Interactive Web Skills Project

1) First, make sure you have updated our resource repository by '**cd**' ing into your *~/Documents/CodeLab2019* folder and then entering in the command '**git pull**'. This will update the repository and pull down some sound files we will need for this project.

2) We want to first make a new folder to put our project into. 'cd' into your Documents folder (*~/Documents*) and then create a new folder named "**drums**" using the '**mkdir**' command. '**cd**' into your new 'drums' folder, and then use the '**touch**' command to create a new file named '**index.html**', a new file named '**index.js**' and a new file named '**style.css**'. Then, use the '**mkdir**' command to make a new folder named '**sounds**'. You should now have the three new files and your new 'sounds' folder within the 'drums' folder - use the '**ls**' command to verify.

3) Now, we are going to copy our drum machine sounds from the resource repo folder to our new 'sounds' folder. First change into the repo folder by entering '**cd** *~/Documents/CodeLabSpring2019/Interactive_Web_Project/sounds*'. '**ls**' to make sure that you are in the right directory; several .mp3 files should show up. Now, enter the following command:
cp ***.mp3** *~/Documents/drums/sounds*

This command tells the shell to copy all mp3 files in our current directory (the * is a wildcard, so any file that ends with a .mp3 extension is selected here) to our new 'sounds' directory we just created. Now, '**cd**' back to *~/Documents/drums/sounds* and then '**ls**' to verify the files were copied. It is hard to make a drum machine without sounds!

4) We should currently be in the '*~/Documents/drums/sounds*' folder. Let's go up two directories by entering '**cd ../../**'. Now we should just be at our base '**~/Documents**' directory. Enter in the command '**atom drums**' to open our new project in Atom.

5) Once your project has opened in Atom, double-click the 'index.html' file on the left-hand side. You should now be able to enter in code for this file on the right-hand side. Enter in the following code to build the skeleton of our html page:

> **Note:** You may be able to auto-enter much of this if you have the html autocomplete plugin in Atom installed. Begin typing html (without the opening bracket) and then hit tab. If the plugin is active, Atom will pre-populate your html file with most of the tags below, and you can focus on putting in the title text sas well asl all thje and everything it contains.

```
<!DOCTYPE html>
<head>
     <title>Tap Music </title>
</head>

<body>
     <div class="app">
          <header>
               <h1>Tap Music</h1>
               <p>Make music with only one tap</p>
          </header>
     </div>

</body>
</html>
```

We add the required tags to make this an html file by adding the `<html>`, `<head>` (and `<title>`), and `<body>` tags. Then, within the body, we have placed a `<div>` tag with a class of "**app**", which we will place all the rest of our markup into. Our first tag within this is a `<header>` tag, within which we place a bold line of text (`<h1>`) and a description (the `<p>` text). Make sure you save your file.

6) Before we go any farther, let's make sure everything looks good so far. Go back to the terminal; you should still be in your *~/Documents* folder. '**cd**' into the "drums" folder, then '**ls**' to make sure the 'index.html' file is listed. Enter the command '**http-server**' and then enter. The terminal should tell you that it is serving at the *http://127.0.0.1:8080* address.

127.0.0.1 is the local host (your machine you are working on), and 8080 refers to the port that the web server is available at. Ports 80 and 8080 are traditionally used for http, so you may see these numbers a lot when doing web work. You can copy this address from the terminal using `Shift + CTRL-C`, or just remember it. Now, open up a web browser on your system and enter this same address into the address bar of the browser. If all goes well, you should see your web page appear.

