255,0,0);
    isRed = true;
  }
}
```

Testing, testing, 1, 2, 3.

p5.AudioIn https://p5js.org/reference/#/p5.AudioIn



```
/*
Sound from LittleRobotSoundFactory
(https://freesound.org/people/LittleRobotSoundFactory/ )
https://freesound.org/people/LittleRobotSoundFactory/packs/16681
/
*/

var isRed;
var c = "rgb(255,255,0)";
var mic;

function setup(){
  background(0,0,255);
  createCanvas(375,600);
  isRed = false;
  mic = new p5.AudioIn();
  mic.start();
}

function draw() {
  var eSize = mic.getLevel()*1000;
  fill(c);
  ellipse(mouseX,mouseY,eSize,eSize);
}

function touchStarted() {
  if (isRed) {
    c = color(255,255,0);
    isRed = false;
  } else {
    c = color(255,0,0);
    isRed = true;
  }
}
```

Maybe if you're really into sound, you might want to synthesize your own sounds with code? Now this is another can of worms that I am terrified to go down BUT well worth the journey because everything is better with sound. Dan Shiffman, over to you.

Week 8 Exercise(p5.js)

Explore the sound library and experiment. Create an interactive sketch that uses sound to control an aspect of your visuals. Check that this works on your mobile.

Think of it as your own **BRAND NEW INSTRUMENT**!

Take the sketch further than the exercises in class. Imagine something playful!