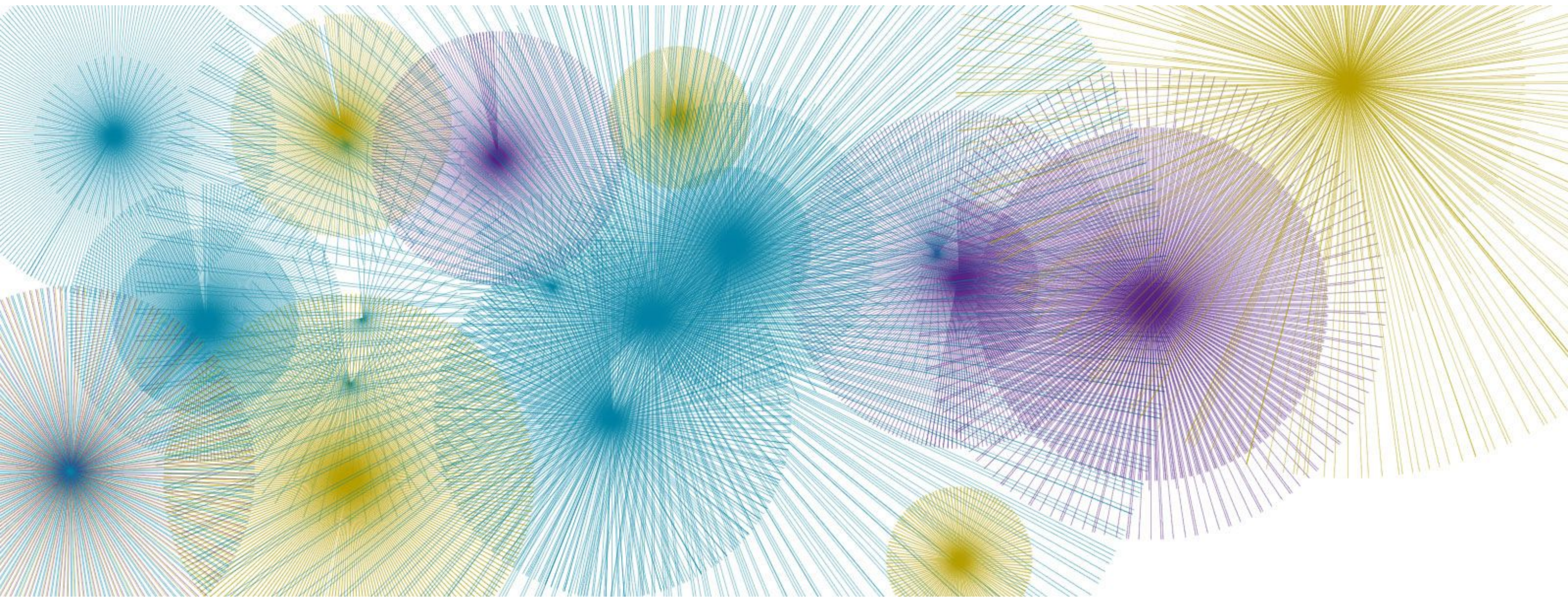

(image source: http://www.lyceelecorbusier.eu/p5js/?p=2611)

An Introduction to P5.js

A manual for the JavaScript P5.js workshop for Creative Technology

1. Table of Contents

2. Introduction

Nowadays, Javascript is used in many online applications such as social media sites, web shops and even used for writing other kinds of software. It is the language that defines the behavior of web pages, while HTML defines the content of webpages and CSS defines how this content looks. Javascript is therefore one of the most used languages in web technology.

This manual explains the basics of JavaScript and the P5.js library by making a little reaction game where you have to click a bell before it disappears. The bells will appear randomly and the fastest reaction time will be shown on the screen. A screenshot of the game is given in Figure 1. We will even make the game make sounds when you click a bell to show how much is possible with P5.js.

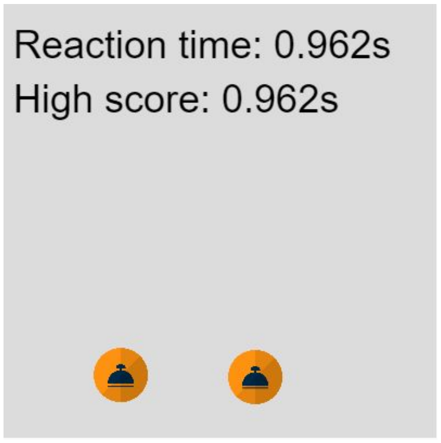


Figure 1: A screenshot of the reaction game

Now as said before, P5.js is not a language. It is a library for JavaScript that makes the JavaScript language more accessible for people who want to be able to work with interaction, but do not have much programming experience or want a nice result as easy and quick as possible. It is used by designers, artists, educators and creative technologists like you!

The creator of P5.js is Lauren McCarthy and as you will notice, P5.js is very familiar to Processing, that is due to the fact that P5.js was created with the same goal as Processing and has received support from the Processing foundation.

3. The online editor

P5.js has a pretty good and complete online editor which can be found on editor.p5js.org. In this editor you can write and run your code in your web browser so no installation of any IDE is required.

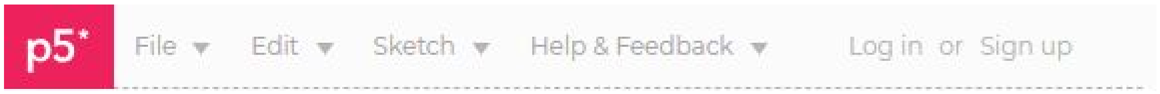
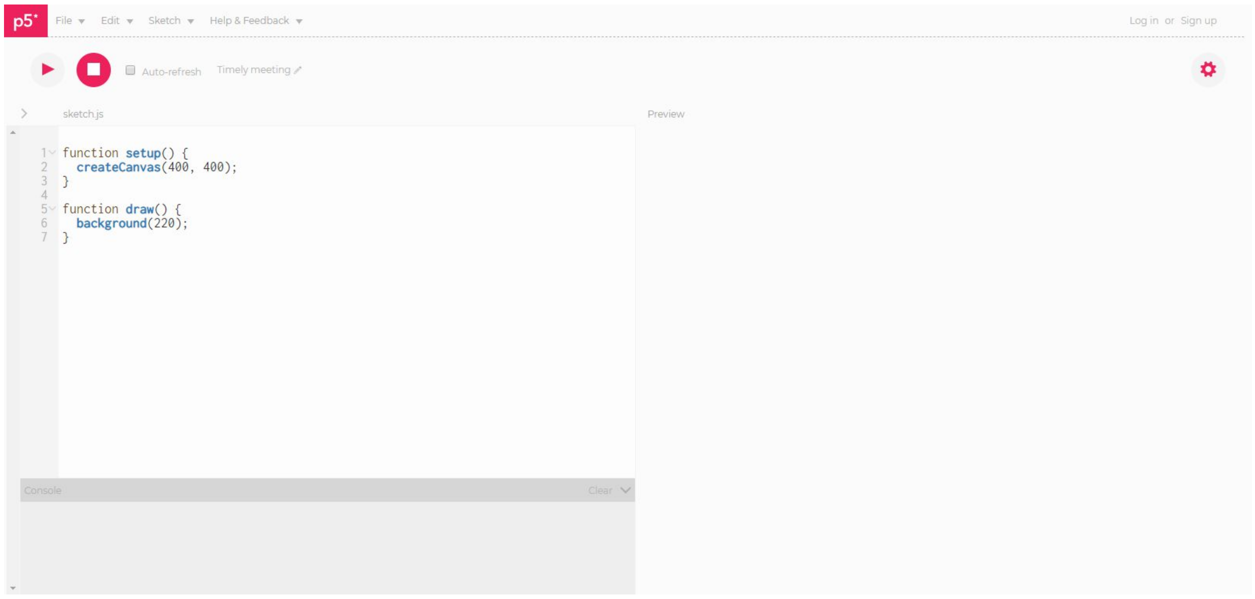


