**Advanced Coding**  
**Monday Topics**

Welcome to Advanced Coding! If you are reading this then it means that you have already found the website for the class (<http://www.cse.uaa.alaska.edu/coding> if you somehow got here some other way). If you want to work from home then the website has links Open Processing site, which is free to use but does require you create an account. We will be coding in a language called JavaScript, which runs in a web browser, using a library called p5.js.

For an intro, watch and interact with this video: <http://hello.p5js.org/>

**What is Programming?**

Programming is a series of commands that you give the computer to execute. The commands are kind of a mixture of English, math, and logic. The computer will faithfully execute your commands every time, but you have to carefully construct the commands to complete the task you want the computer to execute!

Programming requires lots of practice. Listening to a lecture can help, but most people need to practice by writing lots of programs. Could you learn to ride a bicycle or play an instrument only by listening to an instructor? So I encourage you to write lots of programs and experiment. In programming it is normal to “fail” many times by writing programs that don’t work! Those failures help you learn how to write a working program.

One of the great things about programming is you can apply it to pretty much anything. There are obvious examples like writing a video game or an app for your smartphone. But programmers also make robots move on Mars, cars that drive themselves, language translators, control medical devices, and pretty much write programs for every field, profession, and discipline!

In this class we’ll be using the JavaScript programming language. If you have heard of other languages, it is similar to Java and C++. Unlike most other languages, JavaScript runs in a web browser. JavaScript is commonly used to write programs that run on the web! From my experience, it seems to work best with Google Chrome, but should run in other modern web browsers. In particular, we’ll use a library called p5js through the website Open Processing (<http://www.openprocessing.org>).

There is a long history about the evolution of programming languages. P5js is based on a language called Processing that in turn is based on Java. You may see references to Processing so please be aware that there are some slight differences between Processing and p5js.

There are some nice resources to learn p5 programming online if you want to take a go independently at your own pace:

* OpenProcessing Tutorials: <https://openprocessing.org/learn/>
* Introduction to Computational Media from CS4All: <https://cs4all-icm.gitbook.io/introduction-to-computational-media-curriculum/>
* Teacher guide: <https://nycdoe-cs4all.github.io/>
* OpenProcessing cs4all: <https://openprocessing.org/class/59960>

## Getting Started with p5js

You can run p5js programs by themselves with a little installation (see <https://p5js.org/> which also has references you may find useful) but we will be using a website called open processing that makes it easier by running everything in the website. You can also see other programs that people have written!

To get started, launch Google Chrome and then navigate to <https://www.openprocessing.org/>

You should see something like this:

A screenshot of a computer

Description automatically generated with low confidence

Click “Join” (or “Sign In” if you already have an account). You will have to provide your name, an email address where you can read the emails, pick a password, and prove you are not a robot. After you sign in, you should see your “home page”

A screenshot of a computer

Description automatically generated with medium confidence

You can search or browse through other programs (sketches) that have been written, see tutorials, edit your profile, or click on the “Create A Sketch” button to make a new sketch. Similar to other social media sites, you can follow other users and give them hearts. Programs are called “sketches” and initially you have 0 sketches.

Once you have an account the first thing you should do is join the class that I have created. Type or click on this URL:

URL: <https://openprocessing.org/join/27DA4C>

Sign in or create an account. You should now be added to the class!

## Create Your First Sketch

Let’s make your first sketch! To do so click on the Create a Sketch button:

A red rectangular sign with white text

Description automatically generated with medium confidence

You will be brought into a code window. You’ll type your code in the text area. By default, there is a setup function and a draw function.



On the right-hand side make sure that p5.js is selected. We’ll talk about other settings options later.

A screenshot of a computer

Description automatically generated with medium confidence

To run the program, click the triangle. You can click it again to re-run the program. To get back to the code window, click on the </> button.

As you can see, the program draws a circle around the mouse! If you want to save your program click the Save button. Pick a name to save it as and you can select whether or not others can see your program.

