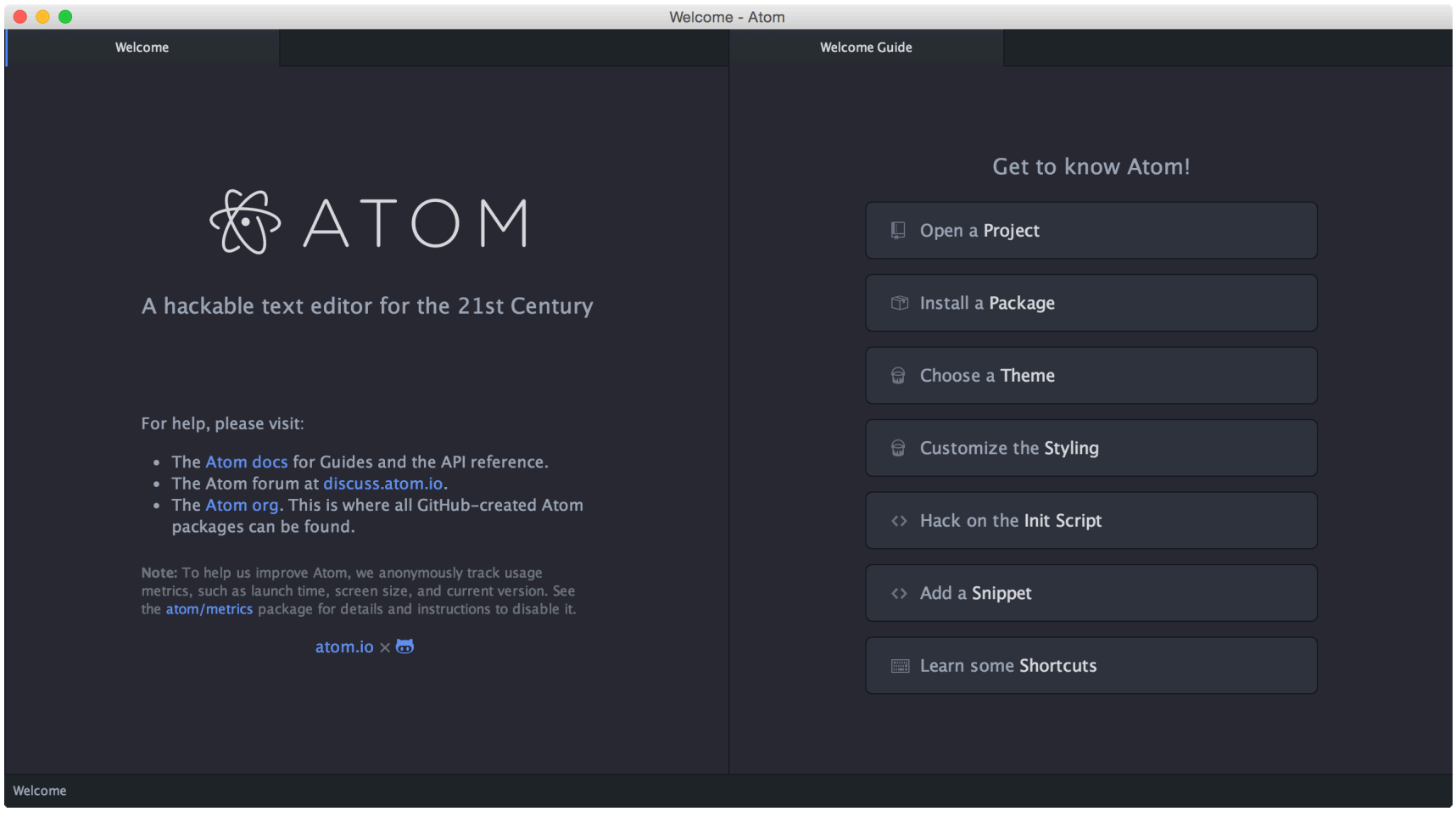
1 Atom



Overview and Purpose

This checkpoint introduces you to Atom, the editor you'll use as part of your web development process.

Objectives

After this checkpoint, you should be able to:

- Use Atom command line commands.
- Use the package manager in Atom.

Why Atom?

A robust text editor makes writing code more efficient and pleasant. Most options have merit, but Bloc chooses **Atom**. Atom is modern, approachable, customizable, open source, and free. It's built and maintained by GitHub.

> If you've set up your environment using a cloud IDE, we suggest sticking with the default text editor. Instead of installing Atom, go to the **assignment** and complete it.

If you have a **local** environment setup, the first step is to **download Atom** and install it.

Take Atom for a Test Drive

Open Atom and create a new file named `hello_world.js`. Type the following lines of code in the editor:

hello_world.js

```
function hello_world() {
  console.log("hello world");
}
```

Atom automatically highlights and colors important syntax elements of the code. Syntax highlighting improves the readability and context of code. It also helps us find mistakes in code.

Atom Command Line Shortcut

You can use Atom to open files from the command line, which is extremely useful. To open a file in Atom, use the `atom` command that was installed with Atom:

Terminal

```
$ atom hello_world.js
```

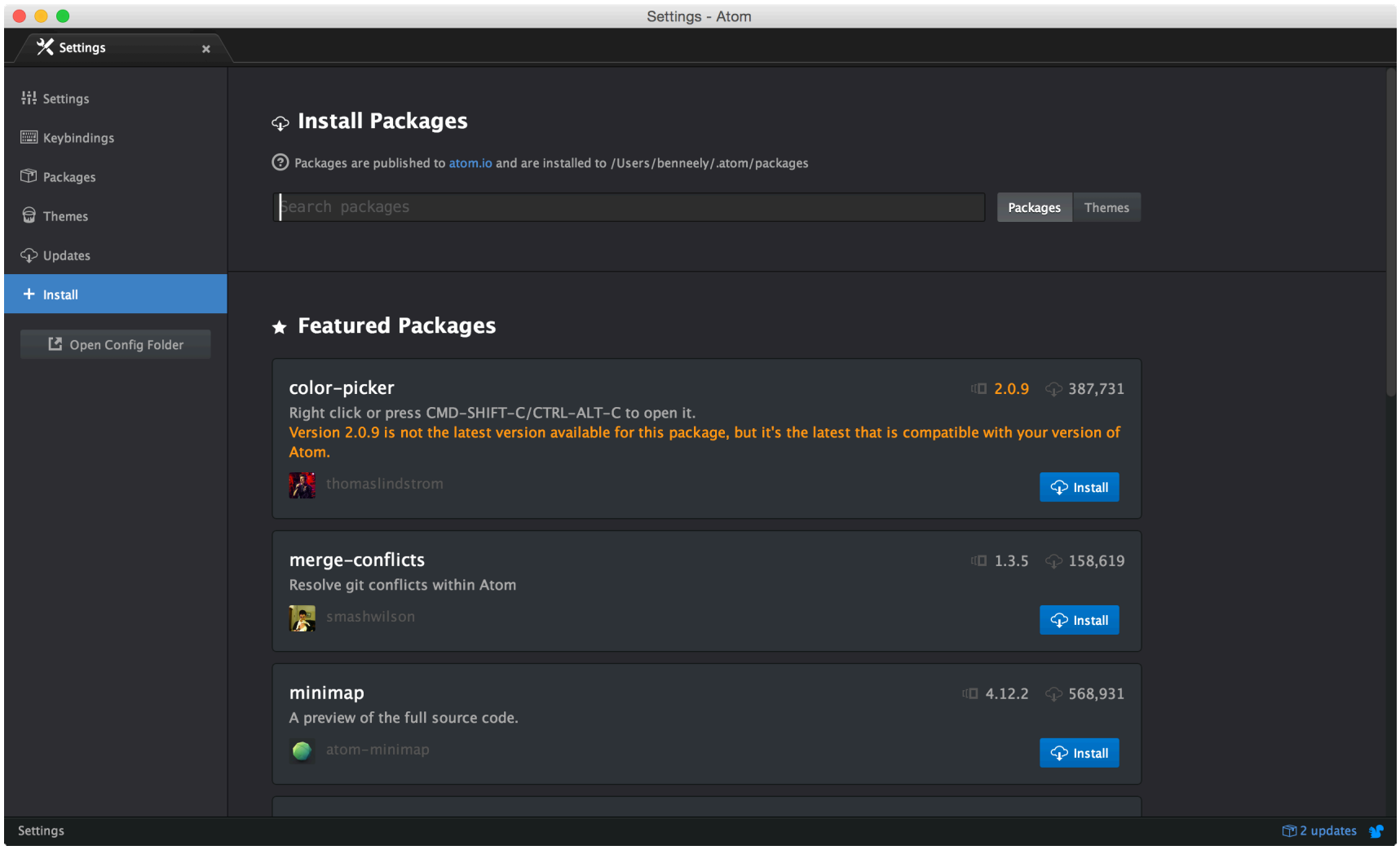
`atom` can also be used to open an entire directory:

Terminal

```
$ atom .
```

Package Manager

Atom ships with a built-in package manager which makes it easy to install plugins that make Atom pleasant and productive to work with. Hit `CMD+SHFT+P` if you're on a Mac or `CTRL+SHFT+P` if you're using a Windows machine. Start typing "Install Package", select "Setting View: Install Packages and Themes" and hit `enter`. This opens the "Install" section of "Setting":



In the search box, enter "atom-live-server", click enter and scroll down to the "atom-live-server" package and install it.

`atom-live-server` launch a simple development HTTP server with live reload capability. This will be a valuable tool to use as you're building BlocJams and other FE projects.

There are many other Atom packages and themes, experiment with changing the default Atom theme.

Concept	Description
Atom	Atom is a sophisticated text editor for code, markup, and prose.
`atom` **command**	The `atom` command opens files and directories in Atom, from the command line.
Atom Package Manager	The Atom Package Manager makes it simple to find, install, and keep packages up-to-date.
Atom Live Server	Atom Live Server launches a simple development HTTP server with live reload capability..

How would you rate this checkpoint and assignment? ☺ ☹

1. Atom

Assignment | Discussion | Submission

2 HTML & CSS: Basic Structure

Overview and Purpose

This checkpoint introduces basic HTML tags, the building blocks of any HTML document.

Objectives

After this checkpoint, students should be able to:

- Recognize an HTML tag.
- Understand and explain the purpose of essential HTML tags: `<html>`, `<head>`, `<body>`.
- Identify HTML elements.
- Discuss various HTML tags – such as `<title>`, `<div>`, `<h1>`, etc. – and their uses.
- Explain why indenting is important in HTML files.

Welcome to Bloc Jams, a digital music player like **Spotify** that we'll use to learn frontend web development. We'll start by building the backbone of the application layout with HTML, add styling and responsiveness using CSS, and implement interactivity with JavaScript.

Setup Terminal/Git

As you build applications, you will need to share the code with your mentor. Git is a distributed revision control system that allows us to share code, rewind our code to previous versions, and keep track of the changes we make along the way. But Git doesn't have the same graphical interface that most programs do. It is run from the command line, or what is often called the Terminal.

If you are using a Mac or Linux computer, you already have Terminal and Git installed. You can move on to creating the Bloc Jams directory. If you are on a Windows machine, please download and install **GitBash**.

GitBash will give you a small executable file that will bring up a working terminal with Git already installed on it.

Create the Bloc Jams Directory and Initialize Git

Create a directory to hold the application.

TerminalOrGitBash

```
$ pwd #=> you should be in `~/bloc`, if not, run cd ~/bloc
$ mkdir bloc-jams
```

`cd` into `bloc-jams` and initialize it as a Git repository.

TerminalOrGitBash

```
$ cd bloc-jams
$ git init
```

Create an `index.html` file, stage the file for commit using `git add`, and make your first commit.

TerminalOrGitBash

```
$ touch index.html
$ git add index.html # stages it for commit
$ git commit -m "first commit" # commits it with the "first commit" message
```

Create a corresponding repository on GitHub called `bloc-jams`, and add the remote location of the repo in the command line:

TerminalOrGitBash

```
$ git remote add origin https://github.com/<your-username>/bloc-jams.git
```

Push your local repository to GitHub:

TerminalOrGitBash

```
$ git push origin master
```

Work on a Feature Branch

Create a new Git feature branch for this checkpoint named `checkpoint-2-basic-html`.
Review **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Add the Basic HTML Structure

HTML, or HyperText Markup Language, supplies the basic structure of a web page using **tags**. Tags are enclosed by a less-than sign (`<`) and a greater-than sign (`>`) on either side, like `<tag-name>`. They represent HTML **elements**, or individual components that make up an HTML document. With few exceptions, HTML tags wrap content (like text or other HTML elements) and must be opened (`<tag>`) and closed (`</tag>`) on either side of that content.

Add the following to `index.html`:

~/bloc/bloc-jams/index.html

```
+ <!DOCTYPE html>
+ <html>
+     <head></head>
+     <body></body>
+ </html>
```

Let's explore each of these tags:

- `<!DOCTYPE html>` tells the browser that this file should be interpreted as HTML.
- The `<html>` tags mark the beginning and end of the content on a page.
- The `<head>` tag contains a set of information that won't be directly rendered by the page. From the Mozilla documentation:

  > The HTML Head Element (`<head>`) provides general information (metadata) about the document, including its title and links to or definitions of scripts and style sheets (**source**)

- The `<body>` element contains the content displayable in the browser.

A `<title>` tag is *required* to ensure that the HTML renders in all browsers. Some browsers may not render anything in the body without the closing `</title>` tag. Let's add one to the head with our application's name:

~/bloc/bloc-jams/index.html

```
<!DOCTYPE html>
<html>
    <head>
+       <title>Bloc Jams</title>
    </head>
    <body></body>
</html>
```

The title is displayed at the top of the tab or window, depending on the browser. Here's where Chrome displays the title:

Add Content to the Body

Start your editor's live preview (in Atom, you can use the keyboard shortcut `CTRL+ALT+L` or select "Packages -> atom-live-server -> Start Server" in the main menu). By refreshing the browser tab after changes, you'll be able to see how CSS affects the appearance of the page.

Add three main sections to `index.html`, which will be wrapped in `<div>` tags (short for "division"):

~/bloc/bloc-jams/index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Bloc Jams</title>
    </head>
    <body>
+       <div> <!-- navigation bar -->
+       </div>
+       <div> <!-- hero content -->
+       </div>
+       <div> <!-- selling points -->
+       </div>
    </body>
</html>
```

We've annotated the purpose of each of these divisions using HTML comments, which are opened with the `<!--` characters and closed with `-->`. The browser will not render anything written as a comment.

Add a Hero Container

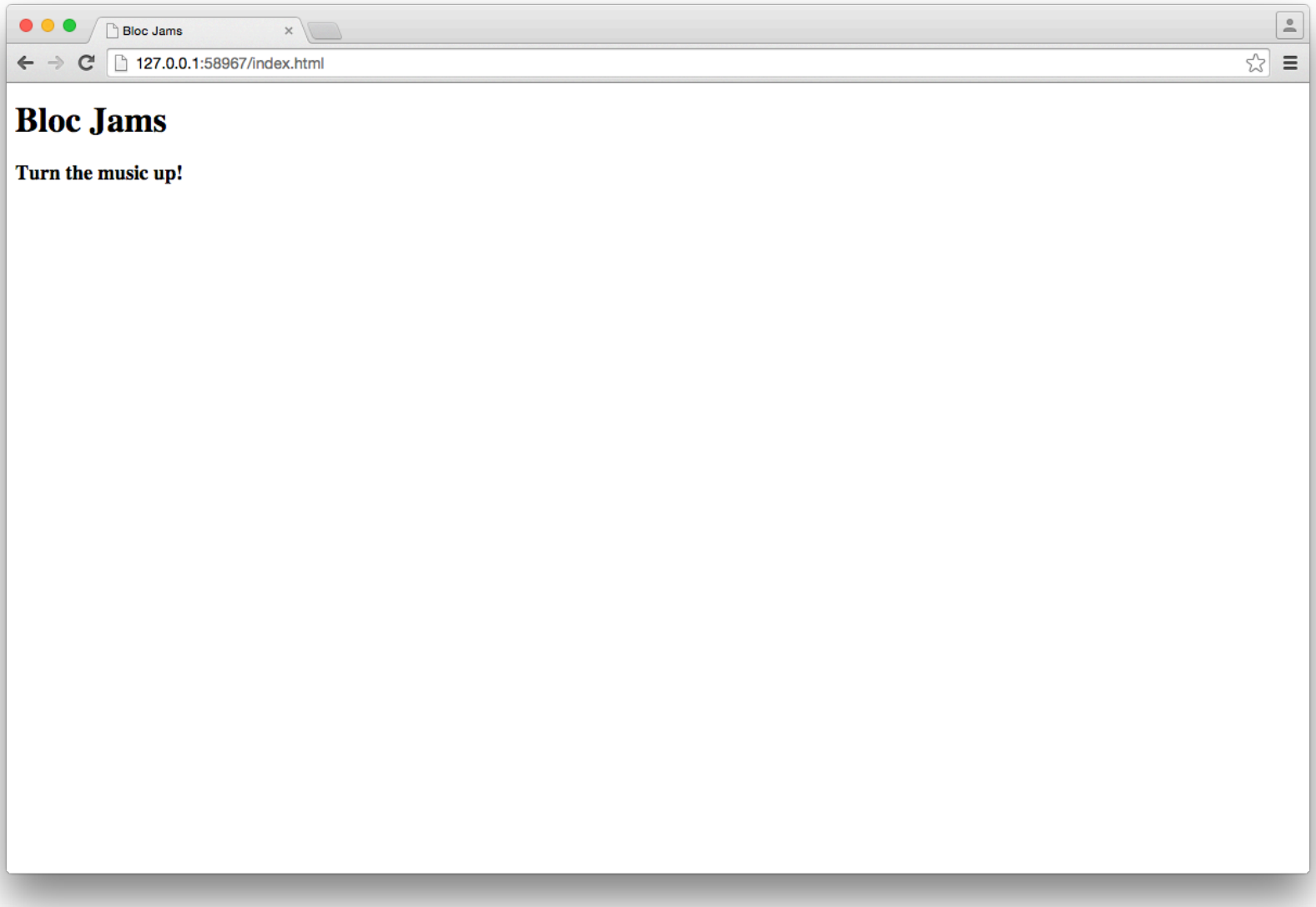
Add the content for the hero container, which will make the name and tagline of the application the focus of the document:

~/bloc/bloc-jams/index.html

```
  <!DOCTYPE html>
  <html>
      <head>
      <title>Bloc Jams</title>
      </head>
      <body>
          <div> <!-- navigation bar -->
          </div>
          <div> <!-- hero content -->
+             <h1>Bloc Jams</h1>
+             <h3>Turn the music up!</h3>
          </div>
          <div> <!-- selling points -->
          </div>
      </body>
  </html>
```

Notice that when we add content within a tag, we indent it four spaces. Putting tags within tags is known as **nesting** tags. Indenting nested tags improves readability and visually establishes a hierarchy in the document.

We've nested an `<h1>` and `<h3>` in our hero `<div>`. `<h1>` and `<h3>` are both **heading tags**. Heading tags are generally used for titles, section names, or important text. They range from `<h1>` (the largest) to `<h6>` (the smallest), and are automatically rendered by browsers in different sizes. See how the browser differentiates between an `<h1>` and an `<h3>` in the screenshot below.



In the following video, we walk through the basic HTML structure described so far:

Add Selling Points

Bloc Jams has a few selling points that visitors to the landing page should see, which we'll add to the corresponding `<div>`:

~/bloc/bloc-jams/index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Bloc Jams</title>
    </head>
    <body>
        <div> <!-- navigation bar -->
        </div>
        <div> <!-- hero content -->
            <h1>Bloc Jams</h1>
            <h3>Turn the music up!</h3>
        </div>
        <div> <!-- selling points -->
+           <div>
+               <h5>Choose your music</h5>
+               <p>The world is full of music; why should you have to listen to music
+           </div>
+           <div>
+               <h5>Unlimited, streaming, ad-free</h5>
+               <p>No arbitrary limits. No distractions.</p>
+           </div>
+           <div>
+               <h5>Mobile enabled</h5>
+               <p>Listen to your music on the go. This streaming service is availab
+           </div>
        </div>
    </body>
</html>
```

The `<p>` element is used for including paragraphs or longer blocks of text. We use it here for the description of each selling point.

For most HTML tags, text is rendered without regard to *whitespace*. **Whitespace** is the term used to describe spaces, tabs, and new lines in formatted text. This can change with CSS, or it can be explicitly defined and rendered using a `<pre>` tag.

Recap

Concept	Description
HTML	HyperText Markup Language makes up the bulk of the Internet. All webpages, including this one, are built in HTML. Browsers render HTML text into a visual interface.

	• **HTML comments**
Tags	An HTML tag names an element with angle brackets `<>`. An end tag closes an opening tag and has a forward slash after the first angle bracket (e.g. `</div>`). • `<title>` • `<h1>` **...** `<h6>` • `<p>` • `<pre>`
Elements	An HTML element is an individual component of a web page.
Nesting	Place nested block elements on a new line and **indent the line** for readability. **Nested inline tags** should not be placed on their own line.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

3 HTML & CSS: Styles

Overview and Purpose

This checkpoint introduces CSS (Cascading Style Sheets) and basic style properties.

Objectives

After this checkpoint, students should be able to:

- Include CSS in an HTML document using external stylesheets.
- Discuss other methods by which styles can be added to an HTML document.
- Explain why resources such as `normalize.css` exist.
- Add classes to HTML tags and use class names to style elements.
- Identify selectors, properties, and values in a CSS file.
- Recognize HTML attributes – such as `rel`, `type`, `href`, `src`, etc. – and discuss the purpose of common attributes.

We left our landing page looking *really* exciting at the end of the last checkpoint. To build on that excitement, we'll be adding styles to our landing page that will add color, placement, icons, and other visual modifications to give it a more modern appearance. Websites are styled using CSS (Cascading Style Sheets), which acts as a set of instructions for how a website should look based on the browser's interpretation.