Figure 2: The menu bar of the online editor



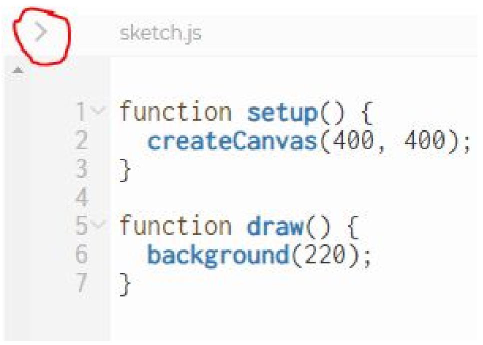
```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
}
```

Figure 3: The P5.js online editor

As you can see in Figure 2, the editor has a menu bar on top where you can safe your sketches and open them, see examples, tidy your code, search for certain keywords, run and stop the sketch and access information about the editor or the P5.js language.

You can also make an account and log in. Please do make an account by signing up by pressing the button in the top-right. Now,. if you are logged in, you can safe sketches and edit assets!

Now as you can see in Figure 3, just like in Processing, there is a play and a stop button which allows you to run and stop your sketch. Underneath these buttons there is an area where you can actually write your code, with the console underneath and the preview which is drawn on to to the right.



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

function draw() {
  background(220);
}
```

Figure 4: the arrow showing the file structure

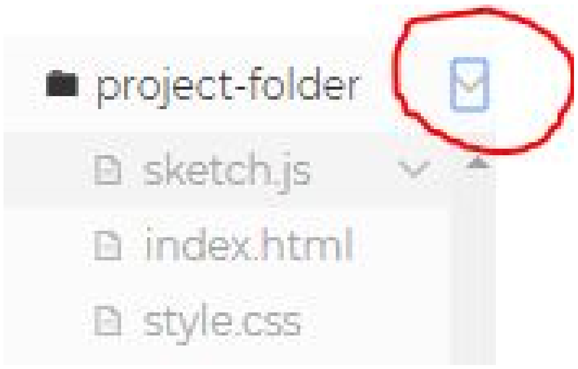


Figure 5: the file structure and the button used to add files

By pressing the arrow indicated in Figure 4, you can show the file structure of your sketch and by pressing the arrow next to the folder-name indicated in Figure 5, you can add files, which are from now on referred to as assets.

4. Program structure

As you can see, a blank program looks like this:

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

function draw() {
  background(220);
}
```

Just as in Processing, a sketch has two default functions most of the time: `setup()` and `draw()`. Instead of void they are now of type function. These functions will be discussed in section 12. But, just as in processing, setup() is called only once at start of the sketch and draw() runs after that and repeats over and over again.

P5.js contains additional methods regarding the program structure such as `preload()`, `loop()` and `noLoop()`.

You can also see the function `createCanvas(400, 400);` in the `setup()`. This creates a canvas element and sets the dimensions of it in pixels, just like `size()` does in Processing. `createCanvas()` should only be called at the start of `setup()` and will result in unpredictable behaviour when called more than once in a sketch.

You can also find the function `background(220);` inside of `draw()`. This function sets the color for the background of the canvas and is typically used at the beginning of `draw()` to clean the canvas for the next frame.

5. Javascript and printing

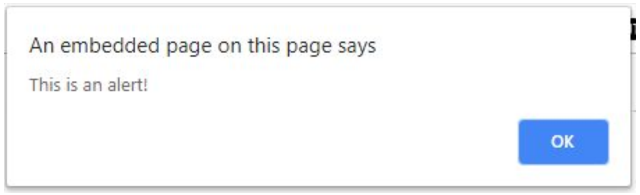
Because P5.js is a JavaScript library, you can run vanilla JavaScript inside the editor. An example of this is the `alert()` function:

```
alert("Good luck!");
```

This function creates a popup box in your browser saying what you specify within its brackets, such as Good luck!. JavaScript has three of these popup boxes: an Alert box, Confirm box and a prompt box.

You can try this by placing the line above inside of your `setup()`:

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);
}
```


An embedded page on this page says

This is an alert!

Figure 6: the alert message shown in Google Chrome

As you can see, an alert box pops up as can be seen in Figure 6.

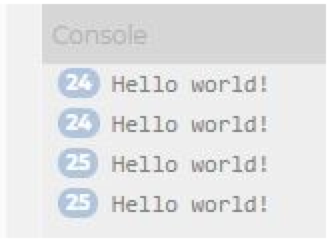
Another useful tool in JavaScript for getting your program to give information is by using the `console.log()` function :

```
console.log("Hello world!");
```

This logs out the given text to the console underneath your editor.

You can try this by writing the line above in your `draw()`:

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);
  console.log("Hello world!");
}
```



```
Console
24 Hello world!
24 Hello world!
25 Hello world!
25 Hello world!
```

Figure 7: the result of the console.log statement in the console

As you can see in Figure 7 and for yourself, now you first get an alert popup and after you click that away, your sketch starts saying hello to you!

This `console.log()` function is actually replaced in P5.js by the `print()` function. This is familiar to Processing and does exactly the same: printing to the console! Replacements like these by P5.js make JavaScript a more readable and accessible language. Now let's replace the `console.log()` function with `print()`:

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);
  print("Hello world!");
}
```

This yields exactly the same result as with using `console.log()`.

6. Drawing shapes

Now we are going to draw shapes on our canvas. This can be done either in 2D or in 3D, but for the sake of simplicity we will stick with 2D in this manual. If you want to explore working in 3D, feel free to explore! Whether you are working in 3D or want to stay in 2 dimensions, you can always consult the reference guide and if you run into any trouble remember that there probably is someone on the internet who already has had the same problem as you.

Just as in processing, you can easily draw shapes to your canvas with the given set of functions such as `ellipse()`, `line()` and `rect()`. Feel free to explore these functions, which are similar to the ones in Processing. For now, we will just draw circles. Let’s draw a circle:

```
ellipse(100, 100, 50, 50);
```

This will draw a circle at 100 pixels from the left and 100 pixels from the top with an x-diameter of 50 pixels and a y-diameter of 50 pixels:

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);
  print("Hello world!");

  ellipse(100, 100, 50, 50);
}
```

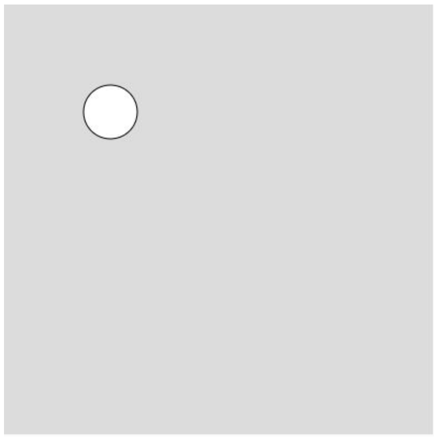


Figure 8: the canvas where the circle is drawn

This will first refresh the background, say hello and then draw the ellipse at the given location as shown in Figure 8. However, it is a little boring.