A screenshot of a computer

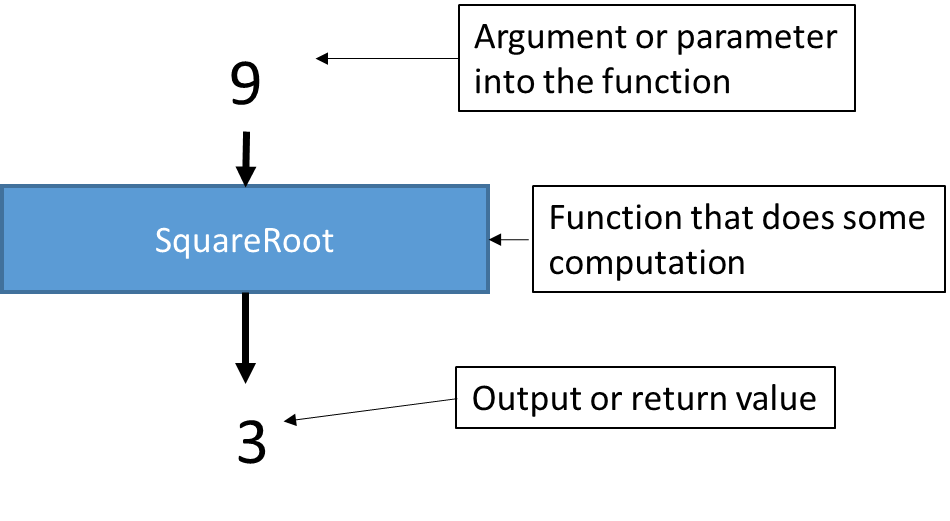
Description automatically generated with medium confidence

A screenshot of a black screen

Description automatically generated with low confidence

Let’s go back to the code window and see explain what is happening!

In JavaScript most of our code is written inside **functions**. You can think of a function kind of like one of the mathematical function buttons on your calculator, like the square root button. On your calculator you can type a number, like 9, hit the square root button, and it does some computation and gives you an answer, in this case, 3. The square root button is the function, the value it operates on (in this case, 9) is called the argument or parameter, and the answer it computes is the return value, or 3.



In JavaScript sometimes our functions have no arguments and may return no value.

A little later we’ll see how to write our own function, but to start with, Open Processing gives us two built-in functions: **setup** and **draw**. Code inside a function is executed (run) from top to bottom.

* setup is run once when the program first starts. Put code in here to set things up.
  + createCanvas tells javascript to make a canvas to draw on. It is set to the size of your web browser’s window.
  + Background sets the background color. If there is a single number it is a value from 0 (black) to 255 (white).
* draw is run repeatedly every frame. Think of a video game or a movie where many frames (say, 30 or 60) are displayed every second. Draw is similar – whatever code you put here will run over and over again, usually 30 or more times every second.
  + circle(mouseX, mouseY, 20) draws a circle that is 20 pixels wide and 20 pixels tall at the location of the mouse.

Let’s look at using the **print** function. Print will output whatever argument is given to it on the screen. If we add this to **setup**:

function setup() {

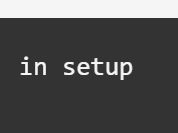
createCanvas(windowWidth, windowHeight);

background(100);

print("in setup");

}

When we run this we should see the program print “in setup” ONCE when the program runs.



Now, try adding this to the bottom of **draw**. The “+” symbol is used to mush together, or concatenate, two different items.

function draw() {

circle(mouseX, mouseY, 20);

print("In draw at " + mouseX + ", " + mouseY);

print(frameCount);

}

Running this program should output a stream of coordinates as you move the mouse along with a counter that increases every time the function is called.

**To do:** Experiment a bit with different numbers for background and different values for the circle instead of 20, and run your program with your experimental values. There is another function called “ellipse” that takes two numbers at the end, for example:

ellipse(mouseX, mouseY, 20, 40);

Experiment with this function as well so see how it works!

**Graphic Coordinate System**

Before we get into drawing we need to learn the coordinate system used by the computer. Computers digitize and represent pictures as tiny dots. If there are enough dots it looks continuous to the eye, because our eye has limited resolution. Each one of these dots, or picture elements, is a pixel.

