JavaScript Events I

John M. Morrison

August 6, 2018

Contents

0 Introduction

By and large, the *raison d'etre* for JavaScript is to make web pages interactive. We have all seen pages that respond to interaction with page elements such as menus or buttons, and behavior induced by the mouse entering, clicking, dragging, or leaving page elements. What we often notice is that communication is made between page elements. For example, you might slide a slider and have an element on the page change color, or you might push a button and have text appear on the page. So, how we establish this communication channel?

This is done via the mechanism of events, which will be the subject of this chapter and the next. You have already learned that, when a page is done loading, the browser broadcasts a `load` event. Also, when a page is being

exited, the browser broadcasts an `unload` event. These are browser-spawned events. When we created the tag

```
<body onload="init();">
```

what we did is to attach an *event listener* to the `body` element that would call the function `init` when the body hears a broadcast `load` event.

Events are objects; depending on the source, they contain information about themselves. For example keyboard events can tell you what key was hit, and mouse events can tell you where they occur on the screen.

Some events are discrete, in that they happen in a single moment. An example of this is the event of mouse being clicked or a button being clicked. Others "poll" and are fired at close intervals over a period of time, such as when a key is pressed and held down. This will broadcast events at a short interval until the user releases the key.

Event handling allows you, the web developer, to respond to these occurrences and to execute code in response to them.

Remember, we have said that what computers do most of the time is to wait for user instruction. Modern computers are event-driven machines; we can use JavaScript event handling to create the kinds of features you would expect on a web page.

We will begin by looking at the basic types of events in JavaScript and then we will see how to use these to create interactivity on web pages.

Event Name	**When it is fired**
`load`	When the page is finished loading
`unload`	When the document is being unloaded
`resize`	When an element is resized

The `unload` event is often used to remind the user that he has not finished his business on a page. For example, changing pages might cause a post to be lost, so the page warns the user with an `alert` box.

The `load` event is fired when all of the page's elements are finished loading. This is important because, if you are using `document.getElementById`, you want the element being fetched to exist before you try to act on it. The most common place you will see it is in the `body` tag.

There are events spawned the mouse. Here is a sampling of some of the most important ones.

Event Name	When it is fired
`mouseenter`	the mouse enters an element with a listener attached
`mouseover`	the mouse is over an element with a listener attached, or one of its children
`mousemove`	the mouse is moving over an element with a listener attached, or one of its children This event is fired continuously (at the browser's polling rate) until the mouse is no longer over the element.
`mousedown`	any mouse button is pressed on an element
`mouseup`	any mouse button is released on an element
`click`	the mouse is clicked on an element
`dblclick`	the mouse is double-clicked on an element

There are also key events, triggered by actions of the user on the keyboard.

Event Name	When it is fired
`keydown`	any key is pressed on an element
`keyup`	any key is released on an element
`keypress`	any key is pressed on an element

1 Making Elements Event-Aware in HTML

Let us begin with a simple HTML element, a button. We will create a page and have its color scheme flop when the button is clicked. Let us begin by creating some HTML in a file called `flip.html`. Notice that, in the header, JavaScript and CSS are included.

```
<!doctype html>
<html>
<head>
<title>flip</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="flip.css"/>
<script type="text/javascript" src="flip.js"></script>
</head>
<body>
    <h2>A Page with a Split Personality</h2>
```

```
    <p class="display"><button onclick="flip();">Go to the Dark Side!</button></p>

    <p>Click on the button and the color scheme on this page will
    invert.</p>

</body>
</html>
```

Next we set the page color scheme with a little CSS.

```
h1, h2, .display
{
    text-align:center;
}
body
{
    background-color:#99BADD;
    color:#001A57;
}
```

Create these two files and open them in a browser tab to see our shiny new page. A button is present on page, but when it is clicked, nothing happens. Now let us talk about the JavaScript. What we want to have happen is for the color scheme to invert when the button is clicked. To do this we will create a "flag," i.e. a boolean to tell if we are using the light color scheme. We will set it to `true` at the start.

```
let light = true;
```