Before styling Bloc Jams, create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Start your editor's live preview (in Atom, you can use the keyboard shortcut `CTRL+ALT+L` or select "Packages -> atom-live-server -> Start Server" in the main menu). By refreshing the browser tab after changes, you'll be able to see how CSS affects the appearance of the page.

The `<link>` Tag and External Stylesheets

Styles can be included in a web browser in a few ways:

- in the `<head>` tag directly, delimited by `<style>` tags
- in an external stylesheet, which is referenced by a `<link>` tag in the head
- on an HTML element directly, using a `style` attribute
- dynamically produced using JavaScript

The following video demonstrates the different ways to add styles:

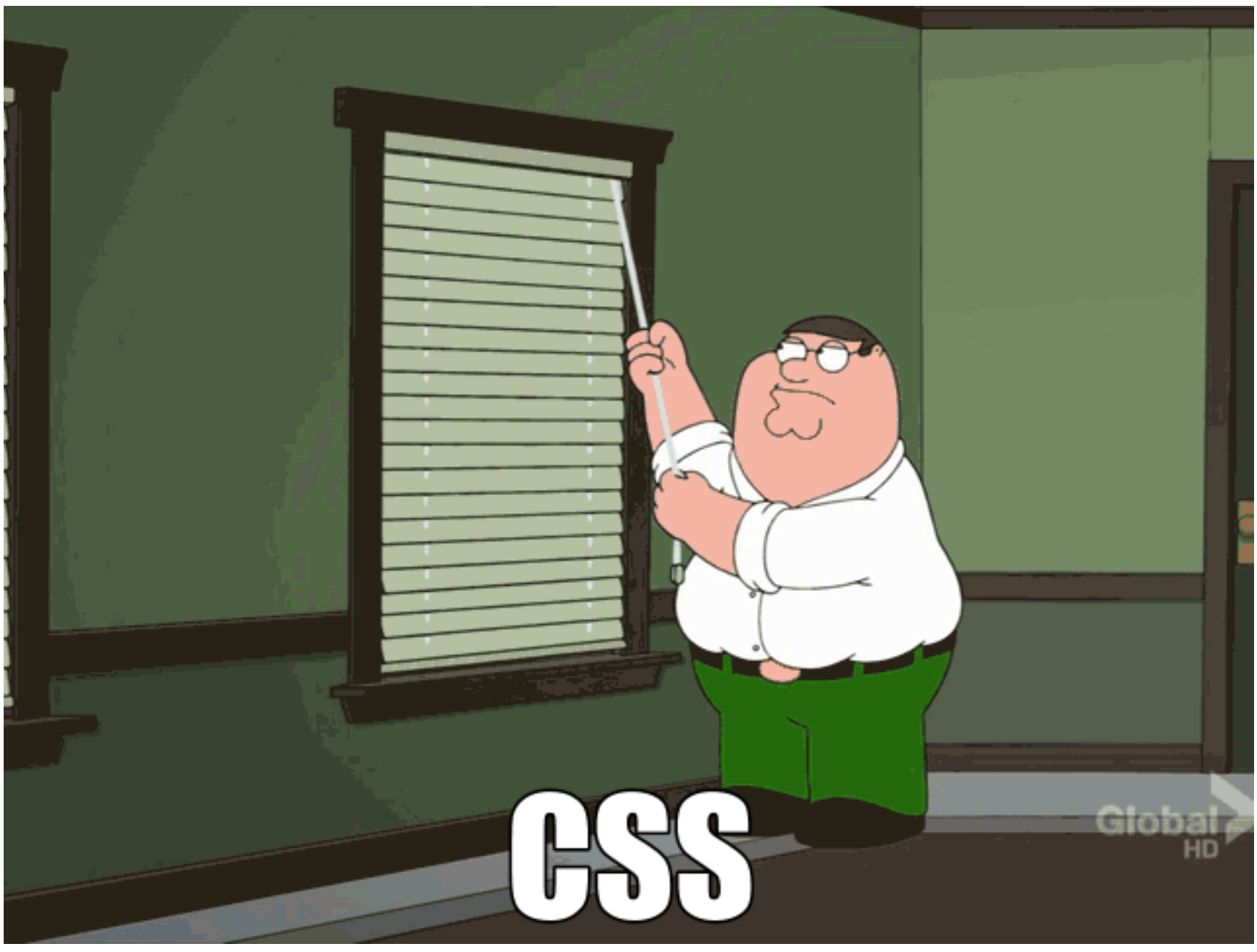
We're going to start by creating and including an external stylesheet for the Bloc Jams landing page.

Create a directory to hold all of the Bloc Jams stylesheets and add our first CSS file, `main.css` to it:

```
~/bloc/bloc-jams/
```

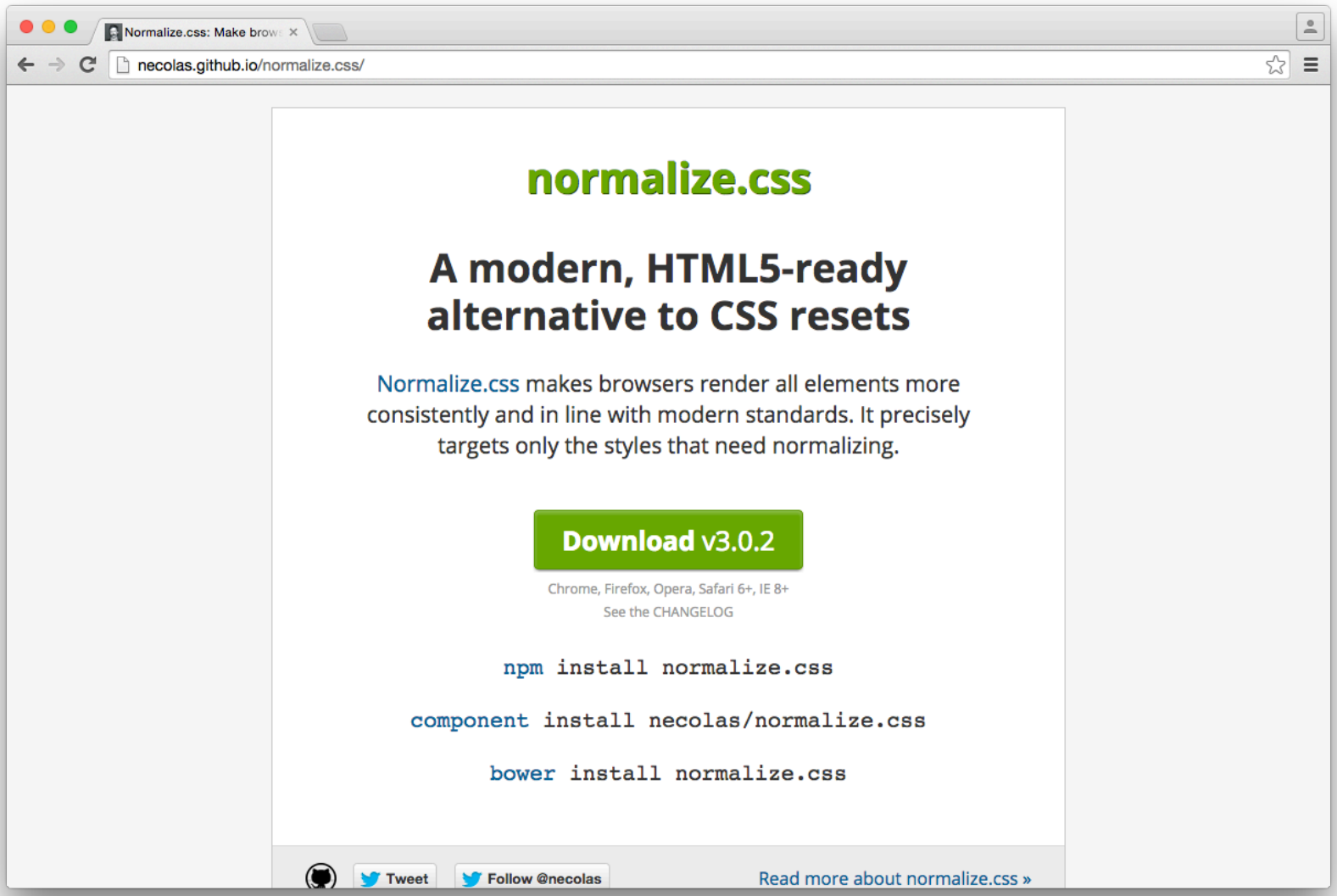
```
$ pwd #=> should be in the bloc-jams project folder
$ mkdir styles
$ touch styles/main.css
```

Download and Include `normalize.css`



> This is what working with CSS can feel like sometimes.

Applying CSS may cause frustration because each browser renders certain styles differently. To minimize this problem, we're going to include `normalize.css`, which "makes browsers render all elements more consistently and in line with modern standards".



Download the `normalize.css` **file** and drag it into the `styles` directory using Finder (on Mac) or File Explorer (on Windows).

Apply its CSS rules to the index page using a `<link>` tag in the head:

~/bloc/bloc-jams/index.html

```
  ...
      <head>
          <title>Bloc Jams</title>
+         <link rel="stylesheet" type="text/css" href="styles/normalize.css">
      </head>
  ...
```

> The ellipsis at the beginning or end of a code block signifies that more code has been written above or below the current code.

The `<link>` tag has some extra inline information: `rel`, `type`, and `href`. These are known as HTML **attributes**. The `rel` attribute defines the **relationship** between the linked file and the HTML document. The `type` attribute lets the page know which **media type** the linked document contains; the most common current media type is `text/css`, which indicates a Cascading Style Sheet format.

The `href` attribute provides a **hyperlink reference** to the file, which is a fancy way of

saying where the file is located. In this case, it's a relative file path for the stylesheet (`bloc-jams` > `styles` > `normalize.css`), but it could be a reference to a remote file. We'll show how to link to a remote file in the next section.

Include Other Linked Stylesheets

We'll also link to two remotely hosted CSS files. These files, found at

`href="http://fonts.googleapis.com/css?family=Open+Sans:400,800,600,700,300"`

and

`href="http://code.ionicframework.com/ionicons/2.0.1/css/ionicons.min.css"`

include the fonts and icons, respectively, that we'll use on the homepage. Then link the `main.css` file, which will have our custom CSS:

~/bloc/bloc-jams/index.html

```
  ...
      <head>
          <title>Bloc Jams</title>
+         <link rel="stylesheet" type="text/css" href="http://fonts.googleapis.com/css?
+         <link rel="stylesheet" type="text/css" href="http://code.ionicframework.com/i
          <link rel="stylesheet" type="text/css" href="styles/normalize.css">
+         <link rel="stylesheet" type="text/css" href="styles/main.css">
      </head>
  ...
```

The order of the CSS files is important. If there are any conflicting styles in the stylesheets listed, then the browser only pays attention to the bottommost definition. For example, if we applied styles to a `.my-div` selector in both `normalize.css` and `main.css`, the browser would render the definition found in `main.css`.

For Bloc Jams, we've opted to use the Open Sans font made available by Google Fonts. In the following video, we demonstrate how to find and include Google Fonts:

Style Bloc Jams

Create a new directory called "assets" and within that directory create another called "images".

~/bloc/bloc-jams/

```
$ pwd #=> make sure you're in the base directory, ~/bloc/bloc-jams
$ mkdir assets
$ mkdir assets/images
```

Download the **logo** and the **background image**. Visit each of those links, right-click on the image, and select "Download" or "Save as" to save the images on your computer.

Move the logo and background into the `images` directory.

Define Styles in `main.css`

Apply CSS to the body and nav bar of Bloc Jams. Update `index.html` with the following:

~/bloc/bloc-jams/index.html

```
  ...
      </head>
-     <body>
+     <body class="landing">
-         <nav> <!-- nav bar -->
+         <nav class="navbar"> <!-- nav bar -->
+             <img src="assets/images/bloc_jams_logo.png" alt="bloc jams logo" class="
          </nav>
  ...
```

We've added **classes** to our HTML elements on the `<body>` and `<nav>` tags. Classes are the most common way for CSS to apply styling to selected elements.

We've also added an `<img>` tag to our navigation for the Bloc Jams logo.

Attribute	Description
`src`	Like the `href` attribute, it links to the source of the image. In this case, the link is relative to the directory containing `index.html`.
`alt`	Tells the browser what text to show if the image doesn't load. This also provides an alternate for visually-impared users.
`class`	Assigns one or more names to this element. These names associate the element with styles defined in the CSS.

Switch to `main.css` and add the following text to see how CSS is applied using **selectors**. We explain the styles using CSS comments, which are delimited by `/*` and `*/`.

~/bloc/bloc-jams/styles/main.css

```
+ html {
+     height: 100%; /* makes sure our HTML takes up 100% of the browser window */
+     font-size: 100%;
+ }
+
+ body {
+     font-family: 'Open Sans'; /* sets our font to the "Open Sans" typeface */
+     color: white;             /* sets the text color to white */
+     min-height: 100%;         /* says the height of the body must be, at minimum, 100
+ }
+
+ body.landing {
+     background-color: rgb(58,23,63); /* sets the background color to an RGB value (se
+ }
```

Selecting elements with CSS can be done a number of ways. In this block of CSS, we've selected the `html` and `body` elements and applied styling using CSS *properties*. **Properties**, such as `height`, `color`, or `font-size` used above, define how a style should look on the page.

A CSS **rule-set** consists of a selector followed by a declaration block:

Selector	Declaration Block
`body`	`{ color: white; }`

The declaration block, which starts with a `{` and ends with a `}`, can have any number of declarations, each of which contains a property-value pair that ends with a semicolon:

Property	Value
`color:`	`white;`

When a selector consists of an HTML tag name immediately followed – as in, not separated by a space – by a class name, as with `body.landing`, the style which follows applies to any `body` tag which has a `landing` class. It's a more specific way of selecting an element and gives the styles attributed to that selector greater importance within the cascade; for example, if the `body` selector also had a `background-color` property applied to it, the `background-color` property of the `body.landing` selector would have greater importance and overrule it.

> Class selectors are preceded by periods (`.landing`) while HTML element selectors are not (`html` and `body`).

The `background-color` property represents the color we've chosen using an **RGB value**, which is one of several ways to identify color in CSS. The RGB color model displays color by selecting the proportion of red, green, and blue, respectively, in a range from 0 to 255.

The table below shows equivalent CSS color values using color names, hexadecimals, and RGB values:

Color Name	Hex Code	RGB Color Model
`white`	`#FFFFFF`	`rgb(255,255,255)`
`red`	`#FF0000`	`rgb(255,0,0)`
`lime`	`#00FF00`	`rgb(0,255,0)`
`blue`	`#0000FF`	`rgb(0,0,255)`
`black`	`#000000`	`rgb(0,0,0)`

> The web color named "lime" actually corresponds to the green primary of an RGB display.

Add some more styles for the navigation bar:

~/bloc/bloc-jams/styles/main.css

```
  ...
  body.landing {
      background-color: rgb(58,23,63);
  }
+
+ .navbar {
+     padding: 0.5rem;
+     background-color: rgb(101,18,95);
+ }
+
+ .navbar .logo {
+     position: relative; /* positioning will be explained in a resource */
+     left: 2rem;
+ }
```

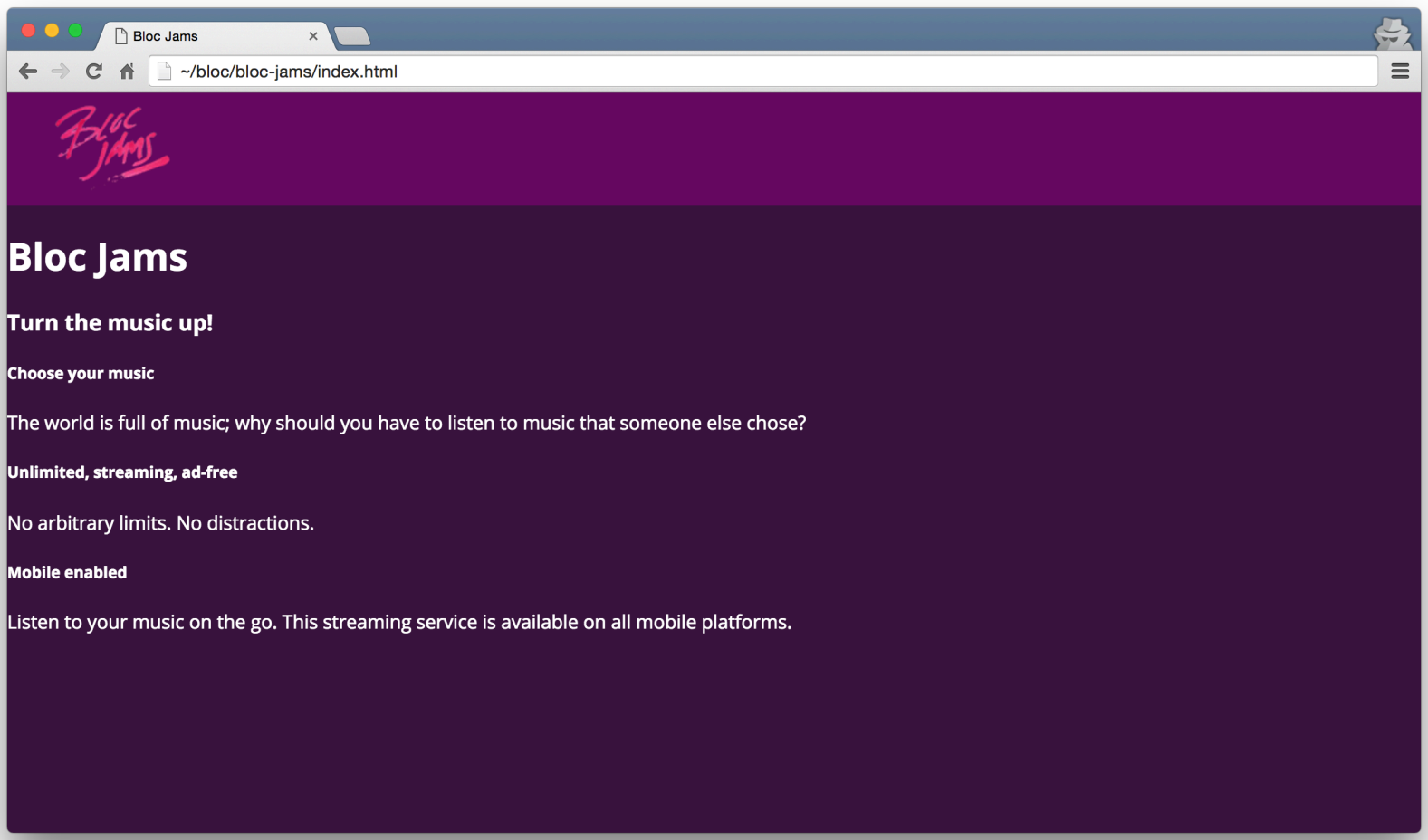
We've used `padding` to provide some space on the *inside* of the navigation bar.

The measurement we've used for padding is called a **root em**. An `em` is a relative unit for calculating the size of an element; it's called an `em` because it's defined to be the width of the letter "m" in a particular font and size of text. *Root* `em`s (`rem`) represent the `font-size` of the root element, such as the `<html>` element. We've chosen this metric to make responsive sizing easier.

> Note that we'll be explaining padding and its context in the CSS Box Model in an external resource.

Notice that we've put `.logo` right after the `.navbar` class selector with a space separating them. This is called a **descendant selector**. The browser applies the style to elements of the `logo` class which are contained in elements of the `navbar` class..

This is different from `body.landing`, which selects the `body` element that has the `.landing` class applied to it, not to elements nested within it.



> Bloc Jams should look like this after adding the navigation classes.

Hero Content

A "hero" container derives its name from theater and film, where a hero prop (or "hero", for short) is intended to be seen more closely and in greater detail than the others.