For example see the dots below:



The graphics screen is set up as a grid of pixels where the upper left coordinate is 0,0. The x coordinate then grows out to the right, and the y coordinate grows down toward the bottom. For example, in the picture below the white pixel is at coordinate (600,400). If you have done graphing in a math class, you are probably used to 0,0 being in the lower left corner instead of the upper left.



If we make selective pixels different colors then we can perceive letters, pictures, and so on.

**Drawing Shapes**

Here is a program that draws a few pixels:

function setup() {

createCanvas(windowWidth, windowHeight);

background(255);

point(100, 100);

point(101, 100);

point(100, 101);

point(101, 101);

}

You might need to look carefully to see the dots, they may be quite small!

By putting this in “setup” the code is only run once, when the program is first executed.

Sometimes you want to leave your code in but not have it run. You can “comment out” code or use this technique to describe how your code works. You can preface a line with two slashes // and the rest of the line is ignored.

You can also use /\* and everything is ignored until \*/

function setup() {

createCanvas(windowWidth, windowHeight);

background(255);

point(100, 100); // Draws only a single pixel

//point(101, 100); // This line is ignored

/\*

point(100, 101);

point(101, 101);

\*/

}

We could draw a lot of things but it quickly becomes painful drawing them pixel by pixel. To help us there are a variety of functions to draw different shapes. For a reference you can go to <https://p5js.org/reference/>

A picture containing text, screenshot, font, line

Description automatically generated

A picture containing text, font, line, white

Description automatically generated

A close-up of numbers

Description automatically generated with low confidence

A screenshot of a computer code

Description automatically generated with low confidence

A picture containing text, screenshot, font, design

Description automatically generated

A picture containing text, screenshot, font, design

Description automatically generated

The most common way to specify a color is with Red, Green, and Blue (RGB) values that are mixed together.

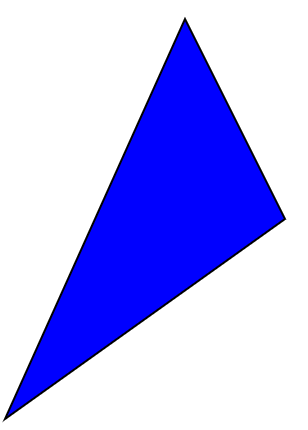


The red, green, and blue values each range from 0 to 255. When they are all 0 you get black. When they are all 255 you get white. If R=255, G=0, and B=0, you get red, and so on. Mixing different values gives a different color.

The fill function takes a red, green, and blue value to make a color other than grey. For example the following sets the fill to strong blue:

fill(0,0,255);

triangle(100,100,150,200,10,300);

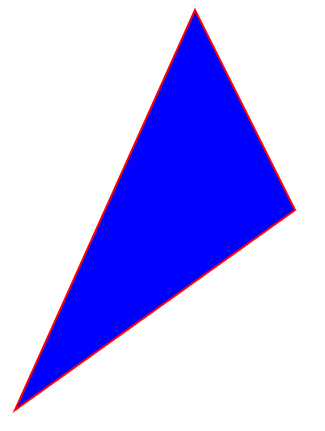


If you want to change the color of the black outline then use the stroke function:

fill(0,0,255); // Blue fill

stroke(255,0,0); // Red outline

triangle(100,100,150,200,10,300);



Other functions to play with:

* clear(); // clear the screen
* noFill(); // no fill
* noStroke(); // no stroke
* strokeWeight(number); // Sets thickness of stroke
* fill can take a fourth number, the alpha value! Experiment with it to see what it does.
* background() can also take up to 4 numbers if you want a color other than gray (R,G,B, and alpha)

Note the order shapes are drawn is top to bottom (determines overlap) and that you have to be precise with upper and lower case or the function will not be recognized.

## Coding Challenge!

Choose a band (optionally, a song) and create an abstract album art for it.

Write a program that uses the shapes and colors covered to create the art. It can be static or you could have something change with the mouse if you like. Refer to the reference at <https://p5js.org/reference/> and add other components such as text if you like (see ‘text’ on the reference).

Some examples more on the fancy level:



When you are ready, share your sketch with the classroom. Give a heart to the sketches you like the best! The sketch with the most hearts wins a prize!