Next we create the function `flip` we have for the `onclick` attribute of the button.

```
let light = true;
function flip()
{
}
```

If `light` is `true`, we will flip the color scheme like so.

```
let light = true;
function flip()
{
    if(light)
    {
        document.body.style.color = "#99BADD";
```

```
        document.body.style.backgroundColor = "#001A57";
    }
}
```

It is now clear what to do when `light` is `false`. Also, after each button push, we should negate `light` so the setup toggles.

```
let light = true;
function flip()
{
    if(light)
    {
        document.body.style.color = "#99BADD";
        document.body.style.backgroundColor = "#001A57";
    }
    else
    {
        document.body.style.color = "#001A57";
        document.body.style.backgroundColor = "#99BADD";
    }
    light = !light;
}
```

Programming Exercise

1. Give the button an id of `flipper`.
2. Use `document.getElementById` to change the text in the button so that when the color scheme is dark, the button says, "Go to the Lighter Side!" and when it is light, the buttons says "Go to the Dark Side!"

2 Achieving Better Separation

You will notice that we had to introduce JavaScript into to our HTML files to get things to work. It is better style to achieve a complete separation between the layers. We did this in CSS by using the `link` tag. So, how do we do this in JavaScript?

We will re-engineer the page we just created to achieve this separation. One consequence of this will be the elimination of the global variable `light`.

You already did this in the programming exercise

```
<button id="flipper" onclick="flip();">Go to the Dark Side!
    </button>
```

Let us get rid of the `onclick` attribute, leaving this.

```
<button id="flipper">Go to the Dark Side!</button>
```

Now, open `flip.js`. Modify it to look like this.

```
let light = true;
let flipper = document.getElementById("flipper");
flipper.addEventListener("click", flip);
function flip()
{
    if(light)
    {
        document.body.style.color = "#99BADD";
        document.body.style.backgroundColor = "#001A57";
    }
    else
    {
        document.body.style.color = "#001A57";
        document.body.style.backgroundColor = "#99BADD";
    }
    light = !light;
}
```

Open the developer tools and look in the console, and you will get this very sad news.

```
flip.js:3 Uncaught TypeError: Cannot read property
    'addEventListener' of null at flip.js:3
```

Uh oh. What happened? Well, we have to remember that pages *load from top to bottom*. When the JavaScript loads in the head of the page, the `button` element we are seeking does not exist. How do we disentangle that? Get a load of this.

```
window.addEventListener("load", init);
function init()
{
    let light = true;
    let flipper = document.getElementById("flipper");
    flipper.addEventListener("click", flip);
    function flip()
    {
        if(light)
        {
```

```
            document.body.style.color = "#99BADD";
            document.body.style.backgroundColor = "#001A57";
        }
        else
        {
            document.body.style.color = "#001A57";
            document.body.style.backgroundColor = "#99BADD";
        }
        light = !light;
    }
}
```

The mysterious first line just says, "Hey, when this page is done loading call the function `init`." This is the same as giving the `body` and `onload` attribute, but it bears the benefit of doing JavaScript in a JavaScript file, not on an HTML page. The symbol `window` is just a name for the global object. Notice how we simply get the button by its `id` and attach a listener that calls `flip` when the button is pushed.

The other new feature here is that you can have a function inside of a function and that the inner function has access to the outer function's symbol table.

Programming Exercise It's time to break out the `for` loop and our old friend `document.getElementsByTagType`.

1. Create a style sheet that gives paragraphs a padding of .25em.
2. Create an HTML page with several paragraphs on it and a button at the that says "Outline all paragraph elements."
3. Add a button that says "Supersize me!" When the user clicks on it, have a padding of 1em put around the paragraphs.
4. When the user clicks on the button, have it put a solid 1px black border around every paragraph.
5. Make the supersizer button a toggle and, when the paragraphs are supersized, have it read "Make me normal sized!"
6. Do a similar thing for the outliner button.