We'll update the hero content with the following classes:

~/bloc/bloc-jams/index.html

```
  ...
          </nav>
-         <section> <!-- hero content -->
+         <section class="hero-content"> <!-- hero content -->
-             <h1>Bloc Jams</h1>
-             <h3>Turn the music up!</h3>
+             <h1 class="hero-title">Turn the music up!</h1>
          </section>
          <section> <!-- selling points -->
  ...
```

and `main.css` with the following styles:

~/bloc/bloc-jams/styles/main.css

```
  ...
+ .hero-content {
+     position: relative; /* positioning will be explained in a resource */
+     min-height: 600px;  /* hero content element must be _at least_ 600 pixels */
+     background-image: url(../assets/images/bloc_jams_bg.jpg);
+     background-repeat: no-repeat;
+     background-position: center center;
+     background-size: cover;
+ }
```

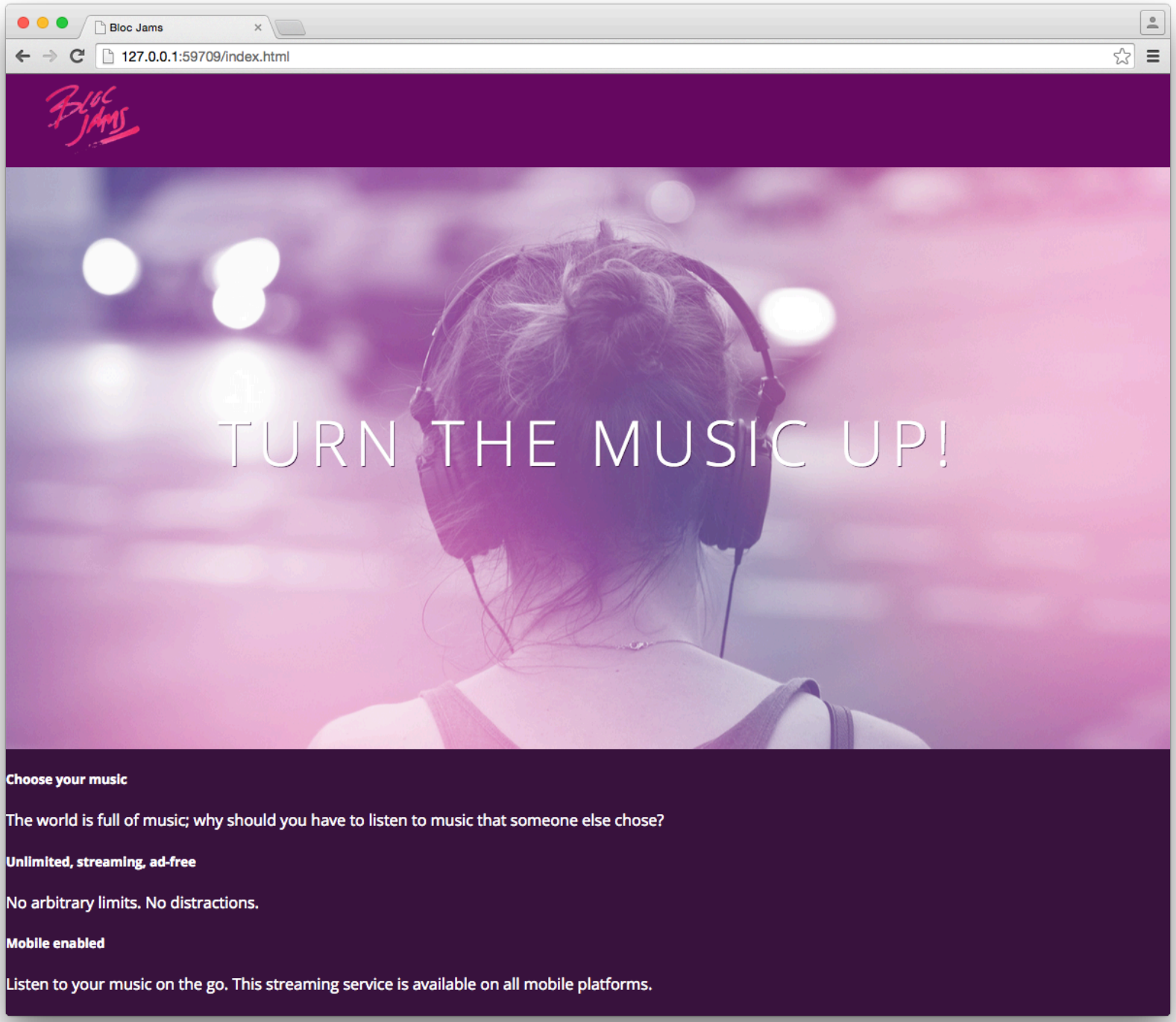
The background-related properties tell our app that we want to use a specific image for the background. `background-image` specifies the image to use. The next property tells the browser not to repeat the image. We also tell the browser to center the image both horizontally (the first `center` in `background-position`) and vertically (the second). The `cover` value of the of `background-size` scales the image so the background is always covered by the image.

Add the styles for the hero content's title:

~/bloc/bloc-jams/styles/main.css

```
  ...
+ .hero-content .hero-title {
+     position: absolute;              /* positioning will be explained in a resource */
+     top: 40%;                        /* positioning will be explained in a resource */
+     -webkit-transform: translateY(-50%);
+     -moz-transform: translateY(-50%);
+     -ms-transform: translateY(-50%);
+     transform: translateY(-50%);
+     width: 100%;                     /* makes sure the title inhabits the full width c
+     text-align: center;              /* aligns text in the horizontal center of the el
+     font-size: 4rem;                 /* specifies the size of the font in root em unit
+     font-weight: 300;                /* makes the font thinner or lighter weight */
+     text-transform: uppercase;       /* transforms the text to be all uppercase letter
+     letter-spacing: 0.5rem;          /* puts 0.5 root ems worth of space between each
+     text-shadow: 1px 1px 0px rgb(58,23,63); /* creates a shadow underneath the text f
+ }
```

The prefixes with names in dashes, like `-webkit-`, are known as **vendor prefixes**. Vendor prefixes are required for any CSS property that hasn't been officially adopted into the CSS3 specification. We're using the `transform` property and the `translateY` value to vertically center the text in the hero-content container. The three prefixes identify the transform property for the Webkit, Mozilla, and Microsoft-based browser, respectively.



> What the hero content should look like after you've added the CSS classes.

Selling Points

Add the CSS classes for the Bloc Jams selling points to the HTML:

~/bloc/bloc-jams/index.html

```
  ...
-         <section> <!-- selling points -->
-             <div>
-                 <h5>Choose your music</h5>
-                 <p>The world is full of music; why should you have to listen to music
+         <section class="selling-points"> <!-- selling points -->
+             <div class="point">
+                 <span class="ion-music-note"></span>
+                 <h5 class="point-title">Choose your music</h5>
+                 <p class="point-description">The world is full of music; why should y
              </div>
-             <div>
-                 <h5>Unlimited, streaming, ad-free</h5>
-                 <p>No arbitrary limits. No distractions.</p>
+             <div class="point">
+                 <span class="ion-radio-waves"></span>
+                 <h5 class="point-title">Unlimited, streaming, ad-free</h5>
+                 <p class="point-description">No arbitrary limits. No distractions.</p
              </div>
-             <div>
-                 <h5>Mobile enabled</h5>
-                 <p>Listen to your music on the go. This streaming service is availab
+             <div class="point">
+                 <span class="ion-iphone"></span>
+                 <h5 class="point-title">Mobile enabled</h5>
+                 <p class="point-description">Listen to your music on the go. This str
             </div>
  ...
```

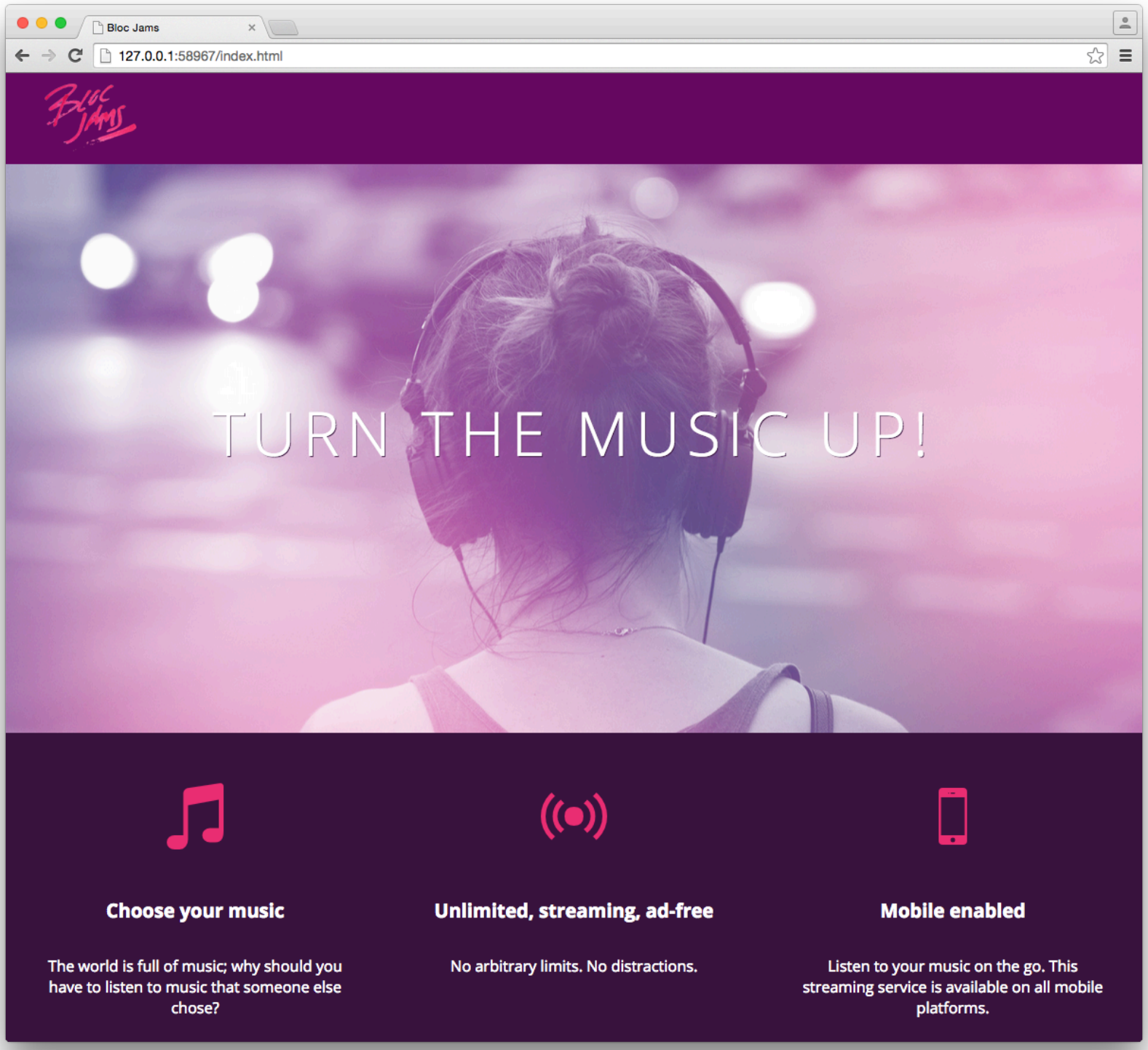
> Note: we've added the icon elements that were provided by the **Ionicons font library** that we added to the header earlier.

Add the styles for the selling points:

~/bloc/bloc-jams/styles/main.css

```
   ...
+  .selling-points {
+      position: relative;
+      display: table;
+      width: 100%;
+      -webkit-box-sizing: border-box;
+      -moz-box-sizing: border-box;
+      box-sizing: border-box; /* we'll explain this in a resource */
+  }
+
+  .point {
+      display: table-cell;
+      position: relative;
+      width: 33.3%;
+      padding: 2rem;
+      text-align: center;
+  }
+
+  .point .point-title {
+      font-size: 1.25rem;
+  }
+
+  .ion-music-note,
+  .ion-radio-waves,
+  .ion-iphone {
+      color: rgb(233,50,117);
+      font-size: 5rem;
+  }
```

Most of the CSS above is familiar at this point. `display: table;` and its descendant `display: table-cell;` place the elements in a table-like layout (think of a spreadsheet), with each selling point sitting side-by-side in a grid.



> The landing page after applying all the styles in this checkpoint.

Recap

Concept	Description
CSS	Cascading Style Sheets control the look and formatting of a website.
Properties	HTML elements are styled with CSS properties. • `background` • `background-position`

RGB color model	Colors can be displayed as hexadecimals (like `#FFFFFF`) or using `rgb(#,#,#)`, where # respectively represents red, green, and blue with a value range of 0-255.
CSS Units	We use font-relative length units, such as `em` and `rem`, to create scalable layouts.
Selectors	Selectors are patterns that reference HTML elements for styling.
Descendant Selectors	Combine two or more selectors with a space to create a descendant selector. A descendant selector references element(s) that exist within a specified ancestor element.
Vendor Prefixes	Some browsers only support certain CSS properties with the use of a vendor prefix.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

4 HTML & CSS: Responsiveness

Overview and Purpose

This checkpoint introduces media queries, a CSS technique used to apply styles based on specified conditions.

Objectives

After this checkpoint, students should be able to:

- Explain what a viewport is and the purpose of a viewport meta tag.
- Explain the use of and know which symbol represents the universal selector.
- Recognize and understand the use of CSS pseudo-elements such as `::before` and `::after`.
- Discuss the benefits of multiple CSS files for a single application.
- Add media queries to a CSS file.
- Discuss the benefits of a grid system.

The Bloc Jams landing page looks nice on a desktop computer. However, we want it to look good on as many devices as possible. We will use the principles of **responsive web development** to define how Bloc Jams should look and function on mobile devices. Responsive web development describes the process which requires us to apply styles that allow our website to adapt to a variety of screen dimensions.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Set the Viewport

The **viewport meta tag** is an HTML tag that indicates how the browser should render the

content of the page in a device's viewport. The **viewport** is a virtual window, like a browser window, that contains the display of the browser's content. Since the viewport tag is a `<meta>` tag, it belongs inside the `<head>` tag:

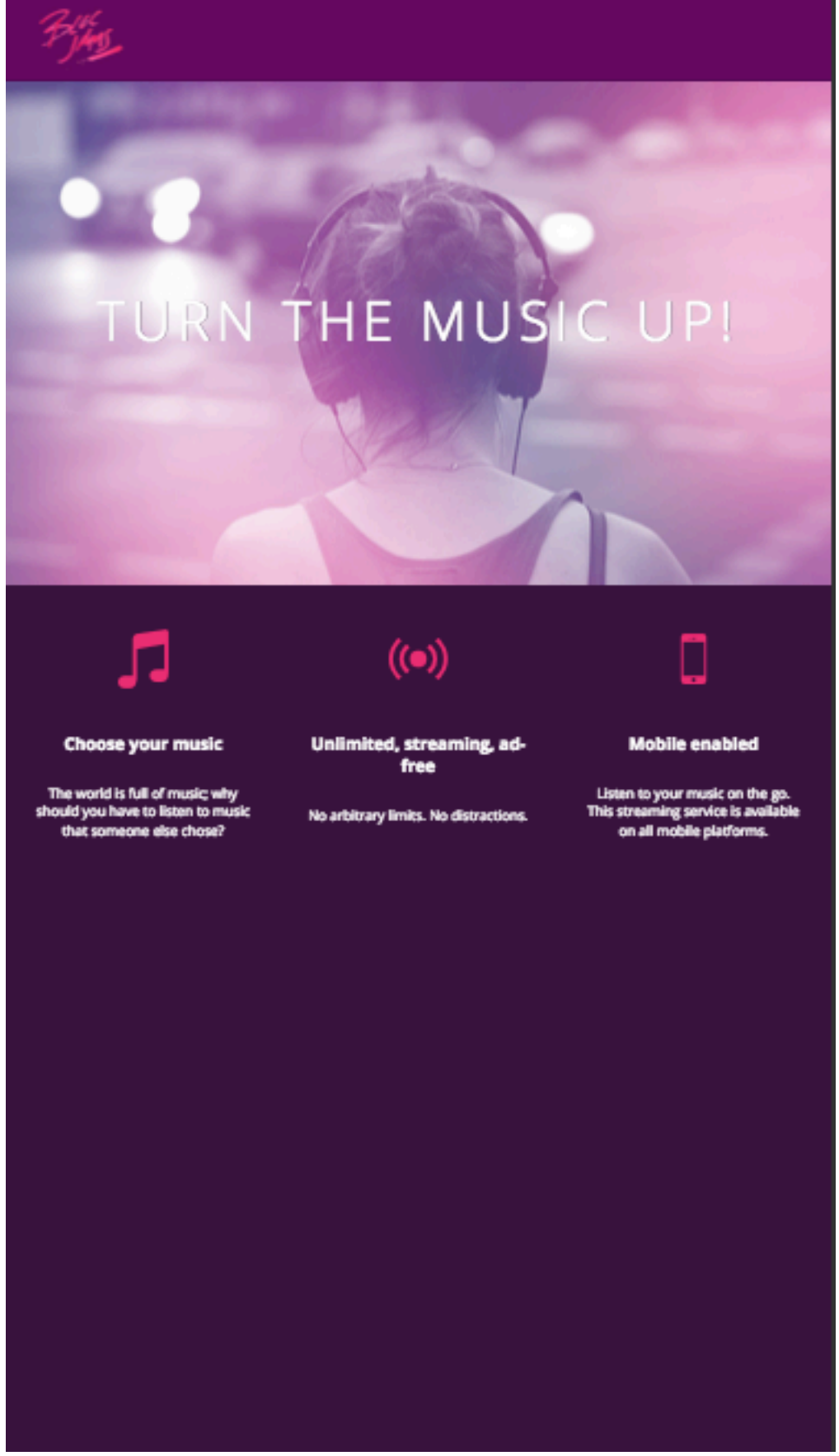
~/bloc/bloc-jams/index.html

```
  ...
  <head>
      <title>Bloc Jams</title>
+     <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" type="text/css" href="http://fonts.googleapis.com/css?fam
      <link rel="stylesheet" type="text/css" href="http://code.ionicframework.com/ioni
      <link rel="stylesheet" type="text/css" href="styles/normalize.css">
      <link rel="stylesheet" type="text/css" href="styles/main.css">
  </head>
  ...
```

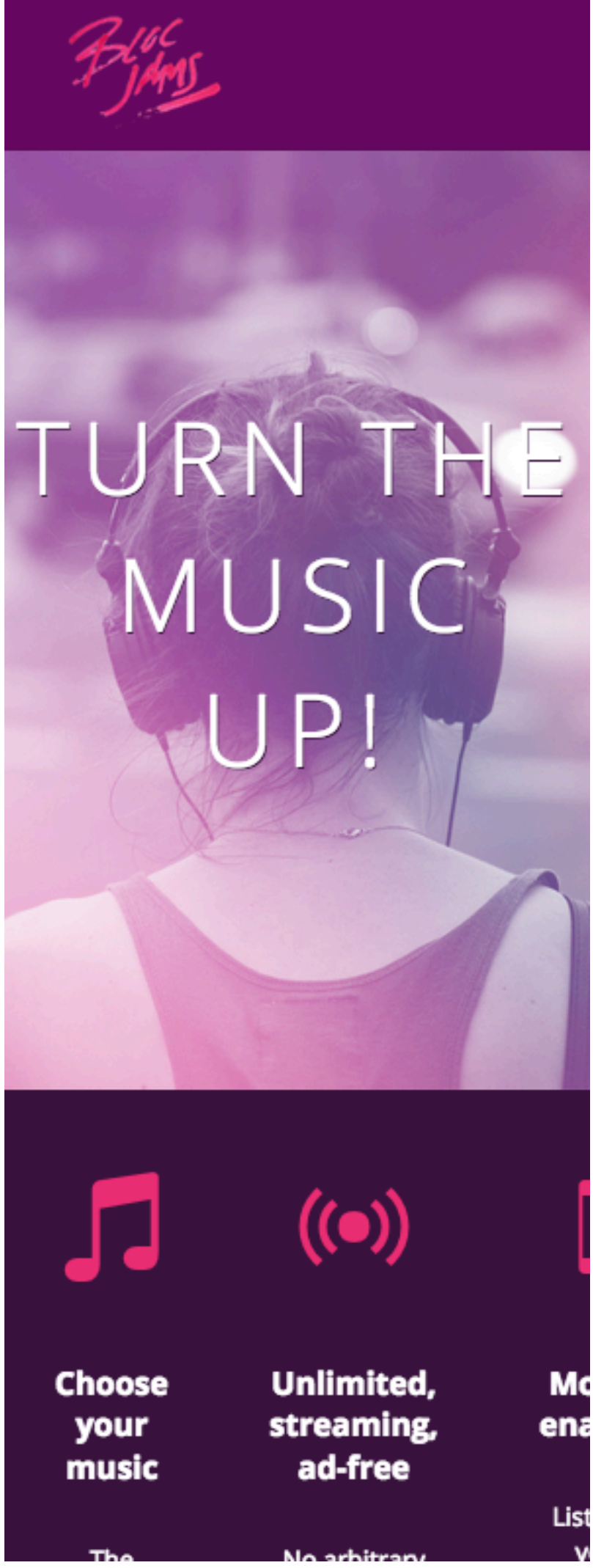
The `content` attribute has properties with values, separated by a comma:

Property/Value	Description
`width=device-width`	Sets the width of the viewport to conform to the width of the device viewing the page.
`initial-scale=1`	Sets the zoom level of the page when it's loaded. With a value of 1, the content remains unscaled. It is rendered as is, based on the CSS rules.

This is what Bloc Jams looks like on a mobile device *without* a viewport tag:



and with a viewport tag:



This difference is the reason why viewport `<meta>` tag is required: to properly render the HTML and CSS for each device. The page scales to fit the width. The selling points exceed the width of the device, but we'll fix that with responsive CSS.

Add `box-sizing: border-box`

We discussed the `border-box` value in the **Box Model resource**; it's important that we

use it here. We want to keep the 2rem padding that we applied to the `point` class. However, we don't want the padding to affect the width of the elements, so they will maintain a consistent ratio for both small and large screens.

Add the following to the top of `main.css`:

~/bloc/bloc-jams/styles/main.css

```
+ *, *::before, *::after {
+     -moz-box-sizing: border-box;
+     -webkit-box-sizing: border-box;
+     box-sizing: border-box;
+ }
  ...
```

The `*` is known as the **universal selector**. It selects all of the elements on a page and applies styles to them. Generally, the universal selector performs poorly on larger web pages, so be careful with how and when you use it. We use it here to make sure all the elements in Bloc Jams are sized using `border-box`.

`::before` and `::after` (sometimes written with only one colon, like `:before` and `:after`) create **pseudo-elements** that are the first and last child of the matched element, respectively.

Create the `landing.css` File

As we add pages to Bloc Jams, styles become difficult to navigate when kept in a single file like `main.css`. It is common practice to organize styles into multiple files where each file serves a specific purpose. We want to keep the styles for `main.css` specifically for **global** styles, or styles that apply to many pages in the application.