## Exploring Other Programs on OpenProcessing

You have probably noticed that you can explore programs that other people have written and shared on the Open Processing website. Feel free to explore and look at the code to see how the programs were written! Note that some of these programs were written in Processing, not p5.js, so you may want to check the settings. If you want to copy it to your own space where you can then make modifications to it you can “fork” it with the icon in the upper right corner.

A white line on a black background

Description automatically generated with low confidence

## Variables

In math, you use variables to represent a value that can change. You can do something similar in JavaScript. You first have to declare a variable. You do this by typing in the keyword **var** or **let** followed by the name of your variable. (let is the more modern term, it has a slight difference in meaning than var but we won’t worry about the distinction here).

You get to pick the name of the variable so it is meaningful. Your variable name can’t have spaces or punctuation characters, so it is basically letters and numbers. It has to start with a letter. JavaScript also considers an uppercase letter to be different than lowercase, so FOO and foo are considered different variable names.

Here is an example that makes a variable named **x** and another variable named **score** and then applies some basic math operations to the variables:

let x, score;

x = 5;

score = 100;

score = score + 1;

print(score); // 101

x = x - 2;

print(x); // 3

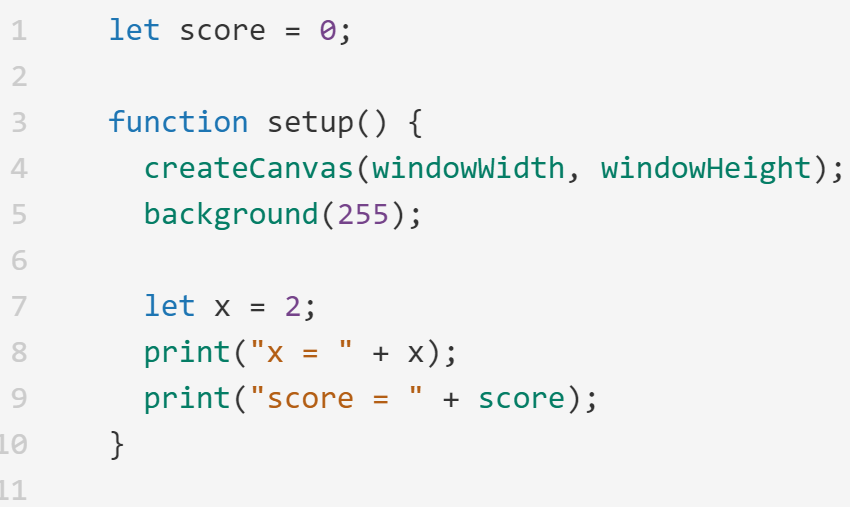
x = x \* 2;

print(x); // 6

x = x / 4;

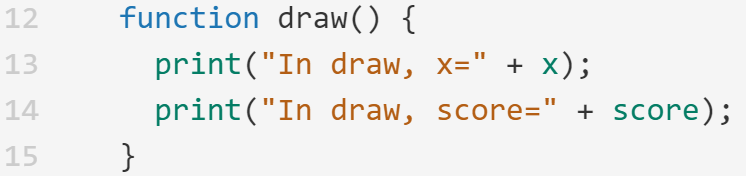
print(x); // 1.5

There is a concept called **scope**. A variable’s scope basically specifies where a variable is alive. When we use **let** to declare a variable, the variable is alive within its block. For now, a block is basically a function or the whole program. Here is an example that illustrates scope:

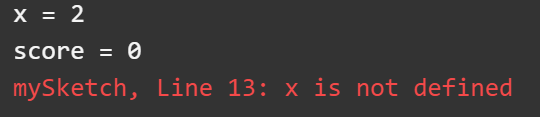


When we run this program it outputs 2 for x and 0 for score (remember the + in the print statement is to concatenate, or squish together, not to add). Just what we expect, right? Except score is declared up at the top on line 1 while x is on line 7! What is the difference?