3 Interacting with Text Inputs

JavaScript has a wide array of different inputs; we will begin with text boxes. We will see how to get text from text boxes and place them in elements on our page. Let us begin by creating this style sheet, `text.css`.

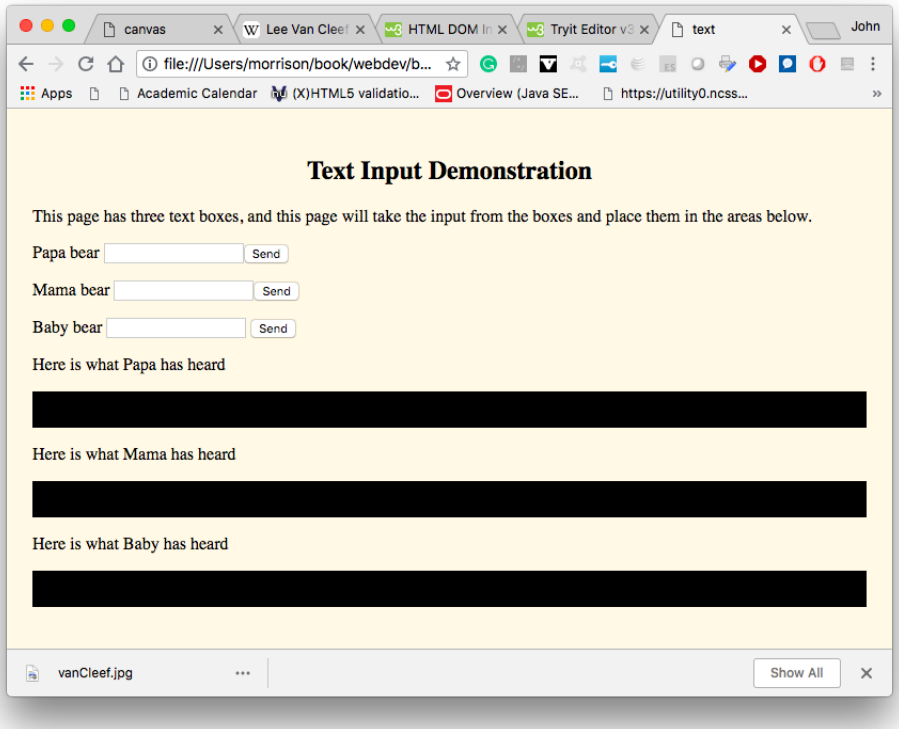
```
h1, h2, .display
{
    text-align:center;
}
#papaHears, #mamaHears, #babyHears
{
    padding:1em;
    border:solid 1px black;
    background-color:black;
    color:lightgreen;
}
body
{
    padding:1em;
    background-color:#FFF8E7;
}
```

Next, we create this HTML file, `text.html`.

```
<!doctype html>
<html>
<head>
<title>text</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="text.css"/>
<script type="text/javascript" src="text.js"></script>
</head>
<body>
    <h2>Text Input Demonstration</h2>

    <p>This page has three text boxes, and this page will take the input from
        the boxes and place them in the areas below.</p>

    <p>Papa bear <input type="text" id="papa"/><button
            id="papaButton">Send</button></p>
    <p>Mama bear <input type="text" id="mama"/><button
            id="mamaButton">Send</button></p>
    <p>Baby bear <input type="text" id="baby"/> <button
            id="babyButton">Send</button></p>

    <p>Here is what Papa has heard</p>
    <div id="papaHears"></div>
    <p>Here is what Mama has heard</p>
    <div id="mamaHears"></div>
    <p>Here is what Baby has heard</p>
    <div id="babyHears"></div>
```

```
</body>
</html>
```

Open the file `text.html` in your browser to reveal this.