Create a new stylesheet called `landing.css`, and move the landing page styles into it by copying them from `main.css` and removing them afterward:

~/bloc/bloc-jams/

```
$ pwd #=> should be in ~/bloc/bloc-jams
$ touch styles/landing.css
```

~/bloc/bloc-jams/styles/landing.css

```
+ body.landing {
+     background-color: rgb(58,23,63);
+ }
+
```

```
+ .hero-content {
+     position: relative;
+     min-height: 600px;
+     background-image: url(../assets/images/bloc_jams_bg.jpg);
+     background-repeat: no-repeat;
+     background-position: center center;
+     background-size: cover;
+ }
+
+ .hero-content .hero-title {
+     position: absolute;
+     top: 40%;
+     -webkit-transform: translateY(-50%);
+     -moz-transform: translateY(-50%);
+     transform: translateY(-50%);
+     width: 100%;
+     text-align: center;
+     font-size: 4rem;
+     font-weight: 300;
+     text-transform: uppercase;
+     letter-spacing: 0.5rem;
+     text-shadow: 1px 1px 0px rgb(58,23,63);
+ }
+
+ .selling-points {
+     position: relative;
+     display: table;
+     width: 100%;
+     -webkit-box-sizing: border-box;
+     -moz-box-sizing: border-box;
+     box-sizing: border-box;
+ }
+
+ .point {
+     display: table-cell;
+     position: relative;
+     width: 33.3%;
+     padding: 2rem;
+     text-align: center;
+ }
+
+ .point .point-title {
+     font-size: 1.25rem;
+ }
+
+ .ion-music-note,
+ .ion-radio-waves,
+ .ion-iphone {
```

```
+     color: rgb(233,50,117);
+     font-size: 5rem;
+ }
```

Make sure to remove these styles from `main.css`. Remove the `border-box` style from the `selling-points` class since we applied the style globally:

~/bloc/bloc-jams/styles/landing.css

```
  .selling-points {
      position: relative;
      display: table;
      width: 100%;
-     -webkit-box-sizing: border-box;
-     -moz-box-sizing: border-box;
-     box-sizing: border-box;
  }
```

And add a `<link>` to `landing.css` in the `<head>` tag:

~/bloc/bloc-jams/index.html

```
  ...
  <head>
      <title>Bloc Jams</title>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" type="text/css" href="http://fonts.googleapis.com/css?fam
      <link rel="stylesheet" type="text/css" href="http://code.ionicframework.com/ioni
      <link rel="stylesheet" type="text/css" href="styles/normalize.css">
      <link rel="stylesheet" type="text/css" href="styles/main.css">
+     <link rel="stylesheet" type="text/css" href="styles/landing.css">
  </head>
  ...
```

The final `landing.css` file should look **like this** and the refactored `main.css` file should look **like this**.

Add Media Queries for Responsive Breakpoints

A **responsive grid** separates elements (or groups of elements) into columns and rows that collapse and expand based on device breakpoints. A **breakpoint** is a measurement that corresponds to the width of a device's screen. It marks when certain mobile or desktop styles should be enabled or disabled. A common method for applying styles based on device breakpoints is CSS **media queries**.

Add our first media queries to `main.css`:

~/bloc/bloc-jams/styles/main.css

```
  ...
+
+ /* Medium and small screens (640px) */
+ @media (min-width: 640px) {
+   /* Style information will go here */
+ }
+
+ /* Large screens (1024px) */
+ @media (min-width: 1024px) {
+   /* Style information will go here */
+ }
```

The `@media` queries we've defined correspond to the root em measurements for medium and large (desktop) devices. We chose values that look good on a number of devices in favor of a more granular spectrum of breakpoints. `1024px` works well for both laptops and tablets, and `640px` works well for smaller tablets and bigger phones. We'll add more styles in the coming sections for devices that are smaller than `640px` as well.

Recall that a root em is relative to the default browser font-size, which is 16 pixels in most browsers. 16 pixels is pretty small, particularly for high-resolution devices. Set the font-size above 100% of the default to make our text more legible on all devices:

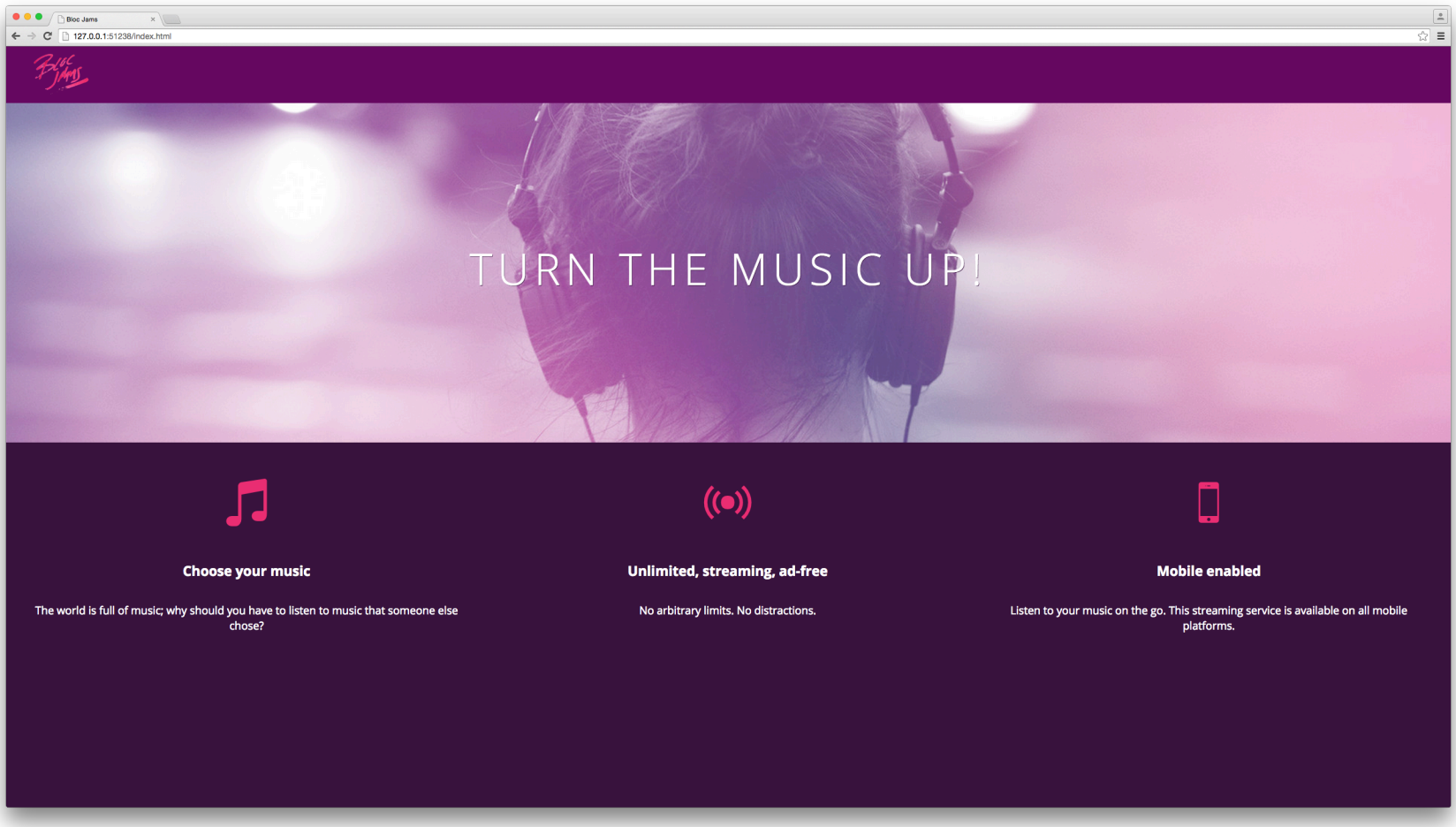
~/bloc/bloc-jams/styles/main.css

```
  ...
  /* Medium screens (640px) */
  @media (min-width: 640px) {
+     html { font-size: 112%; }
  }

  /* Large screens (1024px) */
  @media (min-width: 1024px) {
+     html { font-size: 120%; }
  }
```

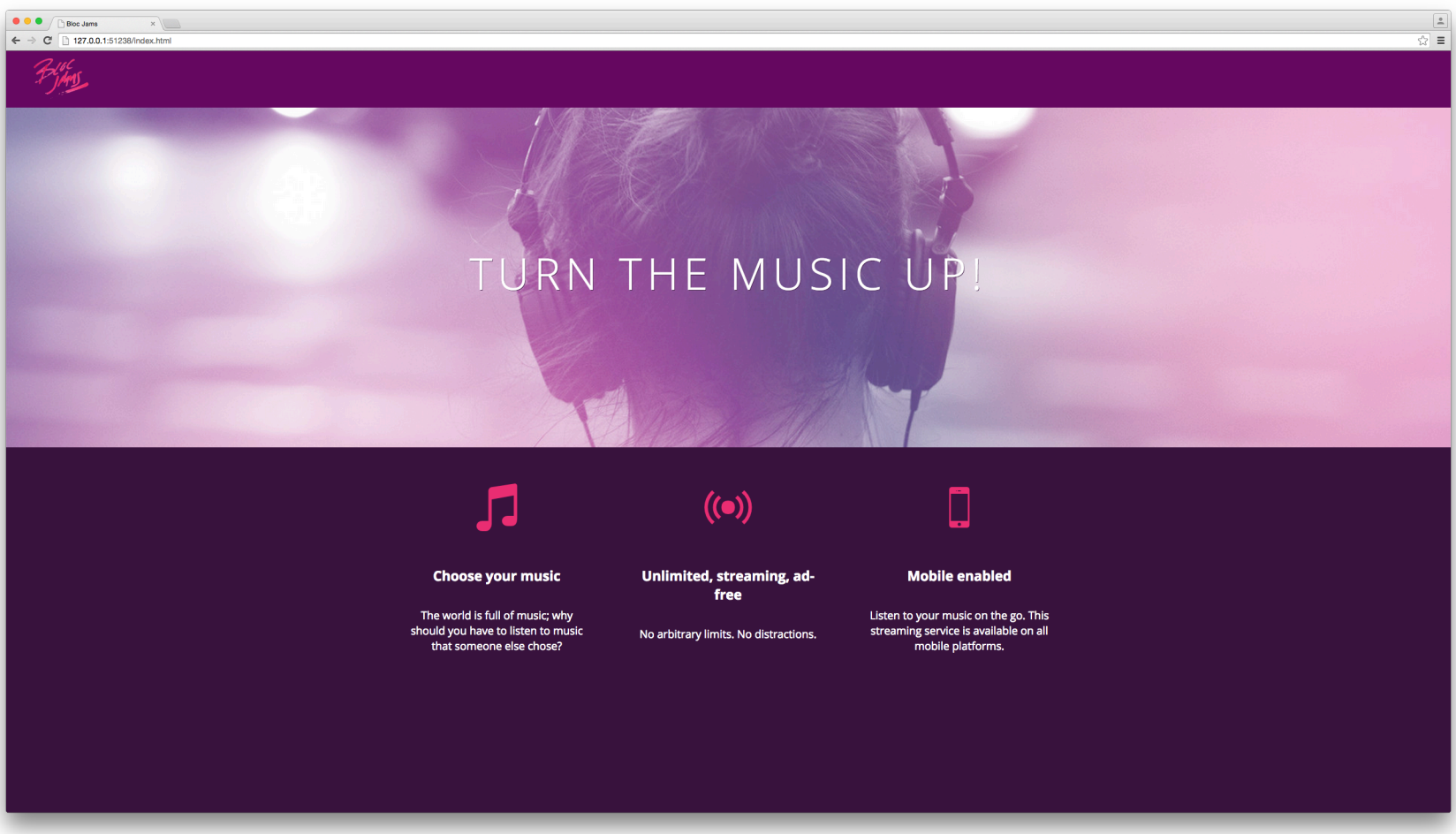
When a browser sets a default text size (which is the basis for root ems) it sets the size *on the* `html` *element*. Therefore, the base text size cannot be a relative unit because there is no parent element to which to relate it.

Add a Max-width Container

Bigger screens cause a website to stretch horizontally in a way that can warp elements and make text difficult to read. Setting a max-width on a container element is a common way to maintain the integrity of a design as the screen expands. Here's what Bloc Jams looks like on a large monitor without a max-width:



and with a max-width:



Set the max-width on a new `container` class, and set the **horizontal** `margin` **value to** `auto` **to center it**:

~/bloc/bloc-jams/styles/main.css

```
  ...
+ .container {
+     margin: 0 auto;
+     max-width: 64rem;
+ }
+
  /* Medium screens (640px) */
  @media (min-width: 640px) {
      html { font-size: 112%; }
  }
  ...
```

Recall that the second argument of the `margin` shorthand determines the size of the margin on the left **and** right side of the element whenever there are only two arguments given. When a width is "fixed" (meaning there is a width or max-width defined), setting the left and right margins to `auto` will center the element.

Add the container class to the `.selling-points` section:

~/bloc/bloc-jams/index.html

```
  ...
-         <section class="selling-points"> <!-- selling points -->
+         <section class="selling-points container"> <!-- selling points -->
  ...
```

Add a Grid System

In responsive web development, a grid system is used to set divisions in a page based on common proportions. Popular implementations include **Bootstrap's** or **Foundation's** grid systems. We'll make our own, simple grid system for Bloc Jams.

Grids set natural changes in the layout at device breakpoints. Add the `.column` under the `min-width: 640px` media query:

~/bloc/bloc-jams/styles/main.css

```
    ...
    @media (min-width: 640px) {
        html { font-size: 112%; }
+
+       .column {
+           float: left;
+           padding-left: 1rem;
+           padding-right: 1rem;
+       }
    }
```

We use the `column` class as a shared class for all items in our grid system. We set the **float** to `left` and add padding to keep some aesthetic space between grid elements. `float: left` ensures that every element with the `column` class will stick to the left-most side in its container.

The browser achieves this by pulling the floated item out of the normal document flow. This is an important distinction because unexpected behavior may occur when floats are not "cleared"; `clear` is a CSS property that specifies whether an element can be next to floating elements that precede it or must be moved down (cleared) below them.

Define additional column classes that divide elements within a container into full-width, halves, thirds, and fourths:

~/bloc/bloc-jams/styles/main.css

```
    ...
    @media (min-width: 640px) {
        html { font-size: 112%; }

        .column {
            float: left;
            padding-left: 1rem;
            padding-right: 1rem;
        }
+
+       .column.full { width: 100%; }
+       .column.two-thirds { width: 66.7%; }
+       .column.half { width: 50%; }
+       .column.third { width: 33.3%; }
+       .column.fourth { width: 25%; }
+       .column.flow-opposite { float: right; }
    }
```

The class names correspond to the fraction of the width they will occupy. We've used the same selector syntax that we used on `body.landing`. We *must* use the proportion classes

(`full`, `half`, etc.) with the `column` class or the styles will not apply.

Add the `third` class to our selling points:

~/bloc/bloc-jams/index.html

```
  ...
-             <div class="point">
+             <div class="point column third">
                  <span class="ion-music-note"></span>
                  <h5 class="point-title">Choose your music</h5>
                  <p class="point-description">The world is full of music; why should y
              </div>
-             <div class="point">
+             <div class="point column third">
                  <span class="ion-radio-waves"></span>
                  <h5 class="point-title">Unlimited, streaming, ad-free</h5>
                  <p class="point-description">No arbitrary limits. No distractions.</p
              </div>
-             <div class="point">
+             <div class="point column third">
                  <span class="ion-iphone"></span>
                  <h5 class="point-title">Mobile enabled</h5>
                  <p class="point-description">Listen to your music on the go. This str
              </div>
  ...
```

Remove the `width` property definition from the `point` and `selling-points` classes in `landing.css` – width is now applied via the `container` and `column` classes. Remove the `display` properties as well. The `column` class's `float` property takes care of displaying our content in a grid, so we no longer need them formatted like a table:

~/bloc/bloc-jams/styles/landing.css

```
  ...
  .selling-points {
      position: relative;
-     display: table;
-     width: 100%;
  }

  .point {
-     display: table-cell;
      position: relative;
      padding: 2rem;
-     width: 33.3%;
      text-align: center;
  }
  ...
```

Recap

Concept	Description
Responsive Web Design	A responsive web site will adapt its layout to a wide range of devices and screen sizes.
Viewport	A viewport controls how a site displays on a mobile device. • `<meta>`
Pseudo-elements	Pseudo-elements like `::before` and `::after` are added to selectors and allow you to style certain parts of a document.
Universal Selector: `*`	The universal selector, represented by the asterisk symbol, matches a single element of any type.
`clear` **CSS property**	Specifies whether an element can be next to floating elements that precede it or must be moved down (cleared) below them.
Breakpoint	A breakpoint is a measurement that corresponds to an approximate screen size of many devices. It marks when certain mobile or desktop styles should be enabled or disabled.

Media Queries	Media queries tailor the presentation of content to a **specific range of output devices** without having to change the content itself.
Grid Systems	A grid system provides a site with solid structure. In responsive web development in particular, grids simplify the division and display of information.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

4. HTML & CSS: Responsiveness

Assignment | **Discussion** | **Submission**

5 HTML & CSS: Collection View

Overview and Purpose

This checkpoint implements a "Collection view" for the Bloc Jams application, which allows a user to view their album collection on a single page.

Objectives

After this checkpoint, students should be able to:

- Create a basic HTML document structure.
- Create links (anchors) to local files.
- Discuss style properties – such as `z-index`, `vertical-align`, `text-decoration`, etc. – and their respective values and units.

To view albums and play songs, we need to see a collection of the available albums that we can play. We also want to continue to use the responsive classes we defined in the last checkpoint, so our album collection looks nice on a mobile device.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Create the Collection View Files

The Collection view needs its own HTML page. Create the file in the base directory, the same place where we created `index.html`:

~/bloc/bloc-jams/

```
$ touch collection.html
```

Create an accompanying CSS file:

~/bloc/bloc-jams/

```
$ touch styles/collection.css
```

Add the HTML Skeleton to `collection.html`

The HTML skeleton for the collection page is similar to the landing page, except that it links to `collection.css` instead of `landing.css`:

~/bloc/bloc-jams/collection.html

```
+ <!DOCTYPE html>
+ <html>
+     <head>
+         <title>Bloc Jams</title>
+         <meta name="viewport" content="width=device-width, initial-scale=1">
+         <link rel="stylesheet" type="text/css" href="http://fonts.googleapis.com/css?
+         <link rel="stylesheet" type="text/css" href="http://code.ionicframework.com/i
+         <link rel="stylesheet" type="text/css" href="styles/normalize.css">
+         <link rel="stylesheet" type="text/css" href="styles/main.css">
+         <link rel="stylesheet" type="text/css" href="styles/collection.css">
+     </head>
+     <body>
+     </body>
+ </html>
```

Update the Navigation to Link to All Bloc Jams Pages

We have two pages in Bloc Jams, and we need a way to navigate between them. We'll make the Bloc Jams logo the link to the landing page and create a separate set of links on the righthand side of the navbar. We may have more pages in Bloc Jams in the future, so we'll create a container to hold any future links to those pages.

Add the new nav markup to `index.html`:

~/bloc/bloc-jams/index.html

```
  ...
      <nav class="navbar"> <!-- nav bar -->
-         <img src="assets/images/bloc_jams_logo.png" alt="bloc jams logo" class="logo"
+         <a href="index.html" class="logo">
+             <img src="assets/images/bloc_jams_logo.png" alt="bloc jams logo">
+         </a>
+         <div class="links-container">
+             <a href="collection.html" class="navbar-link">collection</a>
+         </div>
      </nav>
  ...
```

When linking to other pages, we use an anchor tag (`<a>`) with a defined `href` attribute. Like the `href` in `<link>` tags, the content of the attribute is either a link to a remote location (like another site), or in this case, a relative path to another file (`href="collection.html"`).

We've added some new classes to style the nav. Add the corresponding style rules to `main.css`:

~/bloc/bloc-jams/styles/main.css

```
  ...
  .navbar {
+     position: relative;
      padding: 0.5rem;
-     background-color: rgb(101,18,95);
      /* #1 */
+     background-color: rgba(101,18,95,0.5);
      /* #2 */
+     z-index: 1;
  }

  .navbar .logo {
      position: relative;
      left: 2rem;
+     cursor: pointer;  /* produces the finger-pointer icon when you hover over the log
+ }
+
+ .navbar .links-container {
+     display: table;
+     position: absolute;
+     top: 0;
+     right: 0;
+     height: 100px;
+     color: white;
```

```
      /* #3 */
+     text-decoration: none;
+ }
+
+ .links-container .navbar-link {
+     display: table-cell;
+     position: relative;
+     height: 100%;
+     padding-left: 1rem;
+     padding-right: 1rem;
      /* #4 */
+     vertical-align: middle;
+     color: white;
+     font-size: 0.625rem;
+     letter-spacing: 0.05rem;
+     font-weight: 700;
+     text-transform: uppercase;
      /* #3 */
+     text-decoration: none;
+     cursor: pointer;
+ }
+
    /* #5 */
+ .links-container .navbar-link:hover {
+     color: rgb(233,50,117);
+ }
  ...
```

We've included some new CSS properties in these classes:

- At **#1**: the `rgba()` value is the same as the `rgb()` value for color, except that it takes a fourth argument that determines *alpha* (opacity) of the color. Opacity is measured on a scale from 0 (invisible) to 1 (completely opaque). We've made the navigation element slightly transparent to reveal the new background we'll have on other pages, including the collection page.
- At **#2**: the `z-index` property adds a third dimension to the placement of an element (think of x, y, and z coordinates). If two elements overlap, then the `z-index` determines which of them is displayed on top of the other. By default, `z-index` for an element is set by the browser (`auto`), but we want the nav to sit above all adjacent elements. Elements with larger z-index values sit above lesser ones.
- At **#3**: the `text-decoration` property changes the "decorative" properties of text, allowing for the addition of underlines, strikethroughs, and more. By default, anchor tags (`<a>`) are decorated with an underline, so setting `text-decoration` to `none` removes it. Try commenting out this style to see what the underline looks like.
- At **#4**: `vertical-align` orients `inline` or `table-cell` content vertically. The `middle` value centers our nav links vertically within their containers.

There's also a pseudo-class selector at **#5**, `:hover`, that's appended to the `.navbar-link` class. A "hover" state of an element occurs when a cursor mouses over it. Test this out in your editor's live preview. Note that when hovering over the "Collection" link, it changes to the color specified under the `:hover` style rule.