If we add a draw() function we can see the difference:

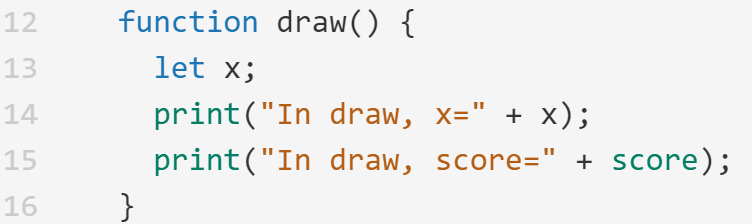


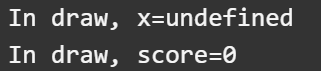
If we try to run this we get an error!



The program ran setup, but when it tried to run draw there was an error. It doesn’t know what variable x is. This is because the variable x is declared inside the setup function, so its scope is local to setup. It is not accessible outside setup.

We can define x again inside draw, but this will create a different variable named x!





However, note that there is no problem with score. That is because it was declared at the top of the program, outside the functions. This gives it global scope so it can be accessed inside multiple functions.

|  |
| --- |
| A variable declared in a function exists only inside that function. This is called a local variable. |
| A variable declared at the top of the program exists for the whole program. It is shared and can be accessed from all functions. This is called a global variable. |

There are a number of built-in variables. We have already seen two of the, mouseX and mouseY. These variables are automatically set by p5.js to the coordinates of the mouse on the screen. Here is a small variant of the first sketch that is created for us when we create a new sketch:

function setup() {

createCanvas(windowWidth, windowHeight);

background(255,255,0);

noStroke();

}

function draw() {

fill(0,0,255,100);

circle(mouseX, mouseY, 20);

}

This draws a blue dot at the coordinates of the mouse. I used an alpha value of 100 so there is a little transparency with each blue dot.

What if we didn’t want to leave a trail of blue dots everywhere, we only want one blue dot to show? We can erase the screen every frame by redrawing the background before drawing the blue dot!

function draw() {

background(255,255,0);

fill(0,0,255,100);

circle(mouseX, mouseY, 20);

}

## Making Decisions

Programs are more interesting when they can make decisions. The most basic way for a program to make a decision is to use an **if statement**.

An if statement consists of the keyword if followed by something in parenthesis. If the thing in parenthesis is true, the code that comes after the if statement is executed. If the thing in parenthesis is false, the code that comes after the if statement is skipped.

if (condition)

{

// Run this if condition is true

}

If condition is true, then the code inside the { } is run. If condition is false, then it is skipped.

We can optionally add an **else** clause. If the condition is false then the code inside the else { } is run.

if (condition)

{

// Run this if condition is true

}

else

{

// Run this if condition is false

}

As an example, there is a variable named **mouseIsPressed** that is set to true or false depending on whether or not the mouse is pressed. This code will change the fill color to yellow if the mouse is not pressed, and set the fill color to blue if the mouse is pressed.

function draw() {

if (mouseIsPressed)

{

fill(0,0,255,100);

}

else

{

fill(255,255,0);

}

circle(mouseX, mouseY, 20);

}

Inside the if statement we can use relational operators to test for less than, greater than, less than or equal to, greater than or equal to, equal to, or not equal. The table below shows how to make these tests:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Symbol in JavaScript** | **Example** |
| Less than | < | if (x < 10) |
| Greater than | > | if (x > 10) |
| Less than or equal to | <= | if (x <= 10) |
| Greater than or equal to | >= | if (x >= 10) |
| Equal to | == | if (x == 10) |
| Not equal to | != | if (x != 10) |

**Testing if the mouse is in a Circle and Motion**

There is a function named **dist** that calculates the distance between two points. If we have a circle and the distance from the mouse to the center of the circle is less than the radius then it means that the mouse is inside the circle.

Here is a longer example that ties together some of the things we have been talking about. It moves a circle left/right on the screen. If you click on the circle then it changes color.

This program is available in the Open Processing Class under the name “Circle Bounce”

let circleX; // X coordinate of center of our circle

let velocityX; // velocity of the circle in the X dimension

let diameter; // Diameter of our circle in pixels

function setup() {

createCanvas(windowWidth, windowHeight);

background(0);

noStroke();

// Initialize variables

circleX = 10;

diameter = 50;

velocityX = 5;

}

function draw() {

// Erase screen

background(0);

// Default circle color is white

fill(255,255,255);

// Compute new location for the circle

circleX = circleX + velocityX;

// Check if we reached the edge of the screen.