Since we have appropriate `id`s for everything, we no longer need to edit the HTML or CSS files. It's time to create the JavaScript. Begin with this.

```
window.addEventListener("load", init);
function init()
{
}
```

Notice that the method `addEventListener` takes two arguments. The first is the event type, and the second is a function that is called when that event occurs. What we are about to see is that we can create an anonymous function, on-the-fly, for an event listener to call.

But, before that, let us assemble all of the page elements our JavaScript will modify.

```
window.addEventListener("load", init);
function init()
{
    //round up our widgets
    let mama = document.getElementById("mama");
    let papa = document.getElementById("papa");
    let baby = document.getElementById("baby");
    let mamaHears = document.getElementById("mamaHears");
    let papaHears = document.getElementById("papaHears");
    let babyHears = document.getElementById("babyHears");
    let mamaButton = document.getElementById("mamaButton");
    let papaButton = document.getElementById("papaButton");
    let babyButton = document.getElementById("babyButton");
}
```

Here is the desired action. When we click on one of the send buttons, the text is sent from that bear's text box, to its "heard" box and the text in the bear's box is cleared. Each time, let's put a newline at the end of the sent text.

Question is, "How do we get the text out of the box?" Happily, a text input box has a property named `value`, which has the text. So, `mama.value` will give us the text in Mama Bear's box. We will take that text, stick a newline on the end of it, and add it to the text in the `mamaHears` div. The code will look like this.

```
mamaHears.innerText +=  mama.value + "\n";
mama.value = "";
```

The first line grab's the text in `mama` and appends it, along with a newline to the `mamaHears` box. The second line clears the `mama` box so the user does not have to backspace over old text.

Now we must attach this behavior to the `mamaButton`. We could write a separate function and then use it as the second argument to `addEventListener`. But why introduce this clutter. Instead we will create an anonymous function like so.

```
mamaButton.addEventListener("click", function()
{
    mamaHears.innerText +=  mama.value + "\n";
    mama.value = "";
});
```

You can also do this.

```
mamaButton.addEventListener("click", () =>
{
    mamaHears.innerText +=  mama.value + "\n";
    mama.value = "";
});
```

They do the same thing. They wrap the two statements we can executed in a function, which gets executed when the send button is clicked. Now run this and watch green text appear in Mama's box.

Programming Exercises

1. Enable similar capabilities for baby bear and papa bear.
2. Make Papa Bear's font 2em, Mama Bear's font 3em, and Baby Bear's font 1em in the black boxes. What file did you modify? Do the simplest thing possible.