7) Now that we've got our site running, let's finish building out our html structure. Below the `</header>` tag, enter in two more divs.

```
<div class="visual">

</div>
<div class="pads">

</div>
```

We are going to add six "drum pads" (ok, colored boxes) to our page, so within our "pads" class, we are going to add six more divs. Add the following code:

```
<div class="pad1"></div>
<div class="pad2"></div>
<div class="pad3"></div>
<div class="pad4"></div>
<div class="pad5"></div>
<div class="pad6"></div>
```

Now, for each drum pad, we need a sound. So we are going to add an '`<audio>`' tag that links to a sound in our 'sounds' folder for each one of our drum pads. Edit your code so that it appears as below:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Tap Music</title>
  </head>
  <body>
    <div class="app">
      <header>
        <h1>Tap Music</h1>
        <p>Make music with a tap</p>
```

```
      </header>
      <div class="visual">

      </div>
      <div class="pads">
        <div class="pad1">
          <audio class="sound" src="./sounds/bubbles.mp3"></audio>
        </div>
        <div class="pad2">
          <audio class="sound" src="./sounds/clay.mp3"></audio>
        </div>
        <div class="pad3">
          <audio class="sound" src="./sounds/confetti.mp3"></audio>
        </div>
        <div class="pad4">
          <audio class="sound" src="./sounds/glimmer.mp3"></audio>
        </div>
        <div class="pad5">
          <audio class="sound" src="./sounds/moon.mp3"></audio>
        </div>
        <div class="pad6">
          <audio class="sound" src="./sounds/ufo.mp3"></audio>
        </div>

      </div> <!-- this is the close tag for the "pads" div -->

    </div> <!-- this is the close tag for the "app" div -->
  </body>
</html>
```

8) Now let's link up our stylesheet and add some basic styles so we can see our drum pads. First,within the top `<head>` tag, we need to add a link so our html page will know how to load our styles. add

```
<link rel="stylesheet" href="./style.css">
```

r

ight above the `<title>Tap Music</title>` tag.

We are also going to use a "web font". A web font is a font that is pulled from an external site, rather then from the computer or device that the web browser is running on. Add the following right below the last line we added:

```
<link href="https://fonts.googleapis.com/css?family=Montserrat" rel="-
stylesheet">
```

right above the `<title>Tap Music</title>` tag. Save your file, then double-click on the **.style.css** file on the lefthand side in Atom, so that we can enter in some css markup.

We are going to add the following into the 'style.css' file:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Montserrat', sans-serif;
}

.app {
  height: 100vh;
  display:flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
}

header h1 {
  margin: 50px 0px 30px 0px;
  text-align: center;
  font-size: 40px;
}
```

The `*{ . . . }` gets applied to everything, and we are using this here to set everything to a known baseline. We then set our base font to "Montserrat". Since we told our html file where to find this font with the link we added, we are able to now use this font in our style sheet.

Notice you need to start the 'app' css with a period, since it is a custom class we created ourselves. The 100vh sets the height of this div (which everything else on the screen resides in) to 100% of our screen height. We are putting each drum pad into a column, so the other rules we are adding here affect how these columns will present their content. Finally, we add a style for the top header tag.

Save this file, then reload the web page in your web browser to see your changes. The title and description should now be centered and using the new font. (If no changes appear, first make sure you saved both the `index.html` and `style.css` file, and if that doesn't help, you may need to clear your web browser's cache.

9) We need to add some more css styles for the drum pads themselves. At the bottom of the style.css file, add the following:

```
.pads {
  background: gray;
  width: 100%;
  display: flex;
}

.pads > div {
  height: 100px;
  width: 100px;
  flex: 1;
```

```
}

.pad1 {
  background: skyblue;
}
.pad2 {
  background: seagreen;
}
```

As you can see, these styles affect the pads themselves. The '.pads' style sets the display to 'flex' here so that the columns run horizontally across, rather than vertically down the screen. The `.pads > div {}` syntax tells the browser to apply the style to every div that the '.pads' div contains (that is what that '>' allows for). This sets all the contained divs (pad1, pad2 etc) to use the same height and width. Then we do set an individual style for each pad so we can assign different colors , but at least we do not need to repeat awidth, hieght, and flex for each one.

10) Now we need to start adding some javascript code to start making things work. First, we need to add a link in our html file so it loads our javascript code, so switch over to index.html. Unlike the links for the font and stylesheets that we placed in our `<header></header>` section, we need to place our link for our script towards the end of the file. Right *above* the closing body tag (`</body>`), add the line:

```
<script type="text/javascript" src="./index.js">
```

The reason that we place the link to the script file here is because we are going to be referring to parts of the html file by class, and we need to have the web browser load the html file before it loads the javascript so that those classes are already available for the code to use. Save the `index.html` file.