// If so, reverse velocity

if (circleX >= windowWidth)

// Off right side of the screen

{

velocityX = velocityX \* -1;

}

if (circleX <= 0)

// Off the left side of the screen

{

velocityX = velocityX \* -1;

}

// Check if we clicked on the circle

if (mouseIsPressed)

{

// If we clicked within the circle change color

// to red

if (dist(mouseX,mouseY,circleX,150) <

(diameter/2))

{

fill(255,0,0);

}

}

circle(circleX, 150, diameter);

}

## Coding Challenge!

Fork the circle bounce program and modify it so the circle also bounces in the Y dimension (like pong). Modify it so the circle moves faster each time it bounces off a wall.

Feel free to play with other colors, shapes, or other things using the mouse position as well!

If you want to add some randomness, see the “random” function on the p5 reference page.

Share your sketch on the Open Processing classroom in the “Monday submissions” folder.

## Coding Challenge!

Learn how to use global variables to control different stages and what is drawn on the screen.

First, the keyPressed() function will tell us if a single key is pressed (this is a distinct keypress as opposed to holding a key down, which we can check for keyIsPressed like we used for mouseIsPressed)

function keyPressed() {

// Runs if we press any key

}

Use this function in your album art sketch so that when the sketch is first run, there is a “splash screen” that says “Press any key to start” and after a key is pressed then your album art is drawn/run. You can use a global variable to determine what should be drawn depending upon whether we have pressed the key or not.

## Creating Functions

So far we’ve been working in the setup and draw functions, but we can make our own! (there are also some other pre-defined functions as we will see).

The format to write a function is pretty much like what we have done so far with setup and draw:

function *nameOfFunction*(parameter1, parameter2, …)

{

// Code

return *returnValue*;

}

For example, in the previous code we used a function called dist that computed the distance between two points. We could write this ourselves using the distance formula derived from the Pythagorean theorem:

function setup() {

createCanvas(windowWidth, windowHeight);

background(100);

**print(myDistance(1,1,3,3));**

}

**function myDistance(x1,y1,x2,y2)**

**{**

**let dist;**

**dist = sqrt((x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2));**

**return dist;**

**}**

From setup, the values 1 and 1 are sent into the myDistance function as variables x1 and y1. These variables are called parameters and the numbers 1 and 1 are called arguments. The values 3 and 3 are sent in as variables x2 and y2.

Inside myDistance, the function computes the square root of (x1-x2)2 + (y1-y2)2 and then returns the value. The return value is sent back and substituted in where the function is called.

Sometimes you don’t want to return a value from a function. If this is the case, you can just omit the return statement. Here is an example of a function that draws a bullseye.

function bullsEye(x, y, diameter)

{

fill(0,0,255); // Big blue circle

circle(x,y,diameter);

fill(255,0,0); // Medium red circle

circle(x,y,diameter\*2/3);

fill(255,255,0); // Smallest yellow circle

circle(x,y,diameter/3);

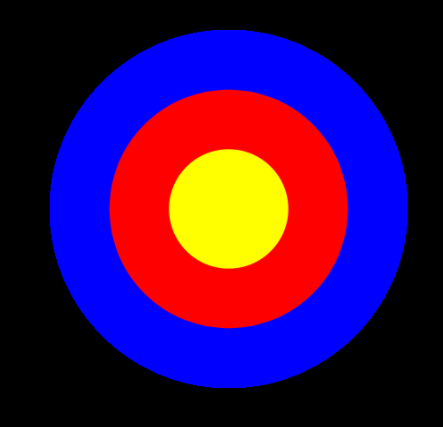
}

I set this up so that x,y is the center of the target and diameter is the diameter of the outer circle. The function could be invoked from draw as follows:

function draw() {

bullsEye(400,400,200);

}



## Note Synthesizer

We can play music and sounds in p5 as well! First we need to enable the p5.sound library. To do this click on the settings icon in the upper right corner:



Scroll down to p5.sound and toggle it on:

A black screen with white text

Description automatically generated with low confidence

This code plays a note based on the mouse position. Notes can be specified as a string (see the reference for more details!) or as a number in Hertz. 262 Hz is middle C.

