11 - Events

p5.js Events

WE'LL COVER THE FOLLOWING

- Intro & mousePressed
- keyPressed
- Summary
- Practice

Intro & mousePressed

In the earlier chapters, we learned about a p5.js variable called **mouseIsPressed**, which assumes the value **true** when the mouse is clicked and **false** for all other instances.

We also learned that this is not a great way of capturing user input as the execution speed of the **draw** function can make it hard to get this variable triggered in a reliable way. That's why there are other ways of handling user input inside p5.js, namely the *events*, which solves this problem.

Using events, we can capture the user input outside the **draw** function loop.

There are numerous event functions in p5.js that we can declare to make use of the event system. One of those functions is the **mousePressed** function.

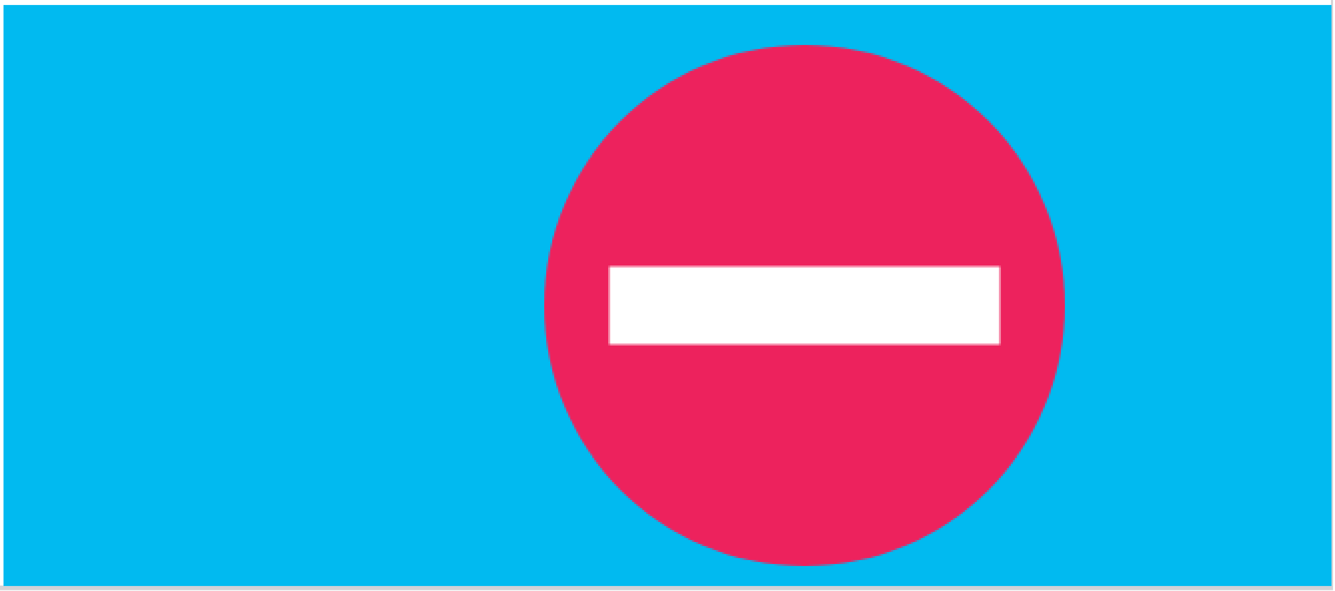
The idea is similar to the **draw** and **setup** functions where we will be declaring this function with this particular name which is treated by p5.js in a special manner (just like **setup** and **draw** functions are).

In a p5.js code, the function that we declare under the name **mousePressed** gets triggered every time the mouse button is clicked. Let's re-write our previous example that makes use of the **mouseIsPressed** variable to make use of **mousePressed** event function instead.

Output

JavaScript

HTML



Well, that was a simple refactor! We are simply declaring a function that we don't execute ourselves. The execution of it is handled by p5.js whenever the corresponding action takes place.

There are lots of other event functions. A complete list can be found under https://p5js.org/reference/#group-Events

keyPressed

One other event function that might be worth learning about is the **keyPressed** function. As the name implies **keyPressed** function gets triggered every time a key is pressed. Let's quickly test how it works in a brand new sketch.