7. Color

To make the circle we draw less boring we should add color! Color is defined as a single grayscale value, an RGB-value or an RGBA-value. Where each element has a range from 0 to 255. The difference between the aforementioned values is as follows:

- A single grayscale value basically stands for a shade on a gradient from black to white. It's on a scale of the shades of gray: grayscale!
- RGB contains three grayscale values which stand form Red, Green and Blue respectively: an RGB value with elements (255, 0, 0) would result in a bright red color, a value with elements (0, 255, 0) would result in a bright green color, and a value with elements (0, 0, 255) would result in a bright red color. Just like in Processing.
- RGBA adds an Alpha value to the RGB spectrum. This value stands for the opacity of the color.

A color can be assigned to an object-to-be-drawn by using the function `fill()`:

```
fill(255, 30, 200);
```

This will give our circle a purple color, already making it more interesting:

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}
```

```
function draw() {
  background(220);
  print("Hello world!");

  fill(200, 30, 255);
  ellipse(100, 100, 50, 50);
}
```

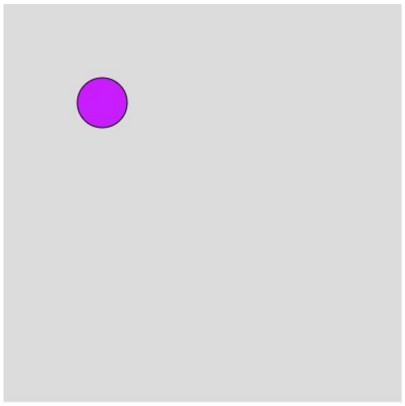


Figure 9: the canvas where the purple circle is drawn

As you can see in Figure 9, the background is basic gray. This is the background we will be using but of course you can also assign a color to the background to make your sketch even more interesting:

```
background(20, 220, 40);
```

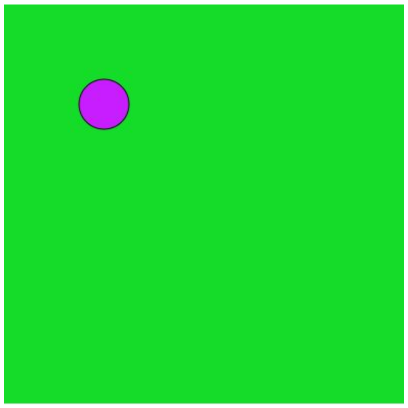


Figure 10: the canvas with a green background

I understand that the result shown in Figure 10 is not the most “visually pleasing”, but you get the point.

8. Comments

Because code can quickly become very difficult for humans to read, it is good to explain in "human language" what your code does. You can do this with comments. Comments are used to explain a line or a section of your code, you can state the title of your code and the author, or you can even use it to disable a line or a section of your code for debugging and testing purposes.

A comment on a single line in JavaScript looks like this:

```
// This is a comment.
```

Any text between // and the end of the line will be ignored by JavaScript and will thus not be executed. Let's use this to explain in our code what we are doing (drawing a purple ball):

```
// Draw a purple ball
fill(200, 30, 255); // Make it purple
ellipse(100, 100, 50, 50); // Draw the ball
```

You can also put comments on a block of lines, called multi-line comments. These comments start with /* and end with */. All text in between is ignored by JavaScript, making it easy to comment out a whole section of your code.

We can use this for example if we do not want to draw our purple ball:

```
  /*
  fill(200, 30, 255);
  ellipse(100, 100, 50, 50);
  */
```

9. Loops

Now if we want to draw a more of our purple circles, we have to retype our code over and over again, like this:

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);
  print("Hello world!");

  fill(200, 30, 255);
  ellipse(100, 100, 50, 50);

  fill(200, 30, 255);
  ellipse(150, 100, 50, 50);

  fill(200, 30, 255);
  ellipse(200, 100, 50, 50);

  fill(200, 30, 255);
  ellipse(250, 100, 50, 50);
}
```

Of course this is very cumbersome and we, programmers, are lazy. The solution: loops! We will discuss two kinds of loops: the while loop and the for loop. These are very similar to Processing, except for the variable declaration.

9.1. The while loop

Let's start with a while loop. This kind of loop executes a bunch of code depending on whether some specified condition is true. The place where this code is stored is called the body of the loop. Take a look at the following example:

```
var number = 1;
while (number <= 10) {  // condition
    print(number);      // body
    number++;           // updater
}
print('Done!');
```

In this example, we first declare a variable called number and initialize it with value 1 (this variable declaration might look a little odd if you are used to Processing, but we will discuss this in the next chapter). We will then start writing our while loop. The line saying `while (number <= 10)` basically means "while number is smaller than or equal to 10, execute the following code". So here we check whether number fulfills our condition and if this is the case, we will go to the body and print out the number. If this is not the case, we ignore the code in the body and print out that we are done.

Please note that inside the while loop there is also the line `number++;` This is called the updater and updates the variable that we check our condition on in such a way that the condition is eventually fulfilled. This is very important, because if the condition will never be fulfilled, there is an infinite loop. This might be undesirable depending on what you want to do with the loop. (a desirable scenario would be a similar function to draw(), which also loops infinitely like `while(true)` where the condition is always fulfilled).

9.2. The for loop

Next is the for loop. It's execution is similar to the while loop, even though it looks different. Take a look at the following example:

```
for(var number = 1; number <= 10; number++) {
    print(number); // body
}
print('Done!');
```

The result of this example is exactly the same as the previous one: it prints out the numbers 1 to 10 and prints ‘Done!’ when it is done. The exception is that now we declare our condition variable, our condition, and the updater on the same line. This line basically translates to “for as long as the variable number starting at 1 is smaller than or equal to 10, increase it by one and execute the code inside the body”. This means that as long as number is smaller than 11, the number is printed. If the number is bigger than 10, the loop is finished and it prints ‘Done!’.

There are two other kinds of loops which are the do-while loop and the for-in loop. The first is almost the same as the while loop, except that the condition is checked after the code is executed. The for-in loop loops through all properties of an object and executes the loop's body once for each of these properties. If you want to know more about these types of loops, go to this link.