Double-click the '`index.js`' file in Atom so that you are able to enter code into it. Add the following code:

```
window.addEventListener('load', () => {
  const sounds = document.querySelectorAll(".sound");
  const pads = document.querySelectorAll(".pads div");

  pads.forEach((pad, index) => {
    pad.addEventListener("click", function() {
      sounds[index].currentTime = 0;
      sounds[index].play();
    });
  });
});
```

And then save your javascript file. There are not very many lines of code here, but there is actually quite a bit going on. The first thing to know is that 'window' is an object that refers to the browser window, and is globally available to us. The first line of code tells the browser window to run the function that follows once it has loaded (again, we need to make sure those parts of the html already exist in the browser before using them in our code). The '`=>`' is called a 'fat arrow' and basically sets the function that follows as a callback function. Essentially this is saying: 'when the window has finished loading, run the following code'.

Once the function has been started on the first line, we set two new variables, the 'sounds' variable that gets every object in the html that has a class of sound (all our audio tags), and the 'pads' variable that gets every object in the html that is a div object within the .pads div. The 'const' is used in javascript to declare a variable that will not change once set (you

could also use 'let' or 'var' if the variable does need to change at some other point in your code).

Once we have gotten a hold of all our sounds, and all of the drum pads we created, we run through each pad and add an event listener to the pad that can play the corresponding sound. We can iterate (go through each object from first to last) that pads array by calling the forEach() funtion that javascript collection objects provide. In that forEach function, we pass a 'pad' variable, which gets used to capture each 'object' found through the iteration process, as well as an 'index' parameter that is provided to us by forEach. We use index to find out what particular iteration we are on. Since we are iterating through each drum pad we set up, the iterator will run six times (the amount of drum pads we created). Since we also have six sounds in our 'sounds' array, we can use the 'index' value to access the first sound for the first drum pad, the second sound for the second drum pad, and so on.

With those parameters set, we then use another callback function to add an event listener to every 'pad' object, that when clicked, will call a function that first sets the sound's current Time to zero, and then plays the sound found at the same index in the sounds array. The reason we set the currentTime to zero is that if we didn't, each sound would have to play the full sample before another sound would play. This 'resets' the audio player so that each time a pad gets clicked, any existing sound that might be playing stops.

Like I say, there is a lot going on here. Javascript isn't necessarily the best first language to learn, simply because the syntax is pretty strange when you are first learning it. Part of the language is supplied by the browser itself, and part of it allows for javascript to work asynchronously, which is extremely important for web applications, but it makes for a lot to know. Luckily, once you understand the logic as explained above, you can adapt it to your own needs, and there are many many references online to particular javascript patterns and usages.

Once all your files are saved, reload the web browser. You should now be able to play your sounds by clicking on your drum pads.

Each completed file for section 1 is listed below.

index.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="./style.css">
    <link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">
    <title>Tap Music</title>
  </head>
  <body>
    <div class="app">
      <header>
        <h1>Tap Music</h1>
        <p>Make music with a tap</p>
      </header>
      <div class="visual">

      </div>
      <div class="pads">
        <div class="pad1">
          <audio class="sound" src="./sounds/bubbles.mp3"></audio>
        </div>
        <div class="pad2">
          <audio class="sound" src="./sounds/clay.mp3"></audio>
        </div>
        <div class="pad3">
```

```
          <audio class="sound" src="./sounds/confetti.mp3"></audio>
        </div>
        <div class="pad4">
          <audio class="sound" src="./sounds/glimmer.mp3"></audio>
        </div>
        <div class="pad5">
          <audio class="sound" src="./sounds/moon.mp3"></audio>
        </div>
        <div class="pad6">
          <audio class="sound" src="./sounds/ufo.mp3"></audio>
        </div>