We want the nav updates reflected on both the landing and collection pages. Add the same markup to `collection.html`:

~/bloc/bloc-jams/collection.html

```
  ...
      <body>
+         <nav class="navbar"> <!-- nav bar -->
+             <a href="index.html" class="logo">
+                 <img src="assets/images/bloc_jams_logo.png" alt="bloc jams logo">
+             </a>
+             <div class="links-container">
+                 <a href="collection.html" class="navbar-link">collection</a>
+             </div>
+         </nav>
      </body>
  ...
```

Download Blurred Backgrounds and Album Placeholder Images

The blurred background effect on the collection page and the images for each album are in downloadable .zip files.

Downloads:

- Download the **album placeholder images**.
- Download the **blurred backgrounds**.

Extract or unzip the files:

- On OS X, double-clicking the .zip file will automatically extract it.
- On Windows, double-clicking the .zip file will open the "Extract all" option at the top of the File Explorer Window. More details can be found in **this Microsoft support document**.

Move the *unzipped* folders to the `bloc-jams/assets/images` directory, where the rest of the Bloc Jams images reside.

Do not rename the folders. Doing so will prevent the images from displaying properly. **Double check that the placeholder image directory is called `album_covers` and that the backgrounds directory is called `blurred_backgrounds`** before continuing with the rest of the checkpoint.

Add `collection.html` Content

Populate the Collection view with the following HTML:

~/bloc/bloc-jams/collection.html

```
  ...
      </nav>
+
+     <section class="album-covers container clearfix">
+         <div class="collection-album-container column fourth">
+             <img src="assets/images/album_covers/01.png"/>
+             <div class="collection-album-info caption">
+                 <p>
+                     <a class="album-name" href="#">The Colors</a>
+                     <br/>
+                     <a href="#">Pablo Picasso</a>
+                     <br/>
+                     X songs
+                     <br/>
+                 </p>
+             </div>
+         </div>
+         <div class="collection-album-container column fourth">
+             <img src="assets/images/album_covers/01.png"/>
+             <div class="collection-album-info caption">
```

```
+                <p>
+                    <a class="album-name" href="#">The Colors</a>
+                    <br/>
+                    <a href="#">Pablo Picasso</a>
+                    <br/>
+                    X songs
+                    <br/>
+                </p>
+            </div>
+        </div>
+        <div class="collection-album-container column fourth">
+            <img src="assets/images/album_covers/01.png"/>
+            <div class="collection-album-info caption">
+                <p>
+                    <a class="album-name" href="#">The Colors</a>
+                    <br/>
+                    <a href="#">Pablo Picasso</a>
+                    <br/>
+                    X songs
+                    <br/>
+                </p>
+            </div>
+        </div>
+        <div class="collection-album-container column fourth">
+            <img src="assets/images/album_covers/01.png"/>
+            <div class="collection-album-info caption">
+                <p>
+                    <a class="album-name" href="#">The Colors</a>
+                    <br/>
+                    <a href="#">Pablo Picasso</a>
+                    <br/>
+                    X songs
+                    <br/>
+                </p>
+            </div>
+        </div>
+    </section>
  </body>
  ...
```

This may look like a lot of markup, but we've repeated the elements that contain the albums in our collection four times. We wrapped the album covers in a `.container` class to maintain an aesthetic content width. There is also a `.clearfix` class to prohibit floated content from sharing the line with these albums. We use the `.column` and `.fourth` classes to create nice, responsive containers for our albums that restrict them to 25% of their parent container.

We want to add one more class to the body of the collection page before we add styles to `collection.css`:

~/bloc/bloc-jams/collection.html

```
   ...
   </head>
-  <body>
+  <body class="collection">
      <nav class="navbar"> <!-- nav bar -->
   ...
```

Add Styles to `collection.css`

Add the following styles to the collection stylesheet:

~/bloc/bloc-jams/styles/collection.css

```
+ body.collection {
+     background-image: url(../assets/images/blurred_backgrounds/blur_bg_2.jpg);
+     background-repeat: no-repeat;
+     background-attachment: fixed;
+     background-position: center center;
+     background-size: cover;
+ }
+
+ .album-covers {
+     position: relative;
+ }
+
+ .collection-album-container {
+     position: relative;
+     margin-top: 30px;
+     margin-bottom: 20px;
+     text-align: center;
+ }
+
+ .collection-album-container .caption {
+     margin-top: 10px;
+ }
+
+ .collection-album-container .caption p {
+     font-size: 1rem;
+     font-weight: 300;
+     color: rgba(255, 255, 255, 0.6);
+ }
+
+ .collection-album-container .caption p a {
+     color: rgba(255, 255, 255, 0.6);
+ }
+
+ .collection-album-container .caption p a.album-name {
+     color: white;
+ }
+
+ .collection-album-container img {
+     width: 80%;
+ }
```

We've seen these properties before, but the background values are worth noting. We're using one of the blurred backgrounds for the background image in `collection.html`.

- The `background-repeat` property has a value of `no-repeat`, which means that if the dimensions of the collection page are greater than the size of the background image, the image will not repeat.

- Instead, `background-size` with a value of `cover` scales the image to cover the dimensions of the browser.
- Lastly, the `background-attachment` property with a value of `fixed` fixes the background so that when the page scrolls, the background image does not move up or down with the content.

The result is a background image that remains in place, scales in size with the browser, and displays without the breaks of a repeating image.

Recap

Concept	Description
Navigation	Navigation is an essential aspect of any website that allows users to quickly access other pages.
RGBa model	Colors can be defined in the red-green-blue-alpha model (RGBa) using the `rgba()` functional notation. RGBa extends the RGB color model to include the alpha channel, allowing specification of the opacity of a color.
CSS Properties	• `z-index` • `text-decoration` • `vertical-align`
Pseudo-classes	A pseudo-class, such as `:hover`, specifies a special state of the element to be selected.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

5. HTML & CSS: Collection View

Assignment | Discussion | Submission

6 HTML & CSS: Album View

Overview and Purpose

This checkpoint implements an "Album view" for the Bloc Jams application, which allows a user to view the details of a single album.

Objectives

After this checkpoint, students should be able to:

- Create HTML tables.
- Use media queries to make a page responsive on mobile devices.

Bloc Jams looks good so far. We'll build an Album view to expand the application's capabilities. From the collection page, a user can choose an individual album and view more details about it. The Album view will include a cover image, information about the album, and an HTML table of the album's songs.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Create the Album View Files

The Album view needs its own HTML page. Create the file in the base directory, the same

place where we created `index.html` and `collection.html`:

~/bloc/bloc-jams/

```
$ touch album.html
```

Create an accompanying CSS file:

~/bloc/bloc-jams/

```
$ touch styles/album.css
```

Add an HTML Skeleton to `album.html`

The HTML skeleton for the album page is very similar to the landing and collection pages, except that we link to `album.css` in the head and give the body an `album` class. Add the following to `album.html`:

~/bloc/bloc-jams/album.html

```
+ <!DOCTYPE html>
+ <html>
+     <head>
+         <title>Bloc Jams</title>
+         <meta name="viewport" content="width=device-width, initial-scale=1">
+         <link rel="stylesheet" type="text/css" href="http://fonts.googleapis.com/css?
+         <link rel="stylesheet" type="text/css" href="http://code.ionicframework.com/i
+         <link rel="stylesheet" type="text/css" href="styles/normalize.css">
+         <link rel="stylesheet" type="text/css" href="styles/main.css">
+         <link rel="stylesheet" type="text/css" href="styles/album.css">
+     </head>
+     <body class="album">
+         <nav class="navbar"> <!-- nav bar -->
+             <a href="index.html"  class="logo">
+                 <img src="assets/images/bloc_jams_logo.png" alt="bloc jams logo">
+             </a>
+             <div class="links-container">
+                 <a href="collection.html" class="navbar-link">collection</a>
+             </div>
+         </nav>
+     </body>
+ </html>
```

Add a Section for Album Details

We have a number of details to display on this page. We'll add a section for the album cover image, the album title, the artist, and release information.

Add the section markup to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
        </nav>
+
+       <main class="album-view container narrow">
+           <section class="clearfix">
+               <div class="column half">
+                   <img src="assets/images/album_covers/01.png" class="album-cover-a
+               </div>
+               <div class="album-view-details column half">
+                   <h2 class="album-view-title">The Colors</h2>
+                   <h3 class="album-view-artist">Pablo Picasso</h3>
+                   <h5 class="album-view-release-info">1909 Spanish Records</h5>
+               </div>
+           </section>
+       </main>
    </body>
  </html>
```

We've wrapped all of the content in a `<main>` tag because it is the main content. We've also wrapped this section in a `<section>` tag to separate it from other information we'll add later.

We've placed the album image in a `<div>` and the album details in another `<div>`. We've given the class names `column` and `half` to each `<div>` such that they expand to half their parent's width, which allows us to display them side by side.

Add Styles to `album.css`

Let's style the album details that we've added. The styles we'll add should all be familiar by now, although we'll use a different background image than the one used on the collection page.

Add the following styles to `album.css`:

~/bloc/bloc-jams/styles/album.css

```
+ body.album {
+     background-image: url(../assets/images/blurred_backgrounds/blur_bg_3.jpg);
+     background-repeat: no-repeat;
+     background-attachment: fixed;
+     background-position: center center;
+     background-size: cover;
+ }
+
+ .album-cover-art {
+     position: relative;
+     left: 20%;
+     margin-top: 1rem;
+     width: 60%;
+ }
+
+ .album-view-details {
+     position: relative;
+     top: 1.5rem;
+     padding: 1rem;
+ }
+
+ .album-view-details .album-view-title {
+     font-weight: 300;
+     font-size: 2rem;
+ }
+
+ .album-view-details .album-view-artist {
+     font-weight: 300;
+     font-size: 1.5rem;
+ }
+
+ .album-view-details .album-view-release-info {
+     font-weight: 300;
+     font-size: 0.75rem;
+ }
```

In our HTML, we've used a new class named `narrow` on the album-view container that sets a `max-width` property to the element. We may want to use this style again – let's add it to `main.css`:

~/bloc/bloc-jams/styles/main.css

```
   ...
   .container {
       margin: 0 auto;
       max-width: 64rem;
   }

+  .container.narrow {
+      max-width: 56rem;
+  }

   /* Medium screens (640px) */
   ...
```

Add a Table for Song Listings

We'll add the song list to complete the album's information.

Add the song list `<table>` below the album details section in `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
          <main class="album-view container narrow">
              <section class="clearfix">
                  ...
              </section>
+             <table class="album-view-song-list">
+                 <tr class="album-view-song-item">
+                     <td class="song-item-number">1</td>
+                     <td class="song-item-title">Blue</td>
+                     <td class="song-item-duration">X:XX</td>
+                 </tr>
+                 <tr class="album-view-song-item">
+                     <td class="song-item-number">2</td>
+                     <td class="song-item-title">Red</td>
+                     <td class="song-item-duration">X:XX</td>
+                 </tr>
+                 <tr class="album-view-song-item">
+                     <td class="song-item-number">3</td>
+                     <td class="song-item-title">Green</td>
+                     <td class="song-item-duration">X:XX</td>
+                 </tr>
+                 <tr class="album-view-song-item">
+                     <td class="song-item-number">4</td>
+                     <td class="song-item-title">Purple</td>
+                     <td class="song-item-duration">X:XX</td>
+                 </tr>
+                 <tr class="album-view-song-item">
+                     <td class="song-item-number">5</td>
+                     <td class="song-item-title">Black</td>
+                     <td class="song-item-duration">X:XX</td>
+                 </tr>
+             </table>
          </main>
  ...
```

Structuring lists of data is a standard use for tables. A table begins and ends with `<table>` tags. Each `<tr>` tag is a row of the table, and each `<td>` tag is an item within that row.

We've included a table row for each song. Additionally, the `<td>`s in each row are the same except for the song content. We'll eventually fill in our album and song list with real data, but for now we'll use placeholder album data.

Style the Song Listings

We'll style the song list table to fit in with the sleek look of our application.

Add the following styles to `album.css`:

~/bloc/bloc-jams/styles/album.css

```
  ...
  .album-view-details .album-view-release-info {
      font-weight: 300;
      font-size: 0.75rem;
  }

+ .album-view-song-list {
+     width: 90%;
+     margin: 1.5rem auto;
+     font-weight: 300;
+     font-size: 0.75em;
+     border-collapse: collapse;
+ }
+
+ .album-view-song-item {
+     height: 3rem;
      /* #1 */
+     border-bottom: 1px solid rgba(255,255,255,0.5);
+ }
+
+ .song-item-number {
+     width: 5%;
+ }
+
+ .song-item-title {
+     width: 85%;
+ }
+
+ .song-item-duration {
+     width: 5%;
+ }
```

At **#1**, we use `border-bottom` instead of `border` to specify one that only exists on the bottom of the element. Recall from the **Box Model resource** that borders have three values: `border: <width> <style> <color>;`. We've used an `rgba` color value with 50% opacity.

Make the Album View Responsive

Add the following at the bottom of `album.css`:

~/bloc/bloc-jams/styles/album.css

```
  ...
  .song-item-duration {
      width: 5%;
  }

  /* #2 */
+ @media (max-width: 640px) and (min-width: 320px) {
+     .album-view-details {
+         text-align: center;
+     }
+
+     .album-view-title {
+         margin-top: 0;
+     }
+ }
+
  /* #3 */
+ @media (max-width: 1024px) and (min-width: 320px) {
+    .album-view-song-list {
+         position: relative;
+         top: 1rem;
+         width: 80%;
+         margin: auto;
+     }
+ }
```

At **#2**, we center the album details text to look better on smaller devices. At **#3**, we adjust the song list to have spacing around it so that it doesn't meet the edge of small devices.

Update Links in Collection View

All the links on the Collection page point to `#`, which is a placeholder URL. Update the URLs of each `<a>` tag to link the album and artist names to the new Album view. For example:

~/bloc/bloc-jams/collection.html

```
   ...
-  <a class="album-name" href="#">The Colors</a>
+  <a class="album-name" href="album.html">The Colors</a>
   <br/>
-  <a href="#">Pablo Picasso</a>
+  <a href="album.html">Pablo Picasso</a>
   ...
```

Recap

Concept	Description
Tables	Use tables to present data in rows and columns.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

7 Programming Reinforcement: Introduction

Overview and Purpose

This checkpoint introduces the Codewars platform and three coding challenges to practice conditionals, basic data structures, string and array manipulation, and algorithms in JavaScript.

Objectives

After this checkpoint, you should be able to:

- Implement algorithms in JavaScript.
- Understand conditional logic and the importance of sequence.
- Understand the differences and similarities between strings and arrays in JavaScript.
- Discuss the advantages of various conditional statements in JavaScript.
- Understand how to read test specifications in JavaScript.

Introduction to Programming Reinforcement

Each foundation checkpoint introduces new concepts, patterns, and assignments. We designed Bloc's curriculum to push you to your learning limits. With that in mind, we've included breaks between lessons to reinforce programming concepts; this is the first of those breaks, known as Programming Reinforcement checkpoints.

In these checkpoints, you will complete coding challenges that help you master JavaScript and learn to think like a programmer. After you complete the challenges, you will meet with your mentor to discuss your solutions.

- Codewars
- Challenges
- Bonus Challenge: Musical Pitch Classes
- Assignment
  - For Mentors

Codewars

Sign up for a free account at **Codewars**. Codewars is a community of developers who help each other complete coding challenges. These challenges require you to apply JavaScript's core features and your strong, analytical thinking to solve them properly.

If this is your first time creating an account, you must solve two initial coding challenges to

complete sign up: **Multiply** and **Broken Greetings**.

> If you attempt Broken Greetings before completing the *this, apply(), call(), bind()* checkpoint, refer to **this solution**. The solution requires the use and understanding of `this`, which we cover in a later checkpoint.

Challenges

You must solve these three Kata (challenges) before submitting the checkpoint. At this point in your program, your skill level meets or exceeds that required by each Kata. While they may challenge you, know that you are capable of completing each one.

> Your mentor may help you, but we strongly encourage you attempt these on your own.

Kata	Hint
Powers of 2	Write a function which will return list of all powers of 2 from 0 to n.
Find Count of Most Frequent Item in Array	Write a function to find count of the most frequent item of an array.
Create Phone Number	Write a function that accepts an array of 10 integers (between 0 and 9), that returns a string of those numbers in the form of a phone number.

Bonus Challenge: Musical Pitch Classes

> This bonus challenge is optional, but we strongly encourage you to attempt it.

In this challenge, you must convert musical notes (C through B) into their numerical representations (0 to 11). The exercise gives you little to begin with:

```
function pitchClass(note){
  return null;
}
```

`note` is a string that represents the note itself and an optional modifier: sharp ('#') or flat ('b'). You must return an integer that represents the natural note (C, D, E, F, G, A, or B) optionally augmented by sharp or flat. Sharp notes are one value **larger** than their natural counterparts, whereas flat notes are one **smaller**. Here are some examples:

```
pitchClass('D') // D
// returns 2
pitchClass('D#') // D Sharp
// returns 3
pitchClass('Db') // D Flat
// returns 1
```

If the note goes below 0 or above 11, its value wraps to the other end of the scale (-1 becomes 11, 12 becomes 0). Some examples:

```
pitchClass('B') // B
// returns 11
pitchClass('B#') // B Sharp
// returns 0
pitchClass('C') // C
// returns 0
pitchClass('Cb') // C Flat
// returns 11
```

This exercise also tests your ability to verify input parameters. Sometimes the data passed to your method is inaccurate. Perform checks to avoid working with data that may cause errors in your code.

For example, does passing `null` to `pitchClass` make sense? How about `56`? Or `'May I have a cupcake?'`?

How would you rate this checkpoint and assignment? ☺ ☹

8 DOM Scripting: Animation

Overview and Purpose

This checkpoint introduces CSS transitions and DOM (Document Object Model) scripting, the most basic form of writing JavaScript to interact with the browser.

Objectives

After this checkpoint, students should be able to:

- Apply CSS transitions to HTML elements.
- Create and save a JavaScript file.
- Include JavaScript directly in an HTML document as well as link to a local or external script source.
- Discuss what the DOM is and what nodes are.
- Understand and explain the purpose of various DOM selectors – such as `getElementById()` and `getElementsByClassName()`.

We've established a structure and style for Bloc Jams, although at the moment the application is entirely static. We'll implement animations on our landing page to spice things up.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Add CSS Transitions

CSS transitions animate between properties to change an element's appearance. Many CSS properties are *animatable*; here's a **full list of animatable properties**.

We'll use CSS transitions to animate the selling points on the landing page.

Add the following to `.point` in `landing.css`:

~/bloc/bloc-jams/styles/landing.css

```
  ...
  .point {
      position: relative;
      padding: 2rem;
      text-align: center;
+     opacity: 0;
+     -webkit-transform: scaleX(0.9) translateY(3rem);
+     -moz-transform: scaleX(0.9) translateY(3rem);
+     transform: scaleX(0.9) translateY(3rem);
+     -webkit-transition: all 0.25s ease-in-out;
+     -moz-transition: all 0.25s ease-in-out;
+     transition: all 0.25s ease-in-out;
+     -webkit-transition-delay: 0.2s;
+     -moz-transition-delay: 0.2s;
+     transition-delay: 0.2s;
  }
  ...
```

The vendor prefixes (`-webkit` and `-moz`) that precede the `transform`, `transition`, and `transition-delay` properties ensure that our animations are compatible across browsers.

The transitions we've added will animate the `point` class elements by scaling and fading them.

First, we set the `opacity` property to `0`, which means that on page load this element will not be visible. We'll write a JavaScript program to trigger the animation and make the elements visible.

The `transform` property **can have many values**. We're using `scaleX(x)` to scale the element to 90% of its width. We're also using `translateY(y)` to to move the element `3rem` down from its normal position.

The `transition` property can have four unique values, of which we're using three: `<transition-property> <transition-duration> <transition-timing-function>`. The first value specifies the name of the CSS property the effect is for; in this case, we want to transition *all* CSS properties of the element. We want the transition to last a quarter of a second, and we want the transition timing function to `ease-in-out`.

Lastly, we use `transition-delay` property to delay the transition by 0.2 seconds.

Connect JavaScript to an HTML File

To trigger the animations on the landing page, we'll write a JavaScript program to activate our CSS transitions and then we'll connect the program to our application.

`<script>` tags allow us to define and run JavaScript on a web page. Open `index.html` and add the following code at the bottom of the file, just before the closing `</body>` tag:

~/bloc/bloc-jams/index.html

```
  ...
+         <script>
+             // our JavaScript will go here
+         </script>
      </body>
  </html>
```

Browsers read HTML files from the top down. Although scripts can be placed anywhere in a file, we've placed the `<script>` tags at the end of the document. Placing scripts near the top of the file may result in slower page loads.

Write a simple alert message to test the script:

~/bloc/bloc-jams/index.html

```
  ...
          <script>
-             // our JavaScript will go here
+             alert("Why hello there! I'm a wee bit of JavaScript.");
          </script>
      </body>
  </html>
```

Use your editor's live preview to see the alert message when the page loads.

HTML Files Are Documents

HTML files are documents that are intended for viewing in a browser. When a browser loads a web page, it uses the **Document Object Model** (DOM) to interpret each of the document's elements as an object (**node**). The DOM builds a tree of objects that is accessible to JavaScript.

JavaScript uses the `document` object to refer to a web page. **Selectors** are methods that exist on the `document` object. We can use selectors to perform actions on elements in a

web page because the DOM interprets those elements as objects. Once an element is selected, certain properties, modeling styles, classes, and other pieces of information are accessible.