10. Variables

We just saw that declaring a variable in JavaScript is different from Processing: you do not have to declare a type. Therefore making a variable looks like this:

```
var number = 1;
```

This has its pros and cons. First of all, you don't have to worry about data types, because Javascript can figure it all out. On the other hand, there are other reasons why types are more useful. For example if an integer only needs 8 bits (e.g. to have a range from 0 to 255), but 16 bits are reserved for it, there is a lot of overhead. JavaScript is thus not the most efficient language to work with in terms of execution speed and memory, but is very efficient in terms of programmer effort.

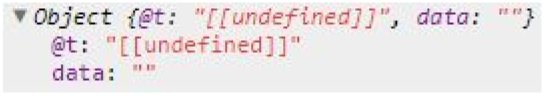
JavaScript variables must all be identified with unique names which are called identifiers. They are case-sensitive and can come in many forms, depending on your whish e.g. x, variable1, varOne, var_1, etc.

A variable is initialized by assigning a value to it using the assignment operator, the equal sign (=). This does not have to be done at the same line as the declaration:

```
var number;
Number = 1;
```

If a JavaScript variable is not initialized, it's value is undefined. No literally, the value is *undefined*. The code below returns the result shown in Figure 11:

```
var value;
print(value);
```



```
▼Object {@t: "[[undefined]]", data: ""}
   @t: "[[undefined]]"
   data: ""
```

Figure 11: a screenshot of printing an undefined value

A JavaScript variable can also hold values other than integers such as floating point numbers and strings:

```
var pi = 3.1415; // a float
var language = "JavaScript"; // a string
var song = 'Viva la Vida'; // a string
```

Or you can even store a color inside a variable:

```
var c = color(200, 30, 255);
```

You can also re-declare a JavaScript variable. If you do, it will not lose its value:

```
var language = "JavaScript"; // a string
var language;
```

After execution of the previous two statements, the variable language still holds the value "JavaScript".

Now that we know about variables, we can try to place our purple ball on a location depending on a variable so that we can make it move:

```
var x = 100; // a global variable holding the x-position
var y = 100; // a global variable holding the y-position

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);
  print("Hello world!");

  // Draw a purple ball
  fill(200, 30, 255); // Make it purple
  ellipse(x, y, 50, 50); // Draw the ball

  x = x+1; // Add 1 to x
}
```

The code above makes our purple ball move to the right and outside of the canvas.

11. Random()

We can't always rely on hard-coding all values and positions. It is nice to add some randomness to our program and let P5 decide where the ball is placed. We do this with the P5.js function `random()`. By default, this function returns a randomly chosen value between 0 up to 1, excluding 1. You can also tell the function on what range to choose a number from or you can give it an set of numbers (an array) and it will choose one of the given values. We will talk more about arrays later on. Below you can see all different configurations of the `random()` function:

```
random(); // returns a value from 0 up to 1
random( max ); // returns a value from 0 up to max
random( min, max ); // returns a value from min up to max
random( choices ); // returns one of the choices that are given in an array
```

Now we can use the random() function to make our ball appear randomly:

```
var x; // a global variable holding the x-position
var y; // a global variable holding the y-position
```

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");

  x = random(0, width); // Assign a random value between 0 and the screen
width
  y = random(0, height); // Assign a random value between 0 and the screen
height
}

function draw() {
  background(220);
  print("Hello world!");

  // Draw a purple ball
  fill(200, 30, 255); // Make it purple
  ellipse(x, y, 50, 50); // Draw the ball
}
```

In the example above you can see that we give the random() function a range between 0 and the width and height of the canvas. This can only be done after the canvas is defined, otherwise P5.js does not know the width and the height.

You can try to run the example multiple times and see how the ball is drawn at a different random position each time.

12. Functions

The random function we just saw is a small sample from the functions that are readily available in P5.js. But you can also make your own, which is desired if you want to re-use your code. A JavaScript function looks like this:

```
function myFunction(argument1, argument2, argument3) {
    //some code to be executed
}
```

As you can see, a function is defined with the `function` keyword. Just like variables, functions have a unique name as well, myFunction in this case. The name is then followed by a pair of parentheses `()` in which arguments can be specified. These arguments are the variables that are passed to the function in the case the function needs these variables for some application. This is not always needed in which case the parentheses are left empty. Finally, you need to define the scope of the function, which is between the curly brackets `{}`. Between these brackets there is the code to be executed and local variables are declared.

Following are an example of a function that takes no arguments and a function that does:

```
function helloWorld() {
      print('Hello world!');
}

function multiply(a, b) {
      var answer = a * b;
      print(answer);
}
```

To use a function, you can simply call it by its name, followed by a pair of parentheses. If a function requires one or more arguments, make sure to pass these between the parentheses:

```
helloWorld(); // simply prints 'Hello world!'

multiply(3, 4); // prints out 12;
```

Finally, a function can also return an answer. We have already seen such a thing with the `random()` function, which returns a random value. This is done with the `return` statement:

```
function multiply(a, b) {
      var answer = a * b;
      return answer;
}

var c = multiply(3, 4); // c stores the result of the called function: 12
```

The `return` statement stops execution of the function and returns the value specified. The `return` statement can also be used to stop execution of a function without returning an answer:

```
function checkOne(i) {
      if(i == 1) {
            return; // do nothing and exit
      } else {
            print('Not one!'); // print 'Not one'
      }
}
```

Now let’s make a function that draws a purple ball on a specified position for us:

```
// Draw a purple ball
function drawBall(posX, posY) {
  fill(200, 30, 255); // Make it purple
  ellipse(posX, posY, 50, 50); // Draw the ball
}
```

Our resulting program now looks something like this:

```
var x; // a global variable holding the x-position
var y; // a global variable holding the y-position

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");

  x = random(0, width);
  y = random(0, height);
}

function draw() {
  background(220);
  print("Hello world!");

  drawBall(x, y); // Draw a ball

}

// Draw a purple ball
function drawBall(posX, posY) {
  fill(200, 30, 255); // Make it purple
  ellipse(posX, posY, 50, 50); // Draw the ball
}
```

13. Arrays

Before we continue with our little game, we first need to take a look at arrays. Arrays are special variables which can hold multiple values at a time. This is convenient if you have a list of items, values, or properties and you don't want to store each in a single variable. This way you don't have to make a list of variables like this:

```
var sport1 = 'Football';
var sport2 = 'Tennis';
var sport3 = 'Golf';
```

But you can just do this:

```
var sports = ['Football', 'Tennis', 'Golf'];
```

Now we can use the data in our array. This works just like in Processing where you ask for the value using its index, where the first element is at index 0:

```
var sport = sports[0]; // Returns ‘Football’
```

Just like that, you can also change an element inside an array. You just have to re-assign it:

```
sports[0] = 'Hockey';
```

Sometimes, it is also useful to know the length of an array; which is the amount of elements in it. This is done with the length property:

```
print(sports.length); // prints '3'
```

Now we can add elements to our array, making it bigger. This can be done in two ways: using push or specifying the index, which is the length of the array:

```
sports.push('Volleyball');

sports[sports.length] = 'Volleyball';
```

Now, we can use for loops to iterate through our array:

```
for(var i = 0; i < sports.length; i++) {
      pint(sports[i]); // print out the name of each sport
}
```

Finally, we need to be able to delete elements from our arrays. If we want to delete the last element in the array, we can use the `pop()` method. This method removes the last element of the array and returns it:

```
sports.pop(); // Simply removes the last element of the array

var lastElement = sports.pop() // Removes the last element and stores it in
lastElement
```

It might also be desired to remove an element from the array which is on a different position, or even a handful of elements. This can be done with the `splice()` method:

```
sports.splice(2,1); // Remove the third element
```

In the line above, the first parameter, 2, states the position of the first element to be removed. The second parameter, 1, states how many parameters need to be removed.

There are many other methods that can be used on arrays and even the methods explained above can have other uses as well. If you want to learn more about this, you can go to this link.

14. Objects (many of them!)

Now we can really continue making our game, but only one ball does not make our game. If we want multiple balls to be drawn on the screen, and we want each ball to appear randomly, it might be a good idea to make them as objects. An object is basically a type of data, just like a variable is an object. An object may look like this:

```
// The constructor of the ball object
function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Draw a purple ball
  this.display = function() {
    fill(200, 30, 255); // Make it purple
    ellipse(this.x, this.y, this.size, this.size); // Draw the ball
  }
}
```

The code above defines what our ball object is: it has an x and y position, a size specified when created, and a display method, which draws the ball. Now we can create an instance of a ball using the keyword new:

```
var myBall= new Ball(50);
```

This line reserves memory for a ball object and executes the constructor. It takes the value passed as an argument and puts it inside a size variable. The constructor then initializes an x and y position based on a random number between 0 and the dimensions of the canvas. Because the size of the canvas is defined in the setup(), we can only call the constructor of the ball after the definition of the canvas.

Inside the ball class, you can also find the keyword `this` in the variable and function declarations. This keyword connects the variable or method to the class it is part of. It means that *this* property refers to *this* object and is thus a property of *this* object.

Now if we want to draw a ball, we can simply do that using the following line:

```
myBall.display();
```

It is like calling a method, but first telling that it's a method that is executed by a certain object. Therefore you first have to tell which instance of an object the method is called of. This works for calling any method of an object: call its name and tell it what to do.

```
var myBall; // Declare a Ball object

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
  myBall= new Ball(50); // Initialize the Ball object
}

function draw() {
  background(220);
  print("Hello world!");

  myBall.display(); // Display the ball
}

function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Draw a purple ball
  this.display = function() {
    fill(200, 30, 255); // Make it purple
    ellipse(this.x, this.y, this.size, this.size); // Draw the ball
  }
}
```

If you compare the code above to the latest example in section 13, you can see that the declaration and initialization of the x and y position have moved into the ball class, and there is a method that specifies how the ball is displayed. Finally, we can now define the size of the ball by passing it to the constructor and stored in a size variable.

Drawing one ball is fun, but it would be more fun if our myBall object has some friends. Let's make another one using the ball class, but give it a bigger size:

```
var myBall; // Declare a Ball object
var mySecondBall; // Declare another Ball object

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
  myBall = new Ball(50); // Initialize the Ball object
  mySecondBall = new Ball(100); // Initialize the other Ball object

}

function draw() {
  background(220);
  print("Hello world!");

  myBall.display(); // Display the ball
  mySecondBall.display(); // Display the other ball
}
```

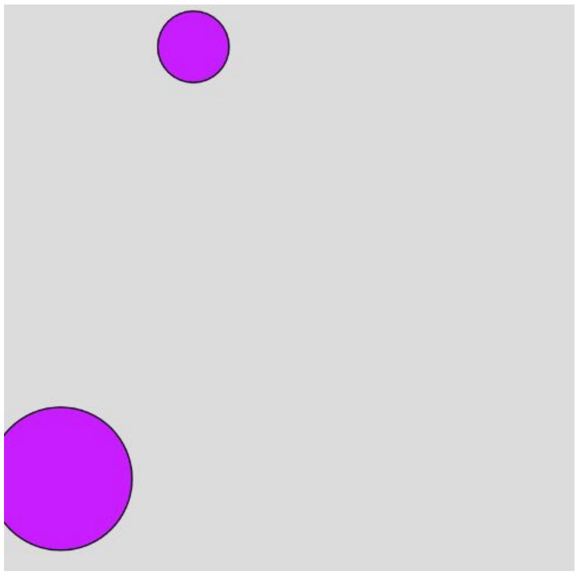


Figure 12: drawing two ball objects with a different size

Now as shown in Figure 12, there are two purple balls on the canvas: a bigger one and a smaller one!

But what if we want more, say 10, with the same size without retyping too much code? Well, like we can make arrays of variables, we can also make arrays of classes.

This is done by first declaring an empty global array that will contain the balls, and in setup we will fill it using a for loop. Then, in draw we can call the display of all ball objects using a for loop as well:

```
var balls = []; // Declare an empty array of Ball objects

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
  for(var i = 0; i <10; i++) {
    balls.push(new Ball(50));
  }
}

function draw() {
  background(220);
  print("Hello world!");

  for(var i = 0; i < balls.length; i++) {
    balls[i].display();
  }
}
```

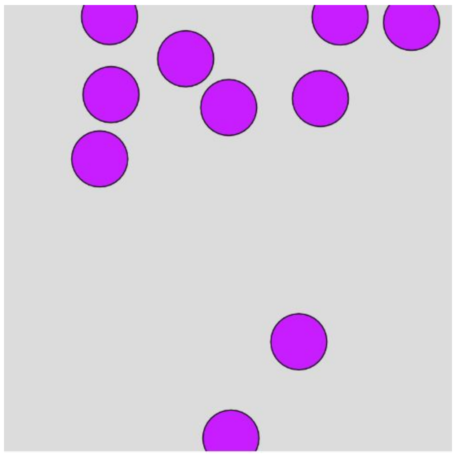


Figure 13: drawing 10 ball objects

As you can see in Figure 13, you can now print out 10 identical ball objects with very little code.

15. if()

Because we want to make a reaction-based game where you have to click a purple ball before a certain time, we need to be able to check the age of a ball and make it disappear when it has reached a certain age. We can do such a check with the `if()` statement.

The `if()` statement looks as follows:

```
if(boolean condition) {
      // code to be executed
}
```

This block of code makes sure that a certain condition is met, the specified code is executed.

Such a condition is specified with comparison operators and maybe logical operators:

Comparison operator	**Description**
==	Equal to
===	Equal value and equal type
!=	Not equal
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator

Logical operator	**Description**
&&	Logical and
||	Logical or

!	Logical not

So a real if statement may look like this:

```
if(age >= 18 && hasDriversLicense) { // If your age is 18 or older and you
have a driver's license
      print('You may drive a car');
}
```

Now what if we want something to happen when the condition is met and we want something else to happen when the condition is not met? For that we can use an `if else` statement:

```
if(age >= 18 && hasDriversLicense) { // If your age is 18 or older and you
have a driver's license
      print('You may drive a car');
} else { // If your age is younger than 18
      print('You are not allowed to drive yet');
}
```

And now what if we want to have even more options than two? We can use the `else if` statement to specify another condition if the first condition checks out false:

```
if(age >= 18 && hasDriversLicense) { // If your age is 18 or older and you
have a driver's license
      print('You may drive a car');
} else if(hasDriversLicense) { // If your age is younger than 18, but you
have a license
      print('You are not allowed to drive without supervision yet');
} else {
      print('You are not allowed to drive yet');
}
```

(in the Netherlands, you can get your driver's license at 17, but you need the supervision of someone else to drive)

So, now that we have discussed the `if`, `if else` and `if else if` statements, we can check if our ball needs to disappear after a certain time.

We do this using the [`millis()`] function. This function returns the number of milliseconds since starting the program.

var timeOfBirth = millis();

The statement above takes the current amount of milliseconds that has passed since starting the program and stores it in the timeOfBirth variable. The value of this variable does not automatically change as time passes, the value stays the same as long as it is nut updated by setting it again.

We also specify a time to live of 2 seconds, which is 2000 milliseconds:

var timeToLive = 2000;

The above described variables are both properties of the ball object, so we will put them inside the constructor of our ball class:

```
function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Timing properties
  this.timeOfBirth = millis();
  this.timeToLive = 2000;

  // Draw a purple ball
  this.display = function() {
    fill(200, 30, 255); // Make it purple
    ellipse(this.x, this.y, this.size, this.size); // Draw the ball
  }
}
```

Now we can compare the time that each ball object exists (current time - the time of birth) with its time to live, and if the time of existence is bigger than or equal to the time to live, we delete the ball object:

```
for(var i = 0; i < balls.length; i++) { // run the code for all balls in our
array
    balls[i].display(); // display the ball
    if (millis() - balls[i].timeOfBirth >= balls[i].timeToLive) { // check if
the ball has become too old
      balls.splice(i, 1); // delete the ball if the ball is too old
    }
}
```

Now all balls appear after 2 seconds and do not come back. Let’s add code that will randomly make a ball appear all the time, while we can not have more than 3 balls at the same time:

```
  if (balls.length < 3 && random(0, 100) < 2) {
    balls.push(new Ball(50));
  }
```

The code above first checks if the length of the balls array is smaller than 3, we then add some randomness to the function by randomly generating a number from 0 up to 100, now if this number is smaller than 2, we make a new ball object. Thus, every time this if statement is executed and the balls array is not too big, there is about a 2% chance that a new ball is created and added to the array.

The full program that we have now looks like this:

```
var balls = []; // Declare an empty array of Ball objects

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);

  if (balls.length < 3 && random(0, 100) < 2) {
    balls.push(new Ball(50));
  }

  for (var i = 0; i < balls.length; i++) { // run the code for all balls in
our array
    balls[i].display(); // display the ball
    if (millis() - balls[i].timeOfBirth >= balls[i].timeToLive) { // check if
the ball has become too old
      balls.splice(i, 1); // delete the ball if the ball is too old
    }
  }

}

function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Timing properties
  this.timeOfBirth = millis();
  this.timeToLive = 2000;

  // Draw a purple ball
  this.display = function() {
    fill(200, 30, 255); // Make it purple
    ellipse(this.x, this.y, this.size, this.size); // Draw the ball
  }
}
```

16. Events

Next we want to be able to click on a ball to make it disappear and print out our reaction time and whether that is the new highscore.

We first need to be able to detect a mouse click. This is done with a `mouseClicked()` function. This function is called once after a mouse button has been clicked (pressed and then released). Web Browsers may handle these mouse clicks differently, causing this function to only detect a click from the left mouse button. For more information check the reference page.

The function mouseClicked() is used to detect a mouse click as follows:

```
function mouseClicked() {
      print('You clicked the mouse!');
}
```

We can also retrieve the location of the cursor relative to the canvas using mouseX and mouseY:

```
var x = mouseX; // Save the mouse x position
var y = mouseY; // Save the mouse y position
```

In order to check whether a ball has been clicked, we need to compare the distance of the cursor at the time of a click to the radius (size) of each ball. If this distance is smaller than the radius, we can guarantee that the ball has been clicked and we can print out the reaction time (the current time - the time of birth of the ball) in seconds and we can delete the ball:

```
function mouseClicked() {
  var x = mouseX; // Save the mouse x position
  var y = mouseY; // Save the mouse y position
  for (var i = 0; i < balls.length; i++) { // Check which ball has been
clicked
    var distance = dist(x, y, balls[i].x, balls[i].y); // Get the distance
between the cursor and the origin of the ball

    if (distance < balls[i].size) { // Compare the distance with the radius

      var reactionTime = (millis() - balls[i].timeOfBirth) / 1000; // Get the
reaction time
      print("Clicked on shape within ", reactionTime, " seconds!");

      balls.splice(i, 1); // Delete the ball from the array
    }
  }
}
```

As you can see, the distance between the cursor and the origin of the ball is calculated with the function `dist()`. This function is used to calculate the distance between two points, so we give it the x and y position of the cursor, and the x and y position of the ball.

The mouseClicked() event is only a single example of the many types of events that can occur. There are also other mouseEvents such as mouseMoved(), keyboard events such as keyPressed(), and events for smartphones and other touch devices with or without an accelerometer. For more information, take a look at the Events section in the P5.js reference guide.

Now finally, we want to have a high score, so we make a global variable that stores the high score. It might also be nice to have a variable that stores the latest reaction time for later use, called score:

```
var highscore;
var score;
```

Then when a ball is clicked, we want to check the reaction time and if it is lower than the high score, we set the reaction time as the new high score and we set the given reaction time as the latest reaction time. Let’s practice functions one more time and make a function that does this for us:

```
function setScore(time) {
  score = time; // store the reaction time as the latest reaction time
  if (highscore == null || time < highscore) { // set the highscore only if
there is none of the reaction time is lower
    highscore = time;
    print('You have a new high score!'); // Tell the player about his
achievement
  }
}
```

We call the function `setScore()` after we have calculated and printed the reaction time, making our program now look like this:

```
var balls = []; // Declare an empty array of Ball objects
var highscore;
var score;

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");
}

function draw() {
  background(220);

  if (balls.length < 3 && random(0, 100) < 2) {
    balls.push(new Ball(50));
  }

  for (var i = 0; i < balls.length; i++) { // run the code for all balls in
our array
    balls[i].display(); // display the ball
    if (millis() - balls[i].timeOfBirth >= balls[i].timeToLive) { // check if
the ball has become too old
      balls.splice(i, 1); // delete the ball if the ball is too old
    }
  }
}

function mouseClicked() {
  var x = mouseX; // Save the mouse x position
  var y = mouseY; // Save the mouse y position

  for (var i = 0; i < balls.length; i++) { // Check which ball has been
clicked
  var distance = dist(x, y, balls[i].x, balls[i].y); // Get the distance
between the cursor and the origin of the ball
```