      </div> <!-- this is the end of the pads div -->

    </div> <!-- this is the end of the app div -->
    <script type="text/javascript" src="./index.js">

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

style.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Montserrat', sans-serif;
}

.app {
  height: 100vh;
  display:flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
}

header h1 {
  margin: 50px 0px 30px 0px;
  text-align: center;
  font-size: 40px;
}

.pads {
  background: gray;
  width: 100%;
  display: flex;
}

.pads > div {
  height: 100px;
  width: 100px;
  flex: 1;
}
.pad1 {
  background: skyblue;
}
```

```
.pad2 {
  background: seagreen;
}
.pad3 {
  background: peru;
}
.pad4 {
  background: thistle;
}
.pad5 {
  background: teal;
}
.pad6 {
  background: orange;
}
```

index,js

```
window.addEventListener('load', () => {
  const sounds = document.querySelectorAll(".sound");
  const pads = document.querySelectorAll(".pads div");


  pads.forEach((pad, index) => {
    pad.addEventListener("click", function() {
      sounds[index].currentTime = 0;
      sounds[index].play();
    });
  });
});
```

SECTION 2 Adding a Visual Effect

11) When we initally created our html page, we added a "visual" div directly above our "pads" div, but so far haven't done anything with it. We first need to get a reference to this div in our javascript code, so add the following line right below "*`const pads = documentQuerySelectorAll(".pads div");`* in your **index.js** file :

```
const visual = document.querySelector(".visual");
```

Note that this time we are using querySelector instead of querySelectorAll. We only have one div named "visual' so we just want to get the one object with that class name.

We also want to set up an array of the same colors we used in our css file file for the drum pads, but this time in our javascript. Right below the last line you just entered, add the following line:

```
const colors = ["skyblue", "seagreen", "peru", "thistle", "teal", "orange"];
```

Now, we are going to add another function to our code to make our visuals. Right *before* the final '`});`' in the javascript file, insert your cursor and then hit the enter key a few times to give you a bit of space. Then add the following code:

```
const createBubbles = (index) => {
    const bubble = document.createElement("div");
    visual.appendChild(bubble);
    bubble.style.backgroundColor = colors[index];
    bubble.style.animation = "jump 1s ease";
  };
```

Ok, this is another weird looking thing. But just like a variable in javascript can be assigned to a number value, or a string of text, it can also be assigned to a function. The '`(index) =>` ' allows us to pass an index value to the callback function that follows.

Within the function itself, we create a new vraiable named 'bubble' that does something pretty neat: Simply declaring this 'bubble' tells the html document to create a new `div` element, just like the 'div' elements we created ourselves in the html file. This points to one of the most powerful aspects of web development. Just because a web browser is initally given an html file to render, doesn't mean that we can't ***change*** the html after the fact through the use of javascript and the web browser's document objct model.

The next line tells the 'visual' div that we retrieved using the '`const visual = document.querySelector(".visual");`' to append (add) our new bubble div to it. The next two lines adjust our new bubble div's style object, adjusting both the background color and adding the beginnings of an animated movement.

But how does this createBubbles function actually get called? And we are passing an index number to it, but where does that come from?

Well, the way we resolve this is that we call this function from our `pads.forEach()` function above. Place the following line:

```
createBubbles(index);
```

within the `pads.forEach` function, directly below the `sounds[index].play()` line. What this now does is that whan a drum pad is clicked, along with re-setting the currentTime of the sound to zero and then playing the sound at the associated index, we also then call our new 'createBubbles' function, passing along that same index value we just passed to `sounds[index].play()`. That index value gets passed to createBubbles function, and can then be referenced from there.

This is very common when using functions, regardless of the language you are using. You may create or retrieve a value you need in one function, and pass that value to another function for further action.

12) We need to do a little bit more with our animation, but not in javascript. We have to edit our css file to add some parameters, so that the "jump" animation knows where to start and stop. In the style.css file, add the following style at the bottom of the file:

```
.visual > div {
  position: absolute;
  height: 50px;
  width: 50px;
  border-radius: 50%;
  left: 20%;
  bottom: 0%;
}
```

Much like the style we created for `.pads > div`, this style affects any divs that are contained within the 'visual' div we created. So any bubble divs we create get this style.

The 'position' attribute affects how the placement of the div is placed within its' containing div. The height and width set our bubble to 50 pixels both down and across, and the border-radius at 50% renders our div as a circle (a radius affects the corner of a square, and a 50% radius effectively turns a square into a circle). The bottom and the left values place the div at the bottom of the screen and 20% from the left of the screen. This is the base position the div will start at prior to the animation starting.

Now that we have added this style, we need to flesh out how the animation will render. So below the css style you just added, enter the following:

```
@keyframes jump{
  0% {
    bottom: 0%;
    left: 20%;
  }
  50% {
    bottom:  50%;
    left: 50%;
  }
  100% {
    bottom: 0%;
    left: 80%;
  }
}
```

Because we set our bubble animation to 'jump in our javascript code (`bubble.style.animation = "jump 1s ease";`), the animation can then be extended within the css file.

When using a css animation, we can adjust how the animation 'works' by setting parameters using the css `@keyframes` tag. We have to 'assign' the keyframes to our animation, which is named 'jump' as per how we set it up in javascript. the '1s' (that is a number 1, not the letter l!) that is part of our animation definition tells the browser to run the animation for one second. The `@keyframes` in our css file states that at the very beginning of the animation we start at the same base location we set in the `.visual > div` command, then at half a second into the animation, the bubble will be half-way up and across the screen (the 50%) and at the end of the animation (the 1 second mark or 100% completion) the animation will end at the bottom of the screen and 80% of the distance from the left hand side.

Make sure all your files (`index.html`, `index.js` and `style.css`) have been saved. Then reload your page in the web browser. If all is well, when you click a drum pad, you should now have a circle that moves from the bottom to the middle of the screen, and then back down, along with the drum pad sound. If it doesn't work, carefully check your code. In particular, make sure you aren't missing any ';' at the end of your css values. CSS is touchy about that.

13) We are pretty close, but there are a few things we need to clean up. You may have noticed that our flying bubble appears in front of our drum pad. Luckily, this is pretty easy to change if we want. Add the following line to the `.visual > div` style in `style.css`, right below 'bottom: 0%;"

```
z-index: -1;
```

Then save and reload. The z-index controls how elements get placed on top of each other. By setting the z-index to -1 we send the div to the back, so that it is behind the drum pads instead of the front.

The other thing we need to do, as good developers, is to make sure we are managing our resources correctly. Currently, every time we click on a drum pad, a new 'bubble' div gets created, but it doesn't disappear once the animation has ended.

To illustrate this, load (or reload) your page in Google Chrome. In Chrome, make sure that you are showing the Developer Tools (In the 3-dot navigation item on the right side, then in the 'More Tools' submenu). Once the developer tools are open, make sure the 'Elements' tab is active. You should see your html below. Find the div class visual (you will have to 'open' the 'app' div with the little triangle next to it), then 'open' it as well. Each time you click on a drum pad, a new line of html code gets written. This is a perfect example of how the javascript code 'manipulates' the html from what we have in our written `index.html` file. But as it stands, it isn't a good programming practice. We are asking the browser to maintain that bubble, even though we are done with it. Enough bubbles, and pretty soon your web page is taking a lot of memory, and potentially slowing down or crashing the web browser.

Back in your javascript file, give yourself a line to write on directly below the '`bubble.style.animation = "jump 1s ease`' line. Then enter:

```
bubble.addEventListener("animationend", function() {
      visual.removeChild(bubble);
    });
```

While this code still looks pretty ugly, it should at least be a bit familiar by now. When we add an event listener to an object, we are basically telling it to do something when that event happens. When we added a '**click**' event listener for the drum pad, that enabled the pad to run the code that followed when clicked by a mouse. 'click' is an event that the browser provides us. So is '**animationend**', which is the event that we are telling our bubble to listen to in this code. When the bubble receives a notifcation from the browser that its' animation has stopped, the bubble then tells the 'visual' object (the visual div) that it was appended to earlier to remove it.

Save your change and reload the page. Reopen all the divs within Chrome's developer window so you can see the 'visual' div. Now, click the drum pads. You should see your code adding the bubbles, and then removing them when they have completed. Now your web page is being responsible and maintaining high performance.

Congratulations! You have made it to the end of the tutorial. Experiment with changing the colors (right now I am using named colors for simplicity but I recommend finding an html color picker online and using hex colors).

Find some different (free) .mp3 sounds and swap them out.

Add a couple more drum pads by changing your layout if you are feeling particulary ambitious!

This tutorial was adapted from a very good video tutorial on youtube by Dev Ed:
https://www.youtube.com/watch?v=2VJlzeEVL8A

Final File Listings below

index.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
```

```
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="./style.css">
    <link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">
    <title>Tap Music</title>
  </head>
  <body>
    <div class="app">
      <header>
        <h1>Tap Music</h1>
        <p>Make music with a tap</p>
      </header>
      <div class="visual">
        <!-- Our javacript code places the visual effects here -->
      </div>
      <div class="pads">
        <div class="pad1">
          <audio class="sound" src="./sounds/bubbles.mp3"></audio>
        </div>
        <div class="pad2">
          <audio class="sound" src="./sounds/clay.mp3"></audio>
        </div>
        <div class="pad3">
          <audio class="sound" src="./sounds/confetti.mp3"></audio>
        </div>
        <div class="pad4">
          <audio class="sound" src="./sounds/glimmer.mp3"></audio>
        </div>
        <div class="pad5">
          <audio class="sound" src="./sounds/moon.mp3"></audio>
        </div>
        <div class="pad6">
          <audio class="sound" src="./sounds/ufo.mp3"></audio>
        </div>

      </div> <!-- this is the end of the pads div -->

    </div> <!-- this is the end of the app div -->
    <script type="text/javascript" src="./index.js">

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

style.css

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Montserrat', sans-serif;
}
```

```
.app {
  height: 100vh;
  display:flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
}

header h1 {
  margin: 50px 0px 30px 0px;
  text-align: center;
  font-size: 40px;
}

.pads {
  background: gray;
  width: 100%;
  display: flex;
}

.pads > div {
  height: 100px;
  width: 100px;
  flex: 1;
}

.pad1 {
  background: skyblue;
}
.pad2 {
  background: seagreen;
}
.pad3 {
  background: peru;
}
.pad4 {
  background: thistle;
}
.pad5 {
  background: teal;
}
.pad6 {
  background: orange;
}

@keyframes jump{
  0% {
    bottom: 0%;
    left: 20%;
  }
  50% {
    bottom:  50%;
    left: 50%;
  }
  100% {
    bottom: 0%;
    left: 80%;
  }
}
```

```
.visual > div {
  position: absolute;
  height: 50px;
  width: 50px;
  border-radius: 50%;
  left: 20%;
  bottom: 0%;
  z-index: -1;
}
```

index.js

```
window.addEventListener('load', () => {
  const sounds = document.querySelectorAll(".sound");
  const pads = document.querySelectorAll(".pads div");
  const visual = document.querySelector(".visual");
  const colors = ["skyblue", "seagreen", "peru", "thistle", "teal", "orange"];

  pads.forEach((pad, index) => {
    pad.addEventListener("click", function() {
      sounds[index].currentTime = 0;
      sounds[index].play();

      createBubbles(index);
    });
  });

// Create function to make bubble visuals
  const createBubbles = (index) => {
    console.log(`creating bubblw with index # ${index}`);
    const bubble = document.createElement("div");
    visual.appendChild(bubble);
    console.log(`bubble added to visual div: ${bubble}`);
    // bubble.style.backgroundColor = "#ff0000";
    bubble.style.backgroundColor = colors[index];
    bubble.style.animation = "jump 1s ease";
    bubble.addEventListener("animationend", function() {
      visual.removeChild(bubble);
    })

  };

});
```