Some selectors, such as `getElementsByClassName()`, return a list of element nodes (called a `NodeList`). We can use index notation to reference a node within a `NodeList`. For example, we can select all the `point` class elements in our landing page and reference each individually:

```
var points = document.getElementsByClassName('point');

var firstPoint = points[0];
var secondPoint = points[1];
// etc.
```

Other selectors, such as `querySelector()` and `getElementById()`, return a single node instead of an array. For example, if there was an element that had an ID name of `warning`, we might select it like this:

```
var warningElement = document.getElementById('warning');
```

Animate Elements

We'll use the `getElementsByClassName()` selector to select all the `point` class elements in our landing page. Add the following code to `index.html`:

~/bloc/bloc-jams/index.html

```
  ...
          <script>
-             alert("Why hello there! I'm a wee bit of JavaScript.");
+             var animatePoints = function() {
+
+                 var points = document.getElementsByClassName('point');
+
+                 var revealFirstPoint = function() {
+                     points[0].style.opacity = 1;
+                     points[0].style.transform = "scaleX(1) translateY(0)";
+                     points[0].style.msTransform = "scaleX(1) translateY(0)";
+                     points[0].style.WebkitTransform = "scaleX(1) translateY(0)";
+                 };
+
+                 var revealSecondPoint = function() {
+                     points[1].style.opacity = 1;
+                     points[1].style.transform = "scaleX(1) translateY(0)";
+                     points[1].style.msTransform = "scaleX(1) translateY(0)";
+                     points[1].style.WebkitTransform = "scaleX(1) translateY(0)";
+                 };
+
+                 var revealThirdPoint = function() {
+                     points[2].style.opacity = 1;
+                     points[2].style.transform = "scaleX(1) translateY(0)";
+                     points[2].style.msTransform = "scaleX(1) translateY(0)";
+                     points[2].style.WebkitTransform = "scaleX(1) translateY(0)";
+                 };
+
+                 revealFirstPoint();
+                 revealSecondPoint();
+                 revealThirdPoint();
+
+             };
          </script>
  ...
```

We've written a function named `animatePoints`. When called, this function will activate the CSS transitions on the landing page. It will update the styles from what we've set in place in the CSS file to the new styles in the script. The `opacity` will change from `0` to `1`. The `transform` property will scale the element from 90% to 100% of its width and translate it `3rem` up to its normal position.

When our script runs, it assigns an array-like list of all elements that have the class name `point` to the `points` variable. Our program alters the `style` of each `.point` in the `NodeList`. The `style` property is an object that represents every style applied to the element. We can call `document.getElementsByClassName('point').style` to recover every

style applied to `.point`.

We've written our script, but it won't execute until we call it. Add the following line to call `animatePoints()`:

~/bloc/bloc-jams/index.html

```
  ...
          <script>
              var animatePoints = function() {
                  ...
              };

+             animatePoints();
          </script>
  ...
```

Use your editor's live preview to see the animation when the page loads.

We can also call the script directly from the console. Remove the `animatePoints()` function call that we just added:

~/bloc/bloc-jams/index.html

```
  ...
          <script>
              var animatePoints = function() {
                  ...
              };

-             animatePoints();
          </script>
  ...
```

Use your editor's live preview to view the page. Right click anywhere on the page and choose "Inspect Element" from the dropdown menu. In the DevTools pane, select the "Console" tab. Type in the `animatePoints()` function call directly to the console:

Console

```
animatePoints();
```

Hit the Enter/Return key on your keyboard and view the animation.

Create the JavaScript File and Link It to the

Landing Page

In addition to writing scripts directly in an HTML file, we can run scripts by defining them in a separate file and linking them to a document. We'll add our script to animate the landing page to a separate JavaScript file. In the base directory, create a new directory named `scripts`. In the `scripts` directory, create a file named `landing.js`:

~/bloc/bloc-jams/

```
$ mkdir scripts
$ cd scripts
$ touch landing.js
```

Cut the script from `index.html` and paste it into `landing.js`:

~/bloc/bloc-jams/scripts/landing.js

```
+ var animatePoints = function() {
+
+     var points = document.getElementsByClassName('point');
+
+     var revealFirstPoint = function() {
+         points[0].style.opacity = 1;
+         points[0].style.transform = "scaleX(1) translateY(0)";
+         points[0].style.msTransform = "scaleX(1) translateY(0)";
+         points[0].style.WebkitTransform = "scaleX(1) translateY(0)";
+     };
+
+     var revealSecondPoint = function() {
+         points[1].style.opacity = 1;
+         points[1].style.transform = "scaleX(1) translateY(0)";
+         points[1].style.msTransform = "scaleX(1) translateY(0)";
+         points[1].style.WebkitTransform = "scaleX(1) translateY(0)";
+     };
+
+     var revealThirdPoint = function() {
+         points[2].style.opacity = 1;
+         points[2].style.transform = "scaleX(1) translateY(0)";
+         points[2].style.msTransform = "scaleX(1) translateY(0)";
+         points[2].style.WebkitTransform = "scaleX(1) translateY(0)";
+     };
+
+     revealFirstPoint();
+     revealSecondPoint();
+     revealThirdPoint();
+
+ };
```

In `index.html`, replace the script with a link to our new external JavaScript file:

~/bloc/bloc-jams/index.html

```
  ...
          </section>
+         <script src="scripts/landing.js"></script>
      </body>
  </html>
```

Although the script is now external, we can still call it from the browser's console. Use DevTools to inspect the page again and call `animatePoints()` in the console to confirm.

Recap

Concept	Description
CSS Transitions	CSS transitions provide a way to control transition type, which properties to animate, as well as animation speed, length, and delay. • `transition` **property** • `transition-timing-function` **property**
`<script>`	The `<script>` tag defines a client-side script. We can use it to define statements within an HTML file or to point to an external script using the `src` attribute. While scripts **can be placed anywhere**, we often place them at **the end of the document**.
The DOM	When a browser loads a web page, it uses the Document Object Model (DOM) to interpret each of the document's elements as an object. The DOM builds a tree of objects that is accessible to JavaScript.
Selectors	Selectors are methods that exist on the document object. We use selectors to perform actions on elements in a web page because the DOM interprets those elements as objects.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

Finding Focus: Location

Overview and Purpose

Job markets differ from region to region. Job-seeking students that minimize the number of locations they apply to jobs to will have more success landing their first offer.

Objectives

- Understand the importance of focusing on fewer locations to increase job search success.
- Isolate the locations in which you will conduct your job search.

Common Frustrations

Students become frustrated during the job hunt for many reasons. Many students aren't sure how to spend their time, they become demoralized when they start to feel like they're moving away from a job instead of towards one, and ultimately get frustrated when they can't find a job.

The first common frustration that students have is *not knowing how to spend their time*. Students know that they're supposed to keep learning after Bloc, but what should they focus on? Interview questions? New technologies? Maybe a side-project? They know they should be networking, but how? Who should they network with? What should they talk about? They know they should be applying to jobs, but applying to every open position they see is daunting.

Another common frustration is that *many students move backwards during a job search*. Instead of continuing to grow and learn as developers, students forget the fundamentals and haphazardly learn trendy technologies that don't help their resume. Instead of tactfully building a network of relationships, students weaken their existing network by asking everyone they know for a job. Instead of feeling confident after completing a difficult program, they begin to feel like they're not worthy of an amazing job.

The most unfortunate frustration is the actual *difficulty in finding a job*. Job seeking students will constantly hear that they need more experience – if their applications even make it to the interview stage. Students who try to network will often find that no one is hiring. We've seen amazing students apply to 175 jobs and only hear back from 5 of them – all rejections.

In the end, graduates know they're supposed to be improving their skills, networking, and applying for jobs, but they don't have the time to do it all. After weeks or months, they lose their skills, their focus, and their hope.

Finding Focus

Having a clear focus is a critical factor in finding success in your job hunt. There are three variables that will greatly impact how you conduct your job search, the location(s) where you are willing to work, the technologies you are learning, and what industries you prioritize your search around. We'll be covering the first variable in this checkpoint:

- **Location**
- Technologies
- Industries

Your First Job ≠ Your Dream Job

Your first developer job may not be your dream developer job, but this doesn't mean you won't love it. The very first developer position you land is going to be a huge learning experience for you. This first job, along with the next few that follow, will get you closer to your dream job. This is why you need to focus on your short-term goal. Once you land that first job, you'll have your foot in the door. That's the most difficult part.

Choosing Your Location

To streamline and ensure success in your job search, you'll need to minimize the number of locations in which you conduct your job search. Why is this a necessary step? **Job markets differ from region to region**. Honing in on one location will help you understand the job market that you're aiming to find a job in: the companies, prevalent technologies, and professionals located in this region. Equipped with this base knowledge, you will have the ability to structure your learning in a way that will make you the best candidate for the location in which you'd like to find a job.

How would you rate this checkpoint and assignment? ☺ ☹

9 DOM Scripting: Events

Overview and Purpose

This checkpoint introduces DOM events, which allow user interactions in the browser to trigger method calls.

Objectives

After this checkpoint, students should be able to:

- Understand and explain the concept of events.
- Discuss the purpose of event listeners and handlers.
- Recognize and use some basic DOM scripting methods and properties.

We know how to write and invoke functions that execute JavaScript animations on our landing page. However, we can't expect Bloc Jams users to open the Developer Console and manually call our functions. Ideally, our application should know *when* to run animations. We will implement that behavior with DOM **events**. Events are messages which the browser broadcasts to the DOM. We can use event **listeners** for these broadcasts and execute code as they arrive.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Add an Event That Detects the Page Load

There is a property on the `window` object called `onload` that we can pass a function that executes when the window (like a browser window or tab) finishes loading. Add the `window.onload` property to `landing.js` and pass it an anonymous function that calls a JavaScript `alert()`:

~/bloc/bloc-jams/scripts/landing.js

```
   var animatePoints = function() {
       ...
   };

+  window.onload = function() {
+      alert("The window has loaded!");
+  }
```

We assign the `window.onload` property an **event handler**, a function that *handles* code in response to an event. The event handler executes as soon as an action **fires** an event.

As a page loads, the browser may render elements, styles, and scripts that require additional time to process. This means that some DOM nodes may not exist until the browser finishes loading the page. Put any code that is dependent on a completely-loaded web page in a `window.onload` block, particularly any code that depends on DOM elements to execute properly.

Attach Behavior to the Scroll Event

A common way to initiate animation is to wait for the user to scroll to the elements we wish to animate. When we want to execute code based on an event, we attach a listener that executes a handler. Delete the alert, and call `addEventListener()` to add a listener to the `window` object:

~/bloc/bloc-jams/scripts/landing.js

```
   ...
   window.onload = function() {
-      alert("The window has loaded!");
+      window.addEventListener('scroll', function(event) {
+          console.log(event);
+      });
   }
```

The `addEventListener()` method takes three arguments, but we'll focus on the first two because the third, `useCapture`, is optional and does not affect the current listener:

Order	Parameter	Type	Required?	Description
1	*type*	string	yes	Represents the type of event for which the DOM should be listening.
2	*listener*	function	yes	Passed in as an event handler and contains the code that executes when the event fires.
3	*useCapture*	boolean	no	Specifies whether the user wishes to initiate capture.

The first argument is a string that represents the type of event for which the DOM should be listening. The second is a **callback function** passed in as an event handler, which contains the code that executes when the event fires.

> We'll talk more about callbacks and their role in JavaScript in a resource at the end of this section.

Event listeners provide the first argument to the handler function, `event`, sometimes abbreviated to `e`. `event` is a DOM object whose properties provide event dispatch information. We've added a `console.log()` function to the event listener above so we can inspect the `scroll` event's properties. Open the browser console and scroll the page to see the event object.

Expanding the event object shows its properties.

> We can listen to all events, including `scroll`, on any DOM element. However, we wanted to listen specifically for when the user scrolled the entire page, so we attached it to the `window` object.

Move Selling Point Selector to the Global Scope

The page scrolling event listener needs access to the selling point DOM elements. We could include the selector both inside and outside of the `animatePoints()` function, but then we're repeating ourselves; remember the importance of DRY, or Don't Repeat Yourself.

Instead, move the selector into a new variable that is accessible from the entire file, and add a parameter to `animatePoints()` so that we can pass the elements into the function when we call it:

~/bloc/bloc-jams/scripts/landing.js

```
+ var pointsArray = document.getElementsByClassName('point');
+
+ var animatePoints = function(points) {
- var animatePoints = function() {
-     var points = document.getElementsByClassName('point');
      var revealPoint = function(index) {
          points[index].style.opacity = 1;
          points[index].style.transform = "scaleX(1) translateY(0)";
  ...
```

Measure the Scroll Distance from the Top of the Window

We want `animatePoints()` to run when the client scrolls the selling points into view. The most straightforward way to do this is to determine the distance from the top of the browser window to the selling points. We will trigger the animation after the user scrolls far enough to render the selling points clearly visible.

Calling `getBoundingClientRect()` returns an object with four properties, `top`, `left`, `right`, and `bottom`. Each property measures the distance (in pixels) from the outside of a selected element to the end of the viewport (in this case, the window).

Use `console.log()` to print the `top` property of the `.selling-points` element whenever the user scrolls:

~/bloc/bloc-jams/scripts/landing.js

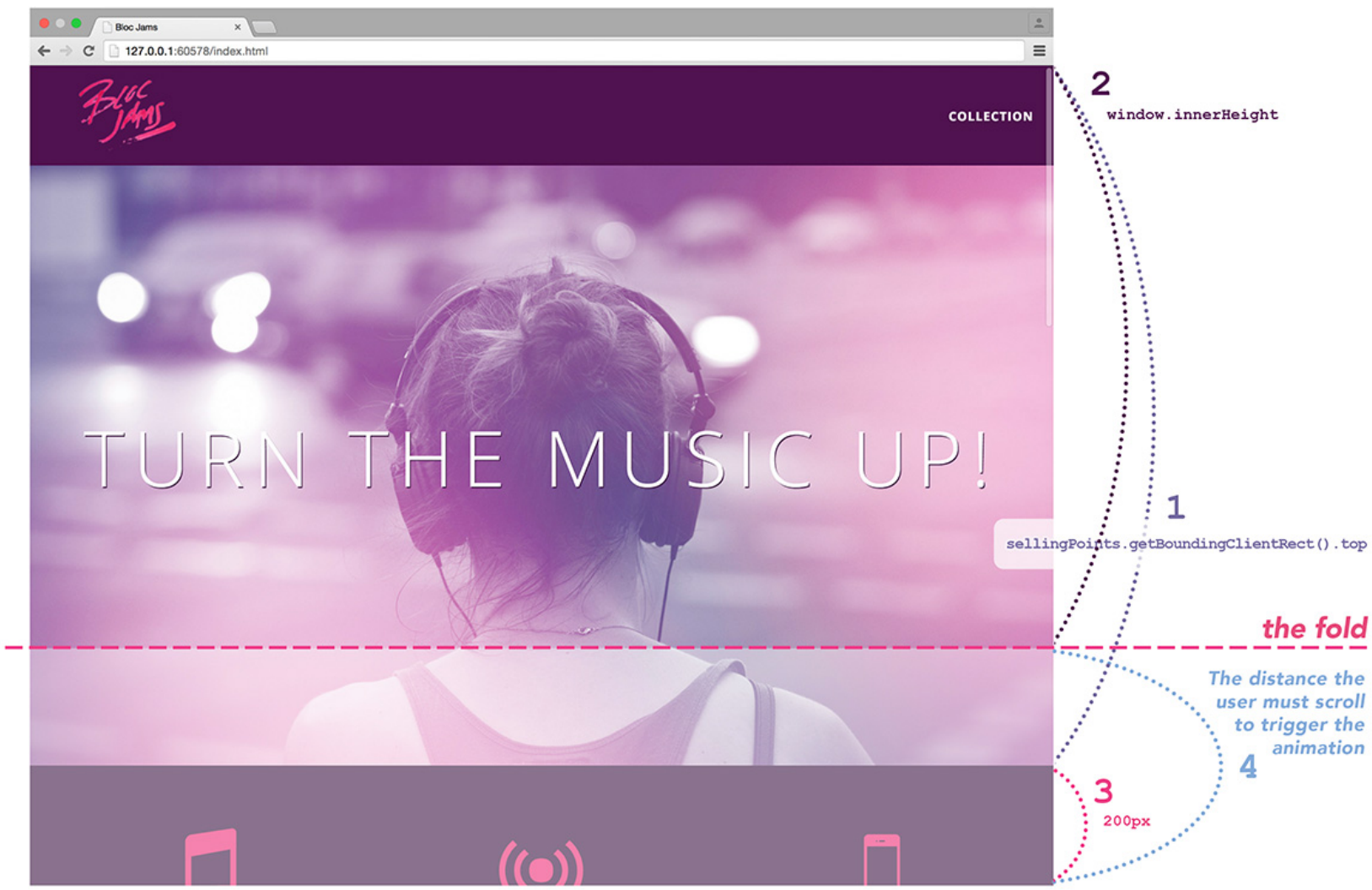
```
  ...
  window.onload = function() {
+     var sellingPoints = document.getElementsByClassName('selling-points')[0];

      window.addEventListener("scroll", function(event) {
-         console.log(event);
+         console.log("Current offset from the top is " + sellingPoints.getBoundingClie
      });
  }
```

Open up the Developer Console and scroll to see the logged distance:

Add `animatePoints()` to the Event Handler

We need to call `animatePoints()` when the selling points are visible in the view. Depending on the browser's zoom level, the selling points won't always be the same number of pixels away from the top of the document. In this case, we need to calculate the difference between (1) the window height and (2) the distance of `.selling-points` from the top of the document, also known as the **offset**. The difference will be the distance the user must scroll to reach the top of `.selling-points`.

This diagram displays the distances we need to measure to figure out the number of pixels the user must scroll to trigger the animation:



In order to see the animation, however, we must scroll a bit more so that `.selling-points` is visible above the fold. Let's trigger the animation when a user scrolls at least 200 pixels into the `.selling-points` element. We'll nest the animation call in an `if` statement:

~/bloc/bloc-jams/scripts/landing.js

```
   ...
   window.onload = function() {
       var sellingPoints = document.getElementsByClassName('selling-points')[0];
+      var scrollDistance = sellingPoints.getBoundingClientRect().top - window.innerHeig

       window.addEventListener("scroll", function(event) {
-          console.log("Current offset from the top is " + sellingPoints.getBoundingClie
+          if (document.documentElement.scrollTop || document.body.scrollTop >= scrollDi
+              animatePoints(pointsArray);
+          }
       });
   }
```

Scroll the page and you should see the points animate as shown in the video at the beginning of this checkpoint.

Animate the Selling Points on Taller Screens

We've tied the animation to a scroll event, and it works for most situations. What happens if the user's screen is tall enough to display all the content on load and prevents them from scrolling?

They won't be able to see the animation when they load the page. To protect against this bug, we want to measure the height of the window when the page loads. If the window is too big, we animate the selling points immediately. The height of all the landing page content is about 950 pixels, so any window taller than that will warrant an immediate animation call. Use the `window`'s `innerHeight` property to detect the height of the browser when the page loads:

~/bloc/bloc-jams/scripts/landing.js

```
   ...
   window.onload = function() {
      // Automatically animate the points on a tall screen where scrolling can't trigge
+      if (window.innerHeight > 950) {
+          animatePoints(pointsArray);
+      }

       var sellingPoints = document.getElementsByClassName('selling-points')[0];
   ...
```

Event Context

Because event handlers are functions, they also have a context. Where they differ from traditional anonymous functions is that their context is set automatically by the *event dispatcher*. The event dispatcher is responsible for keeping track of which events are attached to DOM elements (`div` or `a`, for example), and *dispatching* the events at the appropriate time.

When an event handler is executed, its context changes to the DOM element whose event handler is firing. This can be different from the `event.target`, which is a reference to the object that caused the event to be dispatched. You can see an example of how the context of the event handler changes and can be different than the `event.target` **here**.

Recap

Concept	Description
Events	An event is a message that dispatches to the browser when a certain client action occurs. • `scroll`
Dispatching	If you're curious to know more on how events are dispatched, you can reference the **event flow documentation**.
Event Listener	An event listener is an object that receives a notification of the event when the event of a specified type (e.g. `scroll`) occurs.
Event Handler	An event handler is code that runs in response to an event. The handler is a **JavaScript callback**.
Callback	A callback is a function passed as an argument to another function.
`window` **Object**	Represents a window containing a DOM document.
`window.alert()`	Displays an alert dialog with the optional specified content and an OK button.
`getBoundingClientRect()`	Returns the size of an element and its position relative to the viewport.

`window.innerHeight` **Property**	Height (in pixels) of the browser window viewport.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

10 DOM Scripting: Collection View

Overview and Purpose

This checkpoint uses DOM scripting with the Collection view to generate templates dynamically.

Objectives

After this checkpoint, students should be able to:

- Create and link a JavaScript file.
- Translate a static HTML template into a dynamic JavaScript template.