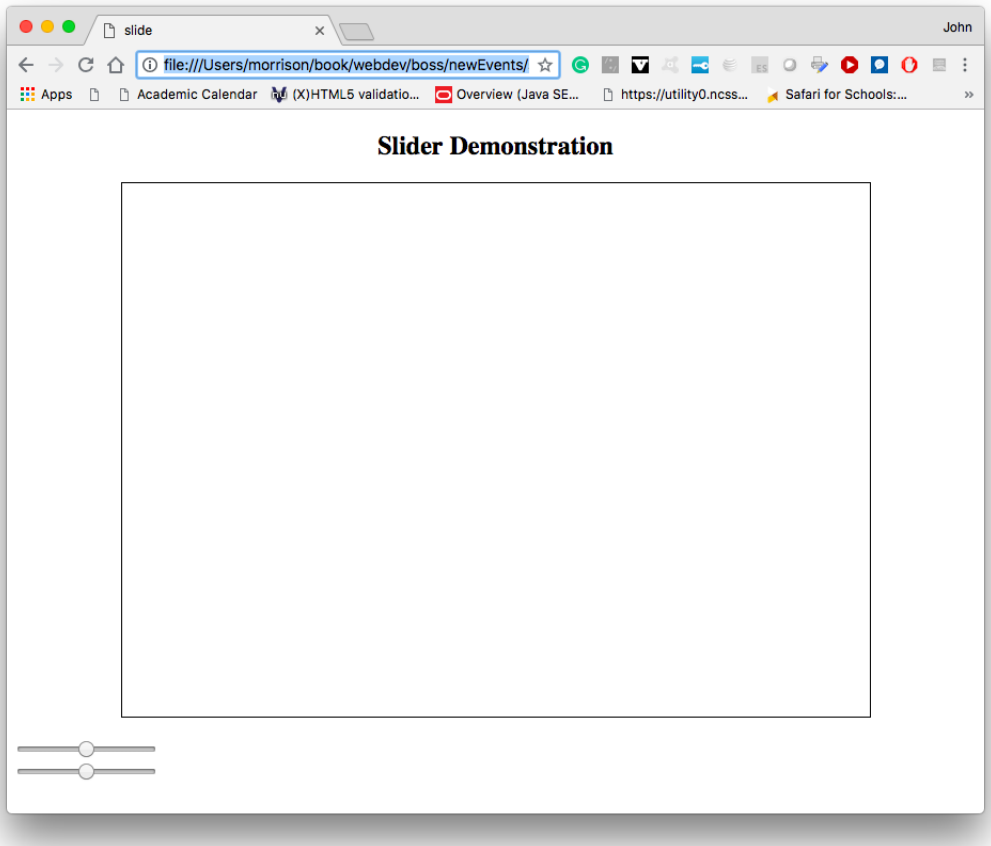
4 Slider Inputs

We will now build a little application that will move a rectangle around inside of a canvas object using two sliders, one to control vertical position, the other horizontal. We will also report the coördinates of the rectangle as it moves. In the programming exercises, you will add some features to this application. Begin by creating this HTML file, `slide.html`.

```
<!doctype html>
<html>
<head>
<title>slide</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="slide.css"/>
<script type="text/javascript" src="slide.js"></script>
</head>
<body>
    <h2>Slider Demonstration</h2>

    <p class="display">
        <canvas width="700" height="500" id="surface">
            Get a modern browser, chump.
        </canvas>
    </p>

    <div class="table">
```

```
        <div class="row">
            <div class="cell">
                <p>Horizontal:</p>
                <input type="range" id="hor"/>
            </div>
            <div class="cell">
                <p>Vertical:</p>
                <input type="range" id="ver"/>
            </div>
        </div>
    </div>
</body>
</html>
```

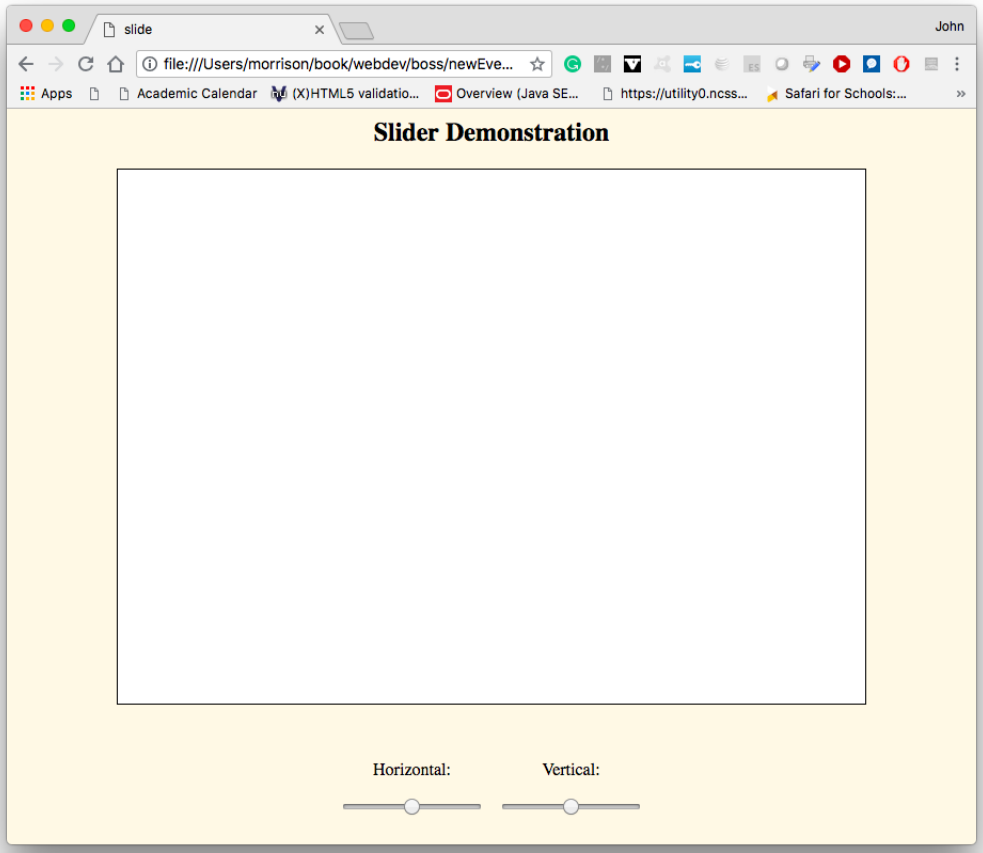
Open this in a browser tab and here is what you will see.