Output

JavaScript

HTML

Console

Clear

In this example, every time we press a key we would see a message ‘pressed’ displayed in the console. Let’s look at a more involving example where pressing a key each time creates a shape in the canvas.

Output

JavaScript

HTML

The shapes are created after we press a key.

Notice a couple of things. First of all, we moved the **background** function to be under the **setup** function. This is to ensure that the shapes we draw remain on the screen. If we are to have a **background** function call inside the **draw** function, then it would paint over everything, on every frame which is not desirable for this use case. Also, we are spreading the **ellipse** function call over a couple of lines again to increase legibility.

We have a global variable called **pressed**. With each key press, we are setting the value of this global variable to be **true**. When this happens, the **draw** function renders an **ellipse** to the screen since the conditional statement gets executed. Then the **draw** function immediately sets the **pressed** value to **false** again so that we only get one ellipse.

We are going to improve this example a little bit to make it more pleasant to the eye. Currently, the circles are looking a bit too uniform, and the colors are a bit too dull. We will make it so that every time we are creating a circle, it uses a random size between 0 and 200 and a random color from a list of predefined random colors.

Output

JavaScript

HTML

To be able to select a random color each time a key is pressed, we need to generate a random **integer** in between 0 and the length of the colors array minus 1. Minus 1 because array indices start counting from 0.

To generate any random number in between 0 and the length of the array minus 1, We can simply write the random function as `random(colors.length)`. This would end up generating a number in between 0 and up until the number of items in **colors** array (excluding that number). The problem though is that the number being generated is a floating point number meaning it has decimal places. Whereas, we need an integer number to be able to access items in an array. So we need to convert the decimal number into a whole number. There are a couple of ways to solve this. One way could be to use the p5.js **floor** function, which rounds down the given floating point number to the nearest integer. Another solution could be to use the native JavaScript function called **parseInt** which converts a given value into an integer - if the value can be converted. We can't expect to throw a string name value to it and receive an integer.

We need to pass a second parameter to **parseInt** function to set the number base that the calculation will happen. That base almost always is 10. Using **parseInt** function on a float number looks something like this.

```
var num = parseInt(0.5, 10);
console.log(num); // will be 0.
```

Identifying the pressed key is only part of the issue, though. Another thing that we should be able to do is to identify which button the user pressed. Inside the **keyPressed** function, we could theoretically identify any key pressed by using the **keyCode** variable. **keyCode** variable holds the last key that the user pressed in an encoded manner. Such that if the user pressed the key 'a', it would return the value '65', for 'b'; '66', etc...

p5.js being a helpful library makes it easier to identify some of the keys by providing pre-defined variables for them, like: **BACKSPACE**, **DELETE**, **ENTER**, **RETURN**, **TAB**, **ESCAPE**, **SHIFT**, **CONTROL**, **OPTION**, **ALT**, **UP_ARROW**,

RETURN, **TAB**, **ESCAPE**, **SHIFT**, **CONTROL**, **OPTION**, **ALT**, **UP_ARROW**, **DOWN_ARROW**, **LEFT_ARROW**, **RIGHT_ARROW**.

For example here is a small code snippet that executes a **console.log** statement whenever the 'Enter' key is pressed.

Output

JavaScript

HTML

Console

Clear

Using the **keyCode** variable we could identify which alphanumeric key is pressed with a bit of decoding. But there is another variable that works specifically well for the alphanumeric characters and that is called **key**. **key** variable stores the value of the alphanumeric key that is pressed as is, so it makes it easier to identify which key was pressed.

Summary

In this chapter, we learned about a better way to handle events, and that is event functions.

We particularly focused on two event functions that are: **mousePressed** and **keyPressed** event functions.

keyPressed event functions.

We also learned about some of the variables that we can use inside the **keyPressed** function, that is **key** and **keyCode**. **key** makes identifying the alphanumeric key presses easier whereas **keyCode** is ideal for detecting other key presses as it can be compared against p5.js variables such as **ENTER**, **SPACE**, etc. that makes identifying those buttons easier.

From the JavaScript part of the things we learned about the **parseInt** function that can be used to convert number-like values (which includes strings that represent a number as well) into an integer number.

Practice

Draw a rectangle to the screen where the keyboard arrow jeys can control the position of the rectangle.