The Collection view displays the available albums. Currently, we're using static placeholders to display albums, but realistically, a user may have hundreds of albums with different songs, cover art, and details. We'll create a script that dynamically generates as many albums as needed to display.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Create and Link the JavaScript File

The Collection view needs its own JavaScript file. Create the file in the `scripts` directory,

the same place where we created `landing.js`:

~/bloc/bloc-jams/scripts

```
$ touch collection.js
```

Connect the JavaScript file to `collection.html` by adding a `<script>` tag at the end of the document:

~/bloc/bloc-jams/collection.html

```
  ...
          </section>
+         <script src="scripts/collection.js"></script>
      </body>
  </html>
```

Remove Static Templates

We'll set up our JavaScript file to generate the album templates. We no longer require the static templates currently in place. Remove them from `collection.html`:

~/bloc/bloc-jams/collection.html

```
  ...
        <section class="album-covers container clearfix">
-           <div class="collection-album-container column fourth">
-               <img src="assets/images/album_covers/01.png"/>
-               <div class="collection-album-info caption">
-                   <p>
-                       <a class="album-name" href="album.html">The Colors</a>
-                       <br/>
-                       <a href="album.html">Pablo Picasso</a>
-                       <br/>
-                       X songs
-                       <br/>
-                   </p>
-               </div>
-           </div>
-           <div class="collection-album-container column fourth">
-               <img src="assets/images/album_covers/01.png"/>
-               <div class="collection-album-info caption">
-                   <p>
-                       <a class="album-name" href="album.html">The Colors</a>
-                       <br/>
```

```
-                        <a href="album.html">Pablo Picasso</a>
-                        <br/>
-                        X songs
-                        <br/>
-                    </p>
-                </div>
-            </div>
-            <div class="collection-album-container column fourth">
-                <img src="assets/images/album_covers/01.png"/>
-                <div class="collection-album-info caption">
-                    <p>
-                        <a class="album-name" href="album.html">The Colors</a>
-                        <br/>
-                        <a href="album.html">Pablo Picasso</a>
-                        <br/>
-                        X songs
-                        <br/>
-                    </p>
-                </div>
-            </div>
-            <div class="collection-album-container column fourth">
-                <img src="assets/images/album_covers/01.png"/>
-                <div class="collection-album-info caption">
-                    <p>
-                        <a class="album-name" href="album.html">The Colors</a>
-                        <br/>
-                        <a href="album.html">Pablo Picasso</a>
-                        <br/>
-                        X songs
-                        <br/>
-                    </p>
-                </div>
-            </div>
        </section>
  ...
```

Build JavaScript Templates with a String

We'll use a template to create each album in the Collection view. Add the following to `collection.js`:

~/bloc/bloc-jams/scripts/collection.js

```
+ var collectionItemTemplate =
+     '<div class="collection-album-container column fourth">'
+   + '  <img src="assets/images/album_covers/01.png"/>'
+   + '  <div class="collection-album-info caption">'
+   + '    <p>'
+   + '      <a class="album-name" href="album.html"> The Colors </a>'
+   + '      <br/>'
+   + '      <a href="album.html"> Pablo Picasso </a>'
+   + '      <br/>'
+   + '      X songs'
+   + '      <br/>'
+   + '    </p>'
+   + '  </div>'
+   + '</div>'
+   ;
```

We create a variable named `collectionItemTemplate` to hold the template. When the script runs, the browser *caches* content generated by the script, such as this template. Cached elements speed up page loading times.

This is our first instance of a multiline-JavaScript string. We've ported over the static template from `collection.html` to our JavaScript file. To make the template a string, it must be wrapped in quotation marks. The template could exist on one line, but for readability we return each line as we normally would in an HTML document. To keep the string together, we use `+` at the start of each line, where it's easy to see its location.

We've used `+` many times before in strings, especially to include variable values. Consider these three strings:

```
var milton1 = 'I believe you have my stapler.';

var milton2 = 'I believe' + ' you ' + 'have my stapler.';

var milton3 = 'I believe'
    + ' you '
    + 'have my stapler.';
```

These three strings produce the same results.

Display the Albums

Now that we've assigned the template to a variable, we'll use that variable to add as many albums as we'd like to the collection page. Add the following to `collection.js`, after the template:

~/bloc/bloc-jams/scripts/collection.js

```
  ...
+ window.onload = function() {
      // #1
+     var collectionContainer = document.getElementsByClassName('album-covers')[0];
      // #2
+     collectionContainer.innerHTML = '';
+
      // #3
+     for (var i = 0; i < 12; i++) {
+         collectionContainer.innerHTML += collectionItemTemplate;
+     }
+ }
```

We need our JavaScript to dynamically generate the album template in place of what was statically there. At **#1**, we select the first (and only, as we've designed it) element with an `album-covers` class name. We assign this specified element to a variable named `collectionContainer`.

At **#2**, We assign an empty string to `collectionContainer`'s `innerHTML` property to clear its content. This ensures we're working with a clean slate before we insert content with JavaScript.

We then create a `for` loop, at **#3**, that inserts 12 albums using the `+=` operator, which appends content to strings. Each loop adds the contents of `collectionItemTemplate` (the template) to the `innerHTML` of `collectionContainer`, thereby generating the albums that display on the collection page.

View Bloc Jams in the browser to see our script in action.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

11 DOM Scripting: Album View

Overview and Purpose

This checkpoint uses DOM scripting with the Album view to generate templates dynamically.

Objectives

After this checkpoint, students should be able to:

- Manipulate HTML elements using DOM selectors.
- Inject stored information in JavaScript into a template using DOM scripting.

The Album view displays an album's details and song list. Currently, we're using static content to display this information. An application such as Bloc Jams will pull the album information from a database and populate the view dynamically. We'll create a script to pull information from album objects and display it in the Album view.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Create and Link the JavaScript File

The Album view needs its own JavaScript file. Create the file in the `scripts` directory, the

same place we created `landing.js` and `collection.js`:

~/bloc/bloc-jams/scripts

```
$ touch album.js
```

Connect the JavaScript file to `album.html` by adding a `<script>` tag at the end of the document:

~/bloc/bloc-jams/album.html

```
  ...
          </main>
+         <script src="scripts/album.js"></script>
      </body>
  </html>
```

Remove Static Elements

We'll set up our JavaScript file to generate the song list information. We no longer require the static elements that we currently have in place. Remove the static elements from `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
            <table class="album-view-song-list">
-               <tr class="album-view-song-item">
-                   <td class="song-item-number">1</td>
-                   <td class="song-item-title">Blue</td>
-                   <td class="song-item-duration">X:XX</td>
-               </tr>
-               <tr class="album-view-song-item">
-                   <td class="song-item-number">2</td>
-                   <td class="song-item-title">Red</td>
-                   <td class="song-item-duration">X:XX</td>
-               </tr>
-               <tr class="album-view-song-item">
-                   <td class="song-item-number">3</td>
-                   <td class="song-item-title">Green</td>
-                   <td class="song-item-duration">X:XX</td>
-               </tr>
-               <tr class="album-view-song-item">
-                   <td class="song-item-number">4</td>
-                   <td class="song-item-title">Purple</td>
-                   <td class="song-item-duration">X:XX</td>
-               </tr>
-               <tr class="album-view-song-item">
-                   <td class="song-item-number">5</td>
-                   <td class="song-item-title">Black</td>
-                   <td class="song-item-duration">X:XX</td>
-               </tr>
            </table>
  ...
```

Create Album Objects

We will create JavaScript objects to represent albums. These objects will store information such as album title, artist, label, songs, etc. Add the following to `album.js`:

~/bloc/bloc-jams/scripts/album.js

```
   // Example Album
+  var albumPicasso = {
+      title: 'The Colors',
+      artist: 'Pablo Picasso',
+      label: 'Cubism',
+      year: '1881',
+      albumArtUrl: 'assets/images/album_covers/01.png',
+      songs: [
+          { title: 'Blue', duration: '4:26' },
+          { title: 'Green', duration: '3:14' },
+          { title: 'Red', duration: '5:01' },
+          { title: 'Pink', duration: '3:21'},
+          { title: 'Magenta', duration: '2:15'}
+      ]
+  };
+
   // Another Example Album
+  var albumMarconi = {
+      title: 'The Telephone',
+      artist: 'Guglielmo Marconi',
+      label: 'EM',
+      year: '1909',
+      albumArtUrl: 'assets/images/album_covers/20.png',
+      songs: [
+          { title: 'Hello, Operator?', duration: '1:01' },
+          { title: 'Ring, ring, ring', duration: '5:01' },
+          { title: 'Fits in your pocket', duration: '3:21'},
+          { title: 'Can you hear me now?', duration: '3:14' },
+          { title: 'Wrong phone number', duration: '2:15'}
+      ]
+  };
```

We've created two album objects to use in our application. In a real-world scenario, we would pull this information from a database, where we could store hundreds or thousands of albums and their corresponding details.

Dynamically Generate Song Row Content

Now we'll create a function named `createSongRow` that generates the song row content. Add the following to `album.js`, after the album objects:

~/bloc/bloc-jams/scripts/album.js

```
   ...
+  var createSongRow = function(songNumber, songName, songLength) {
+      var template =
+         '<tr class="album-view-song-item">'
+       + '  <td class="song-item-number">' + songNumber + '</td>'
+       + '  <td class="song-item-title">' + songName + '</td>'
+       + '  <td class="song-item-duration">' + songLength + '</td>'
+       + '</tr>'
+       ;
+
+      return template;
+  };
```

The `createSongRow` function assigns our previously static song row template to a variable named `template` and returns it. Instead of statically declaring the song number, name, or length, our function takes them as arguments and populates the song row template accordingly.

Set the Current Album

We'll create a function named `setCurrentAlbum` that the program calls when the window loads. It will take one of our album objects as an argument and will utilize the object's stored information by injecting it into the template. Add the following to the end of `album.js`:

~/bloc/bloc-jams/scripts/album.js

```
   ...
+  var setCurrentAlbum = function(album) {
       // #1
+      var albumTitle = document.getElementsByClassName('album-view-title')[0];
+      var albumArtist = document.getElementsByClassName('album-view-artist')[0];
+      var albumReleaseInfo = document.getElementsByClassName('album-view-release-info')
+      var albumImage = document.getElementsByClassName('album-cover-art')[0];
+      var albumSongList = document.getElementsByClassName('album-view-song-list')[0];
+
       // #2
+      albumTitle.firstChild.nodeValue = album.title;
+      albumArtist.firstChild.nodeValue = album.artist;
+      albumReleaseInfo.firstChild.nodeValue = album.year + ' ' + album.label;
+      albumImage.setAttribute('src', album.albumArtUrl);
+
       // #3
+      albumSongList.innerHTML = '';
+
       // #4
+      for (var i = 0; i < album.songs.length; i++) {
+          albumSongList.innerHTML += createSongRow(i + 1, album.songs[i].title, album.s
+      }
+  };
+
+  window.onload = function() {
+      setCurrentAlbum(albumPicasso);
+  };
```

First, at **#1**, we select all of the HTML elements required to display on the album page: title, artist, release info, image, and song list. We want to populate these elements with information. To do so, we assign the corresponding values of the album objects' properties to the HTML elements.

At **#2**, the `firstChild` property identifies the first child node of an element, and `nodeValue` returns or sets the value of a node. Alternatively, we could *technically* use `innerHTML` to insert plain text (like we did in `collection.js`), but it's excessive and semantically misleading in this context because we aren't adding any HTML.

For example, the `.albumTitle` element has only one node and it's a text node. When we use the `firstChild` property and `nodeValue` properties together on the `.albumTitle` element, we set the value of that text node to `album.title`.

When we populated the Collection view with albums, we initially set the value of the parent container's `innerHTML` to an empty string. This ensured that we were working with a clean slate. We do the same here, at **#3**, and clear the album song list HTML to make sure there are no interfering elements.

We use a `for` loop, at **#4**, to go through all the songs from the specified album object and insert them into the HTML using the `innerHTML` property. The `createSongRow` function is called at each loop, passing in the song number, name, and length arguments from our album object.

View Bloc Jams in the browser to see our script in action.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

12 DOM Scripting: Play/Pause Part 1

Overview and Purpose

This checkpoint introduces event delegation and bubbling.

Objectives

After this checkpoint, students should be able to:

- Discuss the benefits of event delegation.
- Explain the process of event bubbling.
- Explain the use of HTML data attributes.

This is the first of two checkpoints that implement working play and pause buttons on the Bloc Jams album page. We'll use events and event handlers to control whether a number or play button appears in the table cell.

Git

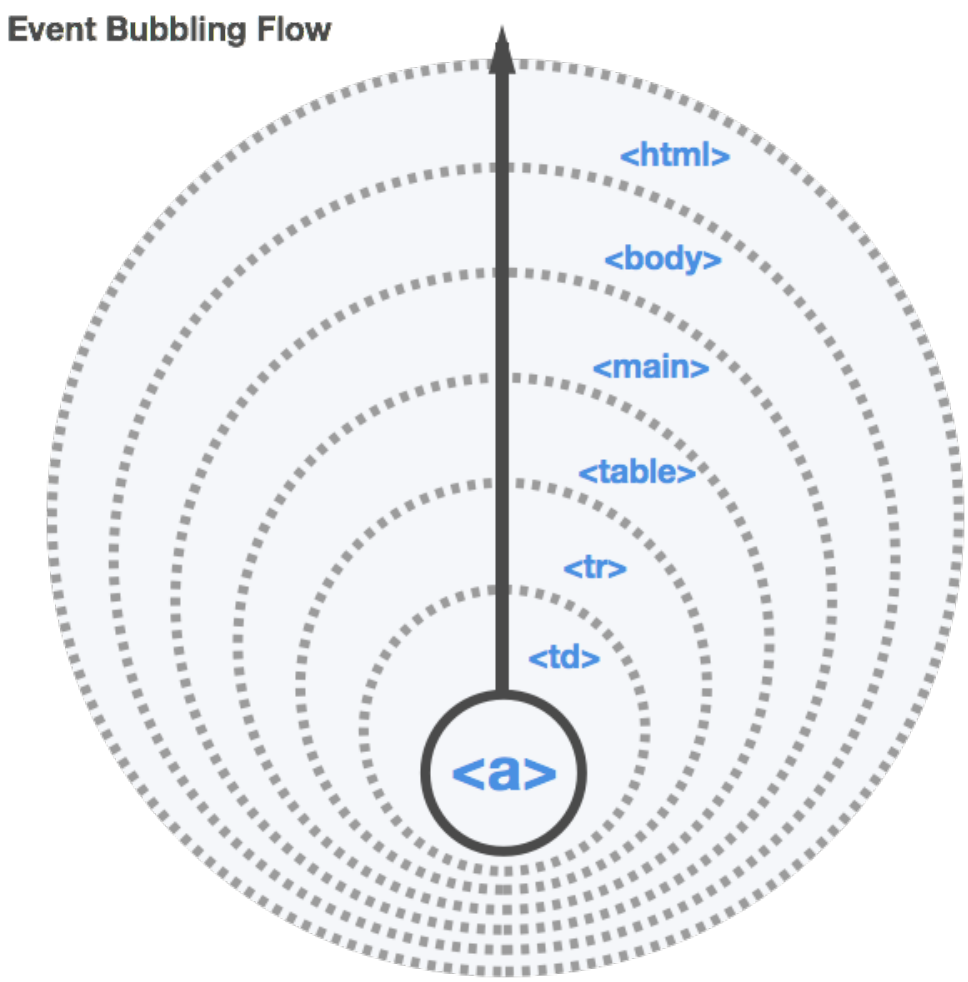
Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Use Event Delegation to Track the Mouse's Position

Adding many event listeners to elements in a document can make the page load and

execute JavaScript more slowly. To reduce the number of event listeners, we'll use **event delegation**. It allows us to listen for an event on a parent element but target the behavior on one of its children.

Listening for an event on a parent is possible because of **event bubbling**. An event may fire on a child element, but it propagates up (or *bubbles* up) the DOM tree to its parent.



The target parent element is the table with the class `.album-view-song-list`. Store the selected table in a variable and add a listener to it for the `mouseover` event in `album.js`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
+ var songListContainer = document.getElementsByClassName('album-view-song-list')[0];

  window.onload = function() {
      setCurrentAlbum(albumPicasso);

+     songListContainer.addEventListener('mouseover', function(event) {
          // #1
+         console.log(event.target);
+     });
  ...
```

The `target` property on the event object at **#1** stores the DOM element where the event occurred. Start your editor's live preview and open up the Developer Console. Mouse over

the table, and the element where the event is dispatched will be logged to the console.

The console output shows that moused-over elements will fire an event that eventually registers with the table's event listener.

Substitute the Play Button for the Song Number

To display the play button when we hover over the table row, we'll change the content of the table cell with the class `.song-item-number`. We'll use the **Ionicons** play button for a nice-looking icon.

Add a template for the play button to `album.js`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var songListContainer = document.getElementsByClassName('album-view-song-list')[0];

  // Album button templates
+ var playButtonTemplate = '<a class="album-song-button"><span class="ion-play"></span>

  window.onload = function() {
  ...
```

And add the CSS for the play button to `album.css`:

~/bloc/bloc-jams/styles/album.css

```
   ...
   .song-item-duration {
       width: 5%;
   }
+
+  .album-song-button {
+      text-align: center;
+      font-size: 14px;
+      background-color: white;
+      color: rgb(210, 40, 123);
+      border-radius: 50% 50%;
+      display: inline-block;
+      width: 28px;
+      height: 28px;
+  }
+
+  .album-song-button:hover {
+      cursor: pointer;
+      color: white;
+      background-color: rgb(210, 40, 123);
+  }
+
+  .album-song-button span {
+      line-height: 28px;
+  }

   /* Centered text looks better on smaller devices */
   @media (max-width: 640px) and (min-width: 320px) {
   ...
```

Also, add a minimum width to `.song-item-number` to account for the width of the button:

~/bloc/bloc-jams/styles/album.css

```
   ...
   .song-item-number {
       width: 5%;
+      min-width: 30px;
   }
   ...
```

Without a minimum width, the cell has the potential to become smaller in width than the size of the button. In this scenario, the song list content would shift to make room for the button when a user hovers over the area to make the button appear. To avoid this, we use a minimum width to make sure there is always room for the button.

We need to restrict the content change to the current table row. When we mouse over a

cell in the item row, whether it's the song's number, title, or duration, we always want to change *only* the cell with the song number. To ensure that this happens every time, we first select the parent element of all three elements, and *then* select the song number's cell.

Add a conditional statement to the `mouseover` event listener that restricts the target to the table row:

~/bloc/bloc-jams/scripts/album.js

```
  ...
      songListContainer.addEventListener('mouseover', function(event) {
-         console.log(event.target);
          // Only target individual song rows during event delegation
+         if (event.target.parentElement.className === 'album-view-song-item') {
+             // Change the content from the number to the play button's HTML
+         }
      });
  ...
```

We use the `parentElement` and `className` properties together to make sure that we only act on the table row. Select the `.song-item-number` element relative to the parent (that is, the table we're mousing-over) and change the `innerHTML` to the play button:

~/bloc/bloc-jams/scripts/album.js

```
  ...
      songListContainer.addEventListener('mouseover', function(event) {
          if (event.target.parentElement.className === 'album-view-song-item') {
-             // Change the content from the number to the play button's HTML
+             event.target.parentElement.querySelector('.song-item-number').innerHTML =
          }
      });
  ...
```

We use the `querySelector()` method because we only need to return a single element with the `.song-item-number` class.

Store the Song Number in the HTML

To revert the play button back to the song's number, we need to store the number before the user gets a chance to mouse over the row. We could do this with JavaScript, but a simpler solution is to use HTML5 **data attributes**. As the name suggests, HTML data attributes allow us to store information in an attribute on an HTML element. Add an attribute called `data-song-number` to the template we generate using the `createSongRow()` function:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {
      var template =
          '<tr class="album-view-song-item">'
-     + '  <td class="song-item-number">' + songNumber + '</td>'
+     + '  <td class="song-item-number" data-song-number="' + songNumber + '">' + songN
      + '  <td class="song-item-title">' + songName + '</td>'
      + '  <td class="song-item-duration">' + songLength + '</td>'
      + '</tr>'
      ;

      return template;
  };
  ...
```

This allows us to access the data held in the attribute using DOM methods when the mouse leaves the table row, and the song number's table cell returns to its original state.