Let us put on a pretty face with a little CSS.

```
h1, h2, .display
{
    text-align:center;
}
canvas
{
    border:solid 1px black;
    background-color:white;
}
.table
{
    margin-left:auto;
    margin-right:auto;
    display:table;
    border-spacing:1em;
}
.row
{
    display:table-row;
}
.cell
{
    display:table-cell;
    text-align:center;
}
body
{
    padding 1em;
    background-color:#FFF8E7;
}
```

The result looks like this.



Now we get down to real business, the JavaScript. We begin by assembling all of the widgets we want interacting.

```
window.addEventListener("load", init)
function init()
{
    let c = document.getElementById("surface");
    let pen = c.getContext("2d");
    let hor = document.getElementById("hor");
    let ver = document.getElementById("ver");
}
```

What we propose to do is to move a little square around inside of the canvas. Let's, for now, make the square be 50×50 pixels and we will color it blue. We can begin to see we will need some variables. Here is a list of some considerations

1. The x-coördinate of the center of the square
2. The y-coördinate of the center of the square
3. The vertical and horizontal sizes of the canvas are needed so we can put the right range on the square
4. Look into the future: you might want to change the square's size, so let's make a variable for that.

DLet's add all of this good stuff to `init`.

```
let hSize = c.width;
let vSize = c.height;
let xCenter = hsize/2;
let yCenter = vsize/2;
let squareSize = 50;
```

What we need to do is to set the ranges of the sliders so the square can't go out of bounds. We can set the `min` and `max` properties in JavaScript appropriately to do this. Here is a big advantage to this: if you go into the HTML and change the size of the canvas, your code will not break. We will have the square start in the center of the canvas.

```
hor.min = sqaureSize/2;
hor.max = hSize - squareSize/2;
hor.value = xCenter;
ver.min = sqaureSize/2;
ver.value = yCenter;
ver.max = vSize - squareSize/2;
```

Nothing demonstrable yet is visible. We now need to activate the sliders. They have two event types: `change` which fires after a change is made and `input`, which fires as they are changing. We will use `input` so we can see it happen live.