```
    if (distance < balls[i].size) { // Compare the distance with the radius

      var reactionTime = (millis() - balls[i].timeOfBirth) / 1000; // Get the
reaction time
      print("Clicked on shape within ", reactionTime, " seconds!");
      setScore(reactionTime); // Compare the score with the high score

      balls.splice(i, 1); // Delete the ball from the array
    }
  }
}

function setScore(time) {
  score = time; // store the reaction time as the latest reaction time
  if (highscore == null || time < highscore) { // set the highscore only if
there is none of the reaction time is lower
    highscore = time;
    print('You have a new high score!'); // Tell the player about his
achievement
  }
}

function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Timing properties
  this.timeOfBirth = millis();
  this.timeToLive = 2000;

  // Draw a purple ball
  this.display = function() {
    fill(200, 30, 255); // Make it purple
    ellipse(this.x, this.y, this.size, this.size); // Draw the ball
  }
}
```

We actually have a fully working game now, but it is still not very visually pleasing. The game still needs some text displaying the score and high score, some images to make the balls prettier, and some sound, to make clicking a ball more fun!

17. Text

Now let's add some text to the game. We want to be able to display the latest reaction time and the high score: the lowest reaction time. P5.js has many functions that are used to display text, but today we will only use the function `text()` and `textSize()`.

Let's make a function called `writeScore()` that displays the latest reaction time and the fastest reaction time. In order to keep track of the latest reaction time, we have already made a global variable that stores this, called score.

There is one small problem: as you might have seen, the reaction time is printed in seconds, but with a lot of decimals behind the comma. This uses a lot of space and is not necessary. We can adjust the way a number is printed as a string by first converting the float variable to a string variable using the function `str()`. This function converts any variable to its string representation.

We then use the function `nf()` to decrease the amount of numbers behind the comma and we store the result in a new variable for each line of text we want to write:

```
var hsText = "Highscore: " + nf(str(highscore), 1, 3) + "s";
var scoreText = "Reaction time: " + nf(str(score), 1, 3) + "s";
```

The function nf() takes three arguments: the number to format as a string, the number of digits left of the comma and the number of digits right of the comma. Because a ball only exists for two seconds, the amount of seconds can never be higher than 10, so we only need one number after the comma. To know the score at least a bit precise, we use the value three to specify that we want three numbers right of the comma.

Now there is another problem: if the reaction time and the high score are undefined, it will print this. We therefore only want to display these times if we actually have something to display. We can do this by checking whether there actually is a high score, because if there is a high score, there is a normal score as well. This is done by checking if the highscore variable is not equal to a null object. If a variable contains this null object, it means that the variable has not been initialized and has thus been undefined, as described in section 10. This check looks as follows:

```
  if (highscore != null) { // Check if there is a high score, which also
exists if there is a score
      // Display the text
  }
```

Now that we have made sure we only display something when there is a score to display, we want to display the texts on two seperate lines with font size 36 using textSize() in black on our canvas:

```
textSize(36); // Make the text large enough
fill(0); // Make the text black
text(scoreText, 10, 50); // First write the text for the latest reaction time
text(hsText, 10, 100); // Then write the high score
```

The resulting writeScore() function now looks like this:

```
function writeScore() {
  var hsText = "High score: " + nf(str(highscore), 1, 3) + "s";
  var scoreText = "Reaction time: " + nf(str(score), 1, 3) + "s";

  if (highscore != null) { // Check if there is a high score, which also
exists if there is a score
    textSize(36); // make the text large enough
    fill(0); // make the text black
    text(scoreText, 10, 50); // First write the text for the latest reaction
time
    text(hsText, 10, 100); // Then write the high score
  }
}
```

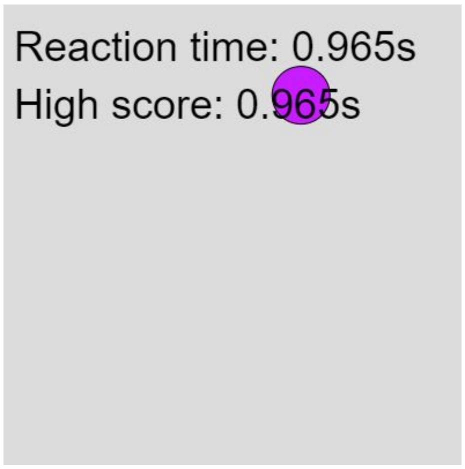


Figure 14: drawing text on the canvas

You can now call this function on the final line inside your draw function to execute it and display the score, as shown in Figure 14.

18. Images

Now we want to make the ball objects to look a bit more fun and give them an interesting image instead of just a purple color. How about a bell? We will use the image shown in Figure 15.

Figure 15: the image of a bell we are going to display

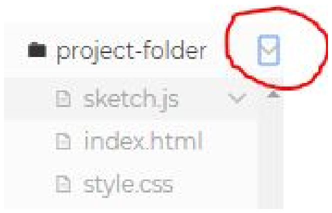


Figure 16: the indicated button to add a file

We now need to start working with assets, which are files we can upload to the editor to work with. To upload a file, press the button indicated in figure 16 and simply add the file and give it a name to work with or simply drag and drop it. The file I am using is called ding.png.

Because it is not necessary to reload the file each time a ball object is made, we want to make a global variable that stores this file, then load it in the setup into this variable and use it when we want to display a ball. Loading an image is done with the function `loadImage(image_path)` and displaying an image is done with the function `image(img_variable, x_position, y_position, horizontal_size, vertical_size)`:

```
var img; // A variable that stores the image for each ball
img = loadImage("assets/Ding.png");  // Load the image
image(img, this.x, this.y, this.size, this.size);  // Display the ball
```

So we want to add the above three lines of code to our global variables, the second line inside our setup, and replace the lines that draw a purple ball inside the ball.display() function with the third line:

```
var img; // A variable that stores the image for each ball
```

```
function setup() {
  createCanvas(400, 400);
  alert("Good luck!");

  img = loadImage("assets/ding.png");  // Load the image
}

function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Timing properties
  this.timeOfBirth = millis();
  this.timeToLive = 2000;

  // Draw a ball
  this.display = function() {
    image(img, this.x, this.y, this.size, this.size); // Display the ball
  }
}
```

Please note that the above code is not the full program, but just the seconds that contain adjusted code.

There are many more operations that you can do with images. Please take a look at the reference guide to learn more.

19. Sound

Now, if we want to make the balls more fun to click on, we can let the program play a “ding!” sound. P5.js has a very complete sound library of which more information is available in the reference guide. For our game, we only need the functions `soundFormats()` to specify which format we are using, `loadSound()` to load the sound during setup, and `play()` to actually play the sound.

We need to add three more lines to our code:

In setup():

```
soundFormats('mp3'); // Specify that we are using the mp3 format
ding = loadSound('assets/ding.mp3'); // Load the ding sound
```

After a ball has been clicked:

```
ding.play(); // Play the ding sound
```

As you can see, it is not necessary to declare a variable for the sound object. Instead, it returns a p5.SoundFile.

Our reaction game is now completely finished and looks as follows:

```
var balls = []; // Declare an empty array of Ball objects
var highscore;
var score;
var img; // A variable that stores the image for each ball

function setup() {
  createCanvas(400, 400);
  alert("Good luck!");

  img = loadImage("assets/ding.png");  // Load the image

  soundFormats('mp3'); // Specify that we are using the mp3 format
  ding = loadSound('assets/ding.mp3'); // Load the ding sound
}

function draw() {
  background(220);

  if (balls.length < 3 && random(0, 100) < 2) {
    balls.push(new Ball(50));
  }

  for (var i = 0; i < balls.length; i++) { // Run the code for all balls in
our array
    balls[i].display(); // Display the ball
    if (millis() - balls[i].timeOfBirth >= balls[i].timeToLive) { // Check if
the ball has become too old
      balls.splice(i, 1); // Delete the ball if the ball is too old
    }
  }
  writeScore();
}
```

```
function mouseClicked() {
  var x = mouseX; // Save the mouse x position
  var y = mouseY; // Save the mouse y position

  for (var i = 0; i < balls.length; i++) { // Check which ball has been
clicked
    var distance = dist(x, y, balls[i].x, balls[i].y); // Get the distance
between the cursor and the origin of the ball

    if (distance < balls[i].size) { // Compare the distance with the radius
      ding.play(); // Play the ding sound

      var reactionTime = (millis() - balls[i].timeOfBirth) / 1000; // Get the
reaction time
      print("Clicked on shape within ", reactionTime, " seconds!");
      setScore(reactionTime); // Compare the score with the high score

      balls.splice(i, 1); // Delete the ball from the array
    }
  }
}

function setScore(time) {
  score = time; // store the reaction time as the latest reaction time
  if (highscore == null || time < highscore) { // set the highscore only if
there is none of the reaction time is lower
    highscore = time;
    print('You have a new high score!'); // Tell the player about his
achievement
  }
}


function writeScore() {
  var hsText = "High score: " + nf(str(highscore), 1, 3) + "s";
  var scoreText = "Reaction time: " + nf(str(score), 1, 3) + "s";

  if (highscore != null) { // Check if there is a high score, which also
exists if there is a score
    textSize(36); // make the text large enough
    fill(0); // make the text black
    text(scoreText, 10, 50); // First write the text for the latest reaction
time
```

```
    text(hsText, 10, 100); // Then write the high score
  }
}

function Ball(size_) {
  this.size = size_;
  this.x = random(0, width);
  this.y = random(0, height);

  // Timing properties
  this.timeOfBirth = millis();
  this.timeToLive = 2000;

  // Draw a ball
  this.display = function() {
    image(img, this.x, this.y, this.size, this.size); // Display the ball
  }
}
```

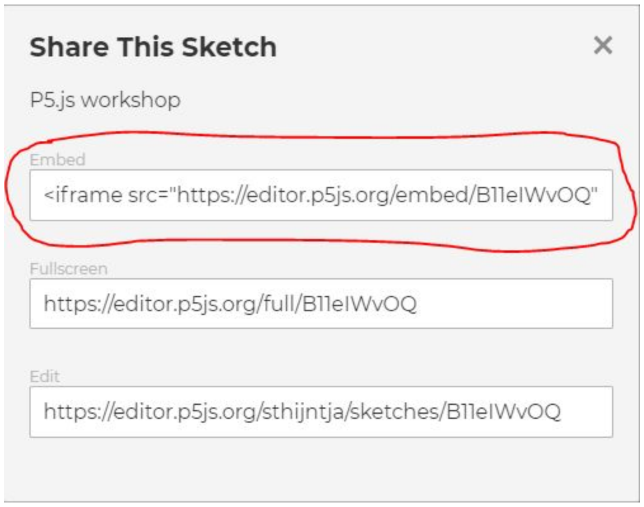
20. Uploading the sketch to your website

Now that you have a nice little game, you can upload it to your website for the whole world to see. Unfortunately, if you are working with assets, you can not run the sketch directly on your own machine. You can only run it from a web server, so that means you first have to upload it to your website and go to the webpage that shows your sketch before you can run it. You can solve this problem by setting up a local web server using wamp.

There are multiple ways to put the sketch on your website by either sharing the sketch or downloading it: you can link to the fullscreen version of the sketch, like this is the link to mine, embedding the sketch as an iframe in your html, or by downloading all files and uploading them to your website, adjusting the html as you like.

I will only explain embedding the sketch, but feel free to experiment downloading the sketch, changing the webpage and uploading it to your website.

The p5.js online editor provides a piece of html code for you to easily embed your sketch. You can get this code by pressing the share button in the file tab:


Share This Sketch

P5.js workshop

<iframe src="https://editor.p5js.org/embed/B11eIWvOQ"

https://editor.p5js.org/full/B11eIWvOQ

https://editor.p5js.org/sthijntja/sketches/B11eIWvOQ

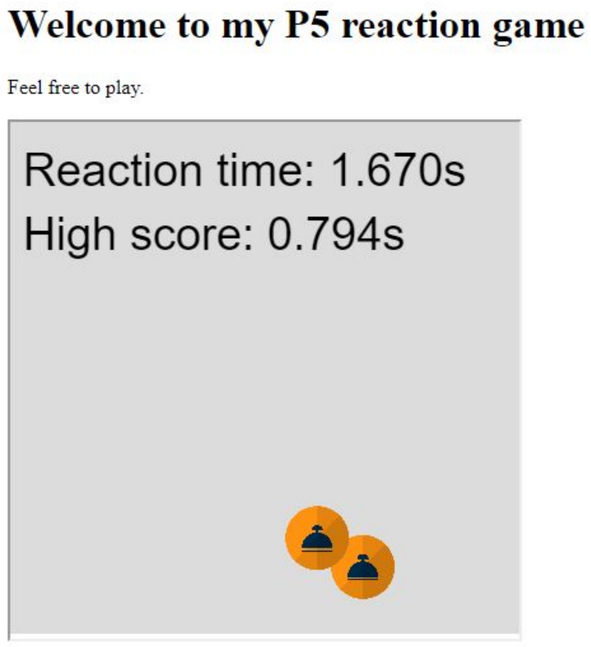
Figure 17: the share window in the p5.js online editor

Figure 17 shows where to find the code to copy to your HTML file:

```
<iframe src="https://editor.p5js.org/embed/B11eIWvOQ"></iframe>
```

You can then copy this code into your html file, which you can then upload to your own website:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>Welcome to my P5 reaction game</h1>
        <p>Feel free to play.</p>
        <iframe src="https://editor.p5js.org/embed/B11eIWvOQ" width="400"
height="404"></iframe>
    </body>
</html>
```



Welcome to my P5 reaction game

Feel free to play.

Reaction time: 1.670s
High score: 0.794s

Figure 18: The resulting webpage with the sketch embedded

As you can see in Figure 18, the size of the iframe is about as big as the size of the canvas. Except the height has been made slightly larger to get rid of a scrollbar. The resulting webpage looks as follows:

21. Practice makes perfect

Now that you have made your own reaction game in P5.js and have uploaded it to your site, you are not done learning P5.js! You have only learned about a small selection of tools that are available to make a program, but you need to do more programming in order to master it. Get inspired and make another game or try to do something cool with visuals and sound. I guess what I am trying to say is that practice makes perfect.