Detect the Mouse Leaving

The DOM uses the `mouseleave` event to signal when a mouse leaves the target element's bounds. For this event, we want to attach event listeners to each table row (instead of using event delegation) because the action of leaving a cell is not something that can be specified as easily by listening on the parent. We will select an array of every table row and loop over each to add its event listener:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  // Elements we'll be adding listeners to
  var songListContainer = document.getElementsByClassName('album-view-song-list')[0];
+ var songRows = document.getElementsByClassName('album-view-song-item');

  var playButtonTemplate = '<a class="album-song-button"><span class="ion-play"></span>

  window.onload = function() {
      setCurrentAlbum(albumPicasso);

      songListContainer.addEventListener('mouseover', function(event) {
          ...
      });

+     for (var i = 0; i < songRows.length; i++) {
+         songRows[i].addEventListener('mouseleave', function(event) {
+             // Revert the content back to the number
+         });
+     }
  }
```

To select the song number container, we'll use a property on the song row DOM selector that gets the first child, and sets the `innerHTML` to the song number:

~/bloc/bloc-jams/scripts/album.js

```
  ...
      for (var i = 0; i < songRows.length; i++) {
          songRows[i].addEventListener('mouseleave', function(event) {
-             // Revert the content back to the number
              // Selects first child element, which is the song-item-number element
+             this.children[0].innerHTML = this.children[0].getAttribute('data-song-nur
          });
      }
  ...
```

The `getAttribute()` method takes a single argument: a string with the name of the attribute whose value we want to retrieve. When the mouse leaves a selected table row, it will change back to the song number using the value obtained from this method.

Recap

Concept	Description
Event Delegation	Event delegation listens for an event on a parent element but targets the behavior on one of its children.
Event Bubbling	Listening for an event on a parent is possible because of event bubbling. An event may fire on a child element, but it bubbles up the DOM tree to its parent.
Data Attributes	HTML data attributes allow us to store information in an attribute on an HTML element.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

12. DOM Scripting: Play/Pause Part 1

Assignment | Discussion | Submission

13 DOM Scripting: Play/Pause Part 2

Overview and Purpose

This checkpoint implements complex JavaScript behavior, which requires forethought and algorithmic thinking.

Objectives

After this checkpoint, students should be able to:

- Assess a problem and attempt to devise a solution in pseudo-code or plain English.
- Translate pseudo-code or plain English actions into JavaScript.

This is the second of two checkpoints completing the play and pause behavior on the Album view.

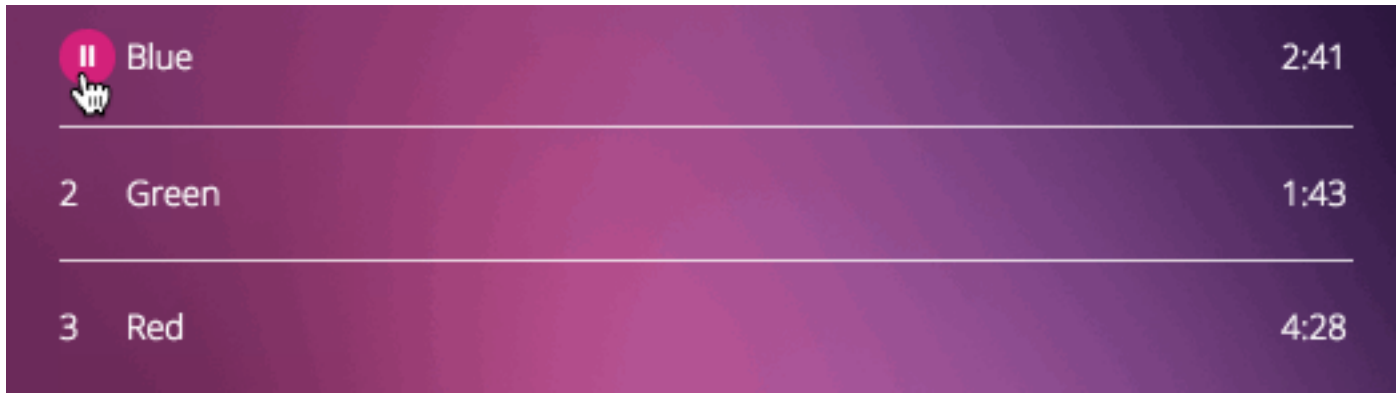
Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

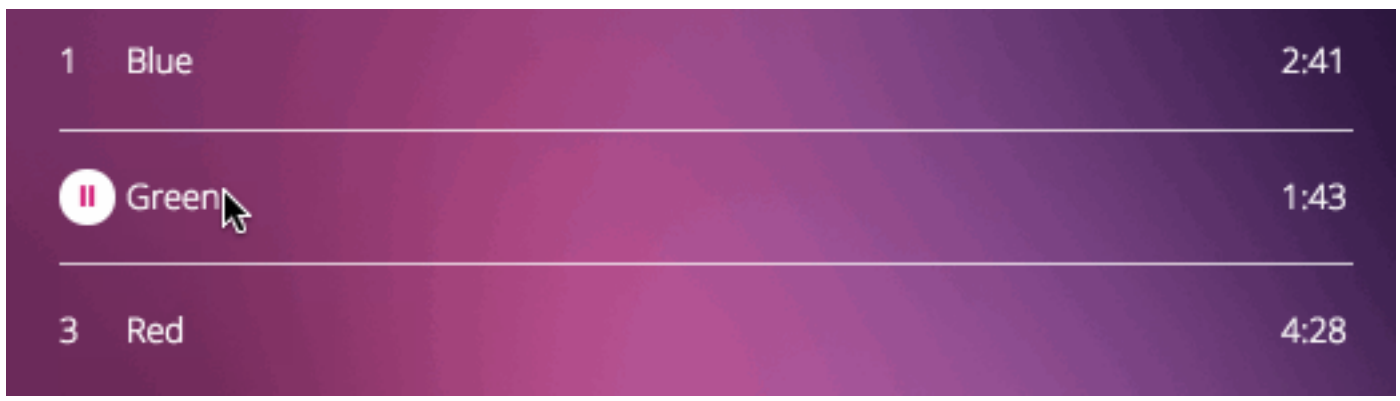
Consider the Behavior Before Writing the Code

This will be the most complex behavior we've implemented in Bloc Jams so far. To help write the code, consider the required functionality in plain English:

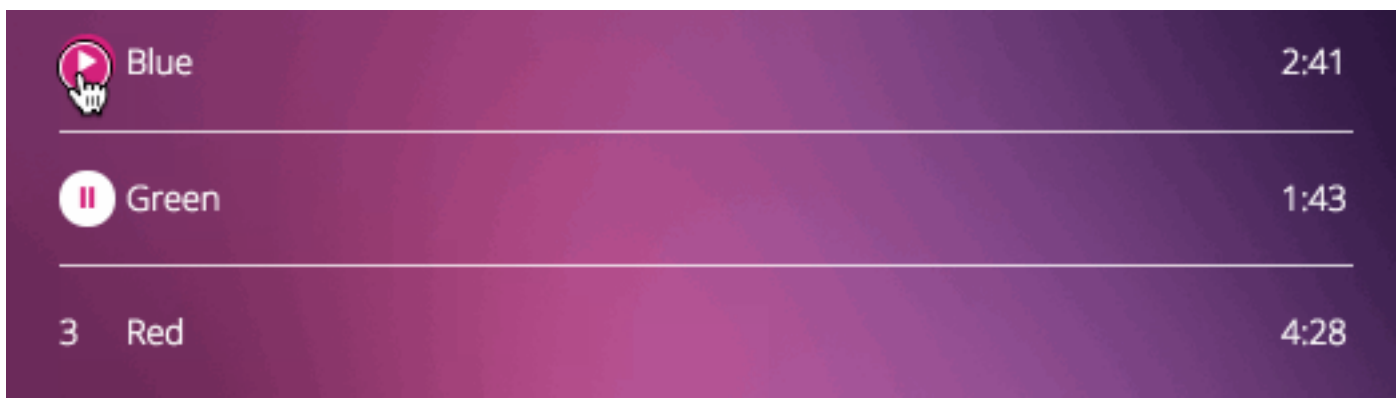
1. When we click a song to play it, the song number should change to a pause button:



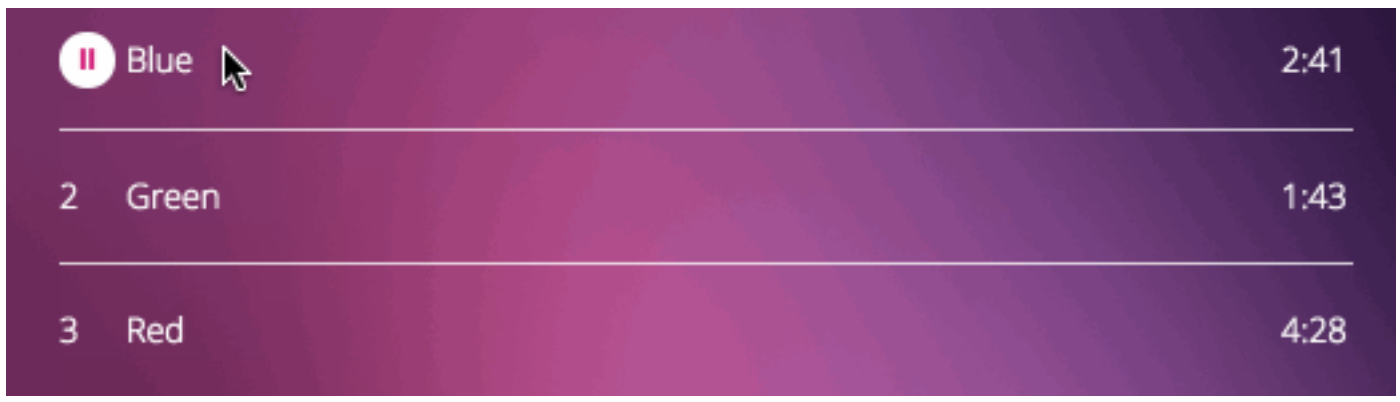
2. When the mouse leaves the table row of the currently playing song, the pause button should remain:



3. When we switch songs, the previously playing song's table cell should revert the content back to the song number:



4. When we hover over each of the songs that aren't playing, the play button should still appear; we *don't*, however, want to show the play button when we hover over the playing song:



Change the Song Number to the Pause Button

When we click to play, switch, or pause a song, we need to change the `innerHTML` of the element with the `.song-item-number` class. There are four different relationships the clicked element can have to the `.song-item-number` table cell:

1. A child, like the icon or the icon's circular container
2. A parent, like the table row
3. A child of the parent, but neither a child nor parent of `.song-item-number`, like the table cells with the classes `.song-item-title` or `.song-item-duration`
4. The `.song-item-number` element itself

We can select items 1 and items 4 using built-in DOM functions, but items 2 and items 3 require us to look up the DOM tree to select a parent element. To address this requirement, write a `findParentByClassName` function that keeps traversing the DOM upward until a parent with a specified class name is found.

Instead of providing the code immediately for this exercise, **you should to try and write it on your own first**. You can check your work by looking at **our implementation**. Write your code below the `setCurrentAlbum` function in `album.js`.

> Message your mentor for help before looking at the solution.

`getSongItem()` Method

The `findParentByClassName` function enables us to write a larger function that will always return the song item. This method, which we'll call `getSongItem`, should take an element and, based on that element's class name(s), use a `switch` statement that returns the element with the `.song-item-number` class.

Once again, **try writing the `getSongItem` function on your own**. We've created **another solution** that you can use to verify your implementation. Write your code below the `findParentByClassName()` function in `album.js`.

> Message your mentor for help before referring to the solution.

Add a `click` Event Listener

Before writing the `click` handler, create a variable to store the template for the pause button, as well as a variable for the currently playing song:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var playButtonTemplate = '<a class="album-song-button"><span class="ion-play"></span>
+ var pauseButtonTemplate = '<a class="album-song-button"><span class="ion-pause"></spa

  // Store state of playing songs
+ var currentlyPlayingSong = null;

  window.onload = function() {
  ...
```

We set it to `null` so that no song is identified as playing until we click one. To register the click that will eventually change the value of `currentlyPlayingSong`, add an event listener for the `click` event in the same `for` loop we created for the `mouseleave` event:

~/bloc/bloc-jams/scripts/album.js

```
  ...
      for (var i = 0; i < songRows.length; i++) {
          songRows[i].addEventListener('mouseleave', function(event) {
              // Selects first child element, which is the song-item-number element
              this.children[0].innerHTML = this.children[0].getAttribute('data-song-nu
          });

+         songRows[i].addEventListener('click', function(event) {
+             // Event handler call
+         });
      }
```

Add a `clickHandler()` Function

Create a function called `clickHandler()` in `album.js` that takes one argument, `targetElement`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
   var getSongItem = function(element) {
       ...
   };

+  var clickHandler = function(targetElement) {
+  };

   // Elements to which we'll be adding listeners
   var songListContainer = document.getElementsByClassName('album-view-song-list')[0];
  ...
```

Store the `.song-item-number` element, selected using the `getSongItem` function, in a variable:

~/bloc/bloc-jams/scripts/album.js

```
   ...
   var clickHandler = function(targetElement) {

+      var songItem = getSongItem(targetElement);

   };
```

Create a conditional that checks if `currentlyPlayingSong` is `null`. If `true`, it should set the `songItem`'s content to the pause button and set `currentlyPlayingSong` to the new song's number:

~/bloc/bloc-jams/scripts/album.js

```
   ...
   var clickHandler = function(targetElement) {

       var songItem = getSongItem(targetElement);

+      if (currentlyPlayingSong === null) {
+          songItem.innerHTML = pauseButtonTemplate;
+          currentlyPlayingSong = songItem.getAttribute('data-song-number');
+      }

   };
   ...
```

Add another conditional to revert the button back to a play button if the playing song is clicked again. Set `currentlyPlayingSong` to `null` after:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var clickHandler = function(targetElement) {

      var songItem = getSongItem(targetElement);

      if (currentlyPlayingSong === null) {
          songItem.innerHTML = pauseButtonTemplate;
          currentlyPlayingSong = songItem.getAttribute('data-song-number');
-     }
+     } else if (currentlyPlayingSong === songItem.getAttribute('data-song-number')) {
+         songItem.innerHTML = playButtonTemplate;
+         currentlyPlayingSong = null;
+     }

  };
  ...
```

If the clicked song is not the active song, set the content of the new song to the pause button:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var clickHandler = function(targetElement) {

      var songItem = getSongItem(targetElement);

      if (currentlyPlayingSong === null) {
          songItem.innerHTML = pauseButtonTemplate;
          currentlyPlayingSong = songItem.getAttribute('data-song-number');
      } else if (currentlyPlayingSong === songItem.getAttribute('data-song-number')) {
          songItem.innerHTML = playButtonTemplate;
          currentlyPlayingSong = null;
-     }
+     } else if (currentlyPlayingSong !== songItem.getAttribute('data-song-number')) {
+         var currentlyPlayingSongElement = document.querySelector('[data-song-number='
+         currentlyPlayingSongElement.innerHTML = currentlyPlayingSongElement.getAttri
+         songItem.innerHTML = pauseButtonTemplate;
+         currentlyPlayingSong = songItem.getAttribute('data-song-number');
+     }

  };
  ...
```

Finally, add the click handler to the event listener:

~/bloc/bloc-jams/scripts/album.js

```
  ...
      for (var i = 0; i < songRows.length; i++) {
          songRows[i].addEventListener('mouseleave', function(event) {
              // Selects first child element, which is the song-item-number element
              this.children[0].innerHTML = this.children[0].getAttribute('data-song-nur
          });

          songRows[i].addEventListener('click', function(event) {
-             // Event handler call
+             clickHandler(event.target);
          });
      }
```

Change the Mouseover and Mouseleave Behavior

We should see the click behavior working with one exception: when the mouse leaves the table cell, the pause button will disappear.

Recall the fourth objective we set before we wrote the code for this checkpoint:

> We still want to show the play button when we hover over each of the songs that aren't playing. However, we *don't* want to show the play button when we hover over the playing song.

We need to add some conditional statements to support the correct behavior.

Update the Code in the `mouseleave` Listener

When we originally wrote the code block inside the listener, we used `this.children[0]` to change the HTML of the table cell. We now have a helper function, `getSongItem()`, that gets this element for us and makes the purpose of the code more explicit.

Remove all references to `this.children[0]`, and add a conditional statement to ensure the row we're exiting does not belong to the currently playing song:

~/bloc/bloc-jams/scripts/album.js

```
  ...
      for (var i = 0; i < songRows.length; i++) {
          songRows[i].addEventListener('mouseleave', function(event) {
-             // Selects first child element, which is the song-item-number element
-             this.children[0].innerHTML = this.children[0].getAttribute('data-song-nur
              // #1
+             var songItem = getSongItem(event.target);
+             var songItemNumber = songItem.getAttribute('data-song-number');
+
              // #2
+             if (songItemNumber !== currentlyPlayingSong) {
+                 songItem.innerHTML = songItemNumber;
+             }
          });

          songRows[i].addEventListener('click', function(event) {
  ...
```

At **#1**, we've cached the song item that we're leaving in a variable. Referencing `getSongItem()` repeatedly causes multiple queries that can hinder performance. We've done the same with the song number.

At **#2**, we've added the conditional that checks that the item the mouse is leaving is not the current song, and we only change the content if it isn't.

Update the Code in the `mouseover` Listener

Finally, update the code in the `mouseover` event with a conditional statement that only changes the `innerHTML` of the table cell when the element does not belong to the currently playing song. **Try implementing the code without guidance first**, and check your work against the **example implementation we've written**.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

13. DOM Scripting: Play/Pause Part 2

14 Programming Reinforcement: Checkpoint 2

Overview and Purpose

This checkpoint introduces three coding challenges to practice object-oriented programming, basic data structures, control flow, algorithms, and closures.

Objectives

After this checkpoint, you should be able to:

- Understand the purpose of an algorithm.
- Apply advanced control flow to an algorithm.
- Explain a closure and how it's used in a program.

Programming Reinforcement

Each foundation checkpoint introduces new concepts, patterns, and assignments. We designed Bloc's curriculum to push you to your learning limits. With that in mind, we've included breaks between lessons to reinforce programming concepts; this is the second of those breaks, known as Programming Reinforcement checkpoints.

In these checkpoints, you will complete coding challenges that help you master JavaScript and learn to think like a programmer. After you complete the challenges, you will meet with

your mentor to discuss your solutions.

- Challenges
- Bonus Challenge: Stacked Deck
- Assignment
  - For Mentors

Challenges

You must solve these three Kata (challenges) before submitting the checkpoint. At this point in your program, your skill level meets or exceeds that required by each Kata. While they may challenge you, know that you are capable of completing each one.

> Your mentor may help you, but we strongly encourage you attempt these on your own.

Kata	Hint
FizzBuzz Array	Complete a classic programming challenge with an array twist.
Jaden Casing Strings	Implement a function that converts convert strings to how they would be written by Jaden Smith.
Transpose Two Strings in an Array	Create a function that will take two strings and transpose them, so that the strings go from top to bottom instead of left to right.

Bonus Challenge: Stacked Deck

> This bonus challenge is optional, but we strongly encourage you to attempt it.

In this challenge, Bloc went all-in on a hand of poker. We had a flush, but Codecademy pulled a ringer and got a full-house. We owe them $3M by next Tuesday and if we don't pay up, they're going to tell everyone we smell.

We're going to raise that money like a gambler… by gambling. To help us win, you will

write a deck-shuffling algorithm. By default, your code must shuffle a deck as per usual: semi-randomly using a **pseudo-random number generator**.

Here's the fun: whenever we pass you a `cheatCode`, you should use that value to build a cheat deck. Whenever we pass you the same cheat code, you must return the same cheat deck each time. Here's an example:

```
var shuffledDeckOne = new StackedDeck();
shuffledDeckOne.shuffle();
var shuffledDeckTwo = new StackedDeck();
shuffledDeckTwo.shuffle();
var cheatDeckOne = new StackedDeck(1);
var cheatDeckTwo = new StackedDeck(1);
cheatDeckOne.shuffle();
cheatDeckTwo.shuffle();
```

`shuffledDeckOne` and `shuffledDeckTwo` are normal decks, shuffled randomly, and are *not* identical. After shuffling `cheatDeckOne` and `cheatDeckTwo`, we expect both of these decks to have the exact same order of cards. We passed the same cheat code to both decks: `1`. Here's another caveat:

```
var cheatDeckA = new StackedDeck(1);
var cheatDeckB = new StackedDeck(2);
var cheatDeckC = new StackedDeck(2);
```

In this example, `cheatDeckB` is identical to `cheatDeckC` but both of them are different than `cheatDeckA`. `cheatDeckA` is identical to `cheatDeckOne` and `cheatDeckTwo` from the previous example. Each cheat code corresponds to a unique deck.

Store the deck in an array:

```
shuffledDeck = [];
```

Each deck must be complete with the full set of 52 unique cards, each represented by a string. For example, `"5c"` for the five of clubs, and `"Qs"` for the queen of spades. Here's the full breakdown:

- **A** = Ace
- **K** = King
- **Q** = Queen
- **J** = Jack

- and **2** through **10** represent the numbered cards

Suits are lower-case letters:

- **d** = Diamonds
- **h** = Hearts
- **c** = Clubs
- and **s** = Spades

Your shuffling must be random enough that each call to `shuffle` generates a different deck unless we provide a cheat code. But please, do *not* mess this up, Bloc is in deep on this one.

How would you rate this checkpoint and assignment? ☺ ☹

15 HTML & CSS: Music Player

Overview and Purpose

This checkpoint reviews basic HTML structure and CSS by implementing a music player bar, which allows a user to control song playback.

Objectives

After this checkpoint, students should be able to:

- Comfortably create and link a CSS file.
- Add new HTML elements to a larger HTML structure.
- Readily style HTML elements.

Bloc Jams is getting closer to playing music! Before it can do that, though, we'll need a music player to control song playback. The music player will allow users to pause and play songs, view playback progress, and adjust volume. We will implement the script to make the player functional in another checkpoint. For now, we'll build the view structure and make it responsive.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Create the CSS File

Create a CSS file for the music player bar in the `styles` directory:

~/bloc/bloc-jams/styles

```
$ touch player_bar.css
```

Link the CSS file to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
          <link rel="stylesheet" type="text/css" href="styles/album.css">
+         <link rel="stylesheet" type="text/css" href="styles/player_bar.css">
      </head>
  ...
```

Set Up the Music Player

The music player will perform distinct functions that we'll separate into different groups. Our player bar will have three "control groups":

1. *Main controls*, which contains the play/pause and previous/next buttons.
2. *Currently playing*, which displays the currently playing song information, including the song progress.
3. *Volume*, which contains the volume control slider.

We will associate corresponding class names to each of these groups: `main-controls`, `currently-playing`, and `volume`, respectively.

In `album.html`, create a `<section>` for the music player bar and add the `<div>`s that will hold each group:

~/bloc/bloc-jams/album.html

```
  ...
        </main>
+
+       <section class="player-bar">
+           <div class="container">
+               <div class="control-group main-controls">
+               </div>
+               <div class="control-group currently-playing">
+               </div>
+               <div class="control-group volume">
+               </div>
+           </div>
+       </section>
        <script src="scripts/album.js"></script>
    </body>
  </html>
```

In `player_bar.css`, add the following styles to get our music player started:

~/bloc/bloc-jams/styles/player_bar.css

```
+ .player-bar {
+     position: fixed;
+     bottom: 0;
+     left: 0;
+     right: 0;
+     height: 200px;
+     background-color: rgba(255, 255, 255, 0.3);
+     z-index: 100;
+ }
+
+ .player-bar a,
+ .player-bar a:hover {
+     color: white;
+     cursor: pointer;
+     text-decoration: none;
+ }
+
+ .player-bar .container {
+     display: table;
+     padding: 0;
+     width: 90%;
+     min-height: 100%;
+ }
+
+ .player-bar .control-group {
+     display: table-cell;
+     vertical-align: middle;
+ }
+
+ .player-bar .main-controls {
+     width: 25%;
+     text-align: left;
+     padding-right: 1rem;
+ }
+
+ .player-bar .currently-playing {
+     width: 50%;
+     text-align: center;
+     position: relative;
+ }
+
+ .player-bar .volume {
+     width: 25%;
+     text-align: right;
+ }
```

We want the `.currently-playing` (middle) group to display wider than the other control