let monoSynth;

function setup() {

createCanvas(windowWidth, windowHeight);

background(100);

monoSynth = new p5.MonoSynth();

}

function draw() {

if (mouseIsPressed)

{

playNote();

}

}

function playNote() {

userStartAudio();

let note = mouseX; // 262 is Middle C

let velocity = 1; // 1 is loudest, 0 is quiet

let time = 0; // Time from now

let dur = 1/6; // note duration (in seconds)

monoSynth.play(note, velocity, time, dur);

}

As another example, we can also play sound files. First, I downloaded a sound file (mine came from <http://soundimage.org/horrorsurreal>).

The sound to play needs to be uploaded to the website. Click on the “Files” tab and drag the file in:

A screenshot of a computer

Description automatically generated with medium confidence

Here is code to play it:

let mySound;

function preload() {

mySound = loadSound('Bells-of-Weirdness.mp3');

}

function setup() {

createCanvas(windowWidth, windowHeight);

background(100);

}

function draw() {

if (mouseIsPressed)

{

mySound.play();

}

}

We have to declare some variables globally so they are accessible in all functions. There is a new function we are using called preload. It is like setup, except it waits for things to finish loading, like sounds or music.

Note that this will overlap sounds if you click while one is already playing. You could use “mySound.isPlaying()” to see if a sound is currently playing, and use “mySound.stop()” to stop a sound from playing.

## Coding Challenge!

1. Can you write a program that determines what frequency range you can hear?
2. Add some type of graphical visualization of your choice to the playing of a sound or of notes. This is pretty open-ended!

Submit your sketches to the Monday Submissions folder.

## Loops

Earlier we made a function that drew a bullseye at specified coordinates in a specified size. Consider the following code that draws 5 bullseyes in slightly different places. Note that I put the line background(100) into draw(), so it won’t erase the previous ball each time the function is called.

function draw() {

background(100);

bullsEye(100,100,50);

bullsEye(200,200,100);

bullsEye(300,300,150);

bullsEye(400,400,200);

bullsEye(500,500,250);

}

A picture containing circle, colorfulness, screenshot, graphics

Description automatically generated

If we wanted to draw 100 bullseyes this would get tedious. To make our life easier we can use a **loop** to repeat a block of code. We will examine the while and for loop constructs.

## While Loop

The while loop allows you to direct the computer to execute the statement in the body of the loop as long as the expression within the parentheses evaluates to true. The format for the while loop is:

while (boolean\_expression)

{

statement1;

…

statement N;

}

As long as the Boolean expression evaluates to true, statements 1 through N will continue to be executed. Generally one of these statements will eventually make the Boolean expression become false, and the loop will exit.

In terms of a flowchart, the while loop behaves as follows:

A diagram of a loop

Description automatically generated with low confidence

Here is an example of a while loop that prints the numbers from 1 to 10:

let i = 1;

while (i <= 10)

{

print(i);

i = i+1;

}

If we wanted to print out 1,000,000 numbers we could easily do so by changing the loop! Without the while loop, we would need 1,000,000 different print statements,

Here is our bullseye drawing code converted to a while loop:

function draw() {

background(100);

let i = 1;

while (i<=10)

{

bullsEye(i\*100,i\*100,i\*50);

i++; // Same thing as i = i + 1;

}

}

## The For Loop

The for loop is a compact way to initialize variables, execute the body of a loop, and change the contents of variables. Normally it is used when you want to count a certain number of times. It consists of three expressions that are separated by semicolons and enclosed within parentheses:

for (expression1; expression2; expression3)

{

Statement;

…

}

Expression1 is used to set initial values, and can set multiple values separated by a

comma.

Expression2 is the condition for the loop to continue (while this is true).

Expression3 contains any operations you’d like to do at the end of each iteration.

Separate different instructions with a comma.

Here is the for loop that counts to 10:

for (let i = 1; i <10; i++)

{

print(i);

}

Here is the bullseye code written as a for loop:

for (let i = 1; i <= 5; i++)

{

bullsEye(i\*100,i\*100,i\*50);

}