When the sliders move, we need to repaint the square in its new location. Let us write a function, inside of `init`, called `refresh` to do just that.

```
function refresh()
{
    pen.fillStyle = "blue";
    xCenter = Number(hor.value);
    yCenter = Number(ver.value);
    pen.fillRect(xCenter - squareSize/2, yCenter - squareSize/2, squareSize, squareSize)
}
```

Let us now see if it puts the square in the right place. We will call it to bring the square to the center.

```
window.addEventListener("load", init)
function init()
{
    let c = document.getElementById("surface");
    let pen = c.getContext("2d");
    let hor = document.getElementById("hor");
    let ver = document.getElementById("ver");
    let hSize = c.width;
    let vSize = c.height;
    let xCenter = hSize/2;
    let yCenter = vSize/2;
    console.log(xCenter);
    console.log(yCenter);
    let squareSize=50;
    hor.min = squareSize/2;
    hor.max = hSize - squareSize/2;
    hor.value = xCenter;
    ver.min = squareSize/2;
    ver.max = vSize - squareSize/2;
    ver.value = yCenter;

    pen.fillStyle = "white";
    pen.fillRect(0,0,hSize, vSize);
    pen.fillStyle = "blue";
    xCenter = Number(hor.value);
    yCenter = Number(ver.value);
    pen.fillRect(xCenter - squareSize/2, yCenter - squareSize/2, squareSize, squareSize);
    refresh();
    function refresh()
    {
        pen.fillStyle = "blue";
        xCenter = Number(hor.value);
        yCenter = Number(ver.value);
        pen.fillRect(xCenter - squareSize/2, yCenter - squareSize/2, squareSize, squareSize)
    }
}
```

Whee! Now add this to `init`.

```
    hor.addEventListener("input", refresh);
    ver.addEventListener("input", refresh);
```

Refresh your browser. You will notice the square gets "smeared." This is

because we have to repaint the background as well. Change `refresh` to look like this.

```
function refresh()
{
    pen.fillStyle = "white";
    pen.fillRect(0,0,hSize, vSize);
    pen.fillStyle = "blue";
    xCenter = Number(hor.value);
    yCenter = Number(ver.value);
    pen.fillRect(xCenter - squareSize/2, yCenter - squareSize/2, squareSize, squareSize);
}
```

Et voila! The square moves as we want it to.

Programming Exercises

1. Change the size of the canvas and see that the application still works.
2. Add a slider to control the size of the square. The square should only expand as far as it can while still staying inside the bounds of the canvas.
3. Add a row of colored buttons and have them change the color of the square to their color when they get pushed.
4. After the word "Vertical" on the page, place a span and show the vertical coördinate. Deal similarly with the horizontal slider.

5 Color Pickers

In this section, we will learn how to allow users of our sites to select a color with a color picker. Begin by creating this HTML file. You will notice that we have two `input` elements of type `color`.

```
<!doctype html>
<html>
<head>
<title>color</title>
<meta charset="utf-8"/>
<link rel="stylesheet" href="color.css"/>
<script type="text/javascript" src="color.js"></script>
</head>
<body>
    <h2>Color Picker Example</h2>
```

```
    <p class="display">Click on the square to change the background
        color:  <input type="color" id="bgpicker" value="#FFFFFF"/></p>

    <p class="display">Click on the square to change the text
        color:  <input type="color" id="fgpicker"/ value="#000000"></p>
</body>
</html>
```

Now let's add a little very minimal CSS.

```
h1, h2, .display
{
    text-align:center;
}
```

We will now activate the color pickers. If you click on them now, a color picker will come up and it won't do anything. We add this JavaScript. Note the use of anonymous "arrow functions" as second arguments to the `addEventListener` method.

```
window.addEventListener("load", init);
function init()
{
    let bgpicker = document.getElementById("bgpicker");
    bgpicker.addEventListener("input", e =>
    {
        document.body.style.backgroundColor = bgpicker.value;
    });
    let fgpicker = document.getElementById("fgpicker");
    fgpicker.addEventListener("input", e =>
    {
        document.body.style.color = fgpicker.value;
    });
}
```

As you change the color in a color picker, the color on the page changes.

Programming Exercise

1. Change the first argument in `addEventListener` to `"change"`. What happens?
2. Add a color picker to `slider.html` to change the color of the square and make it work.

6 Dropdown Menus: CSS and JS on the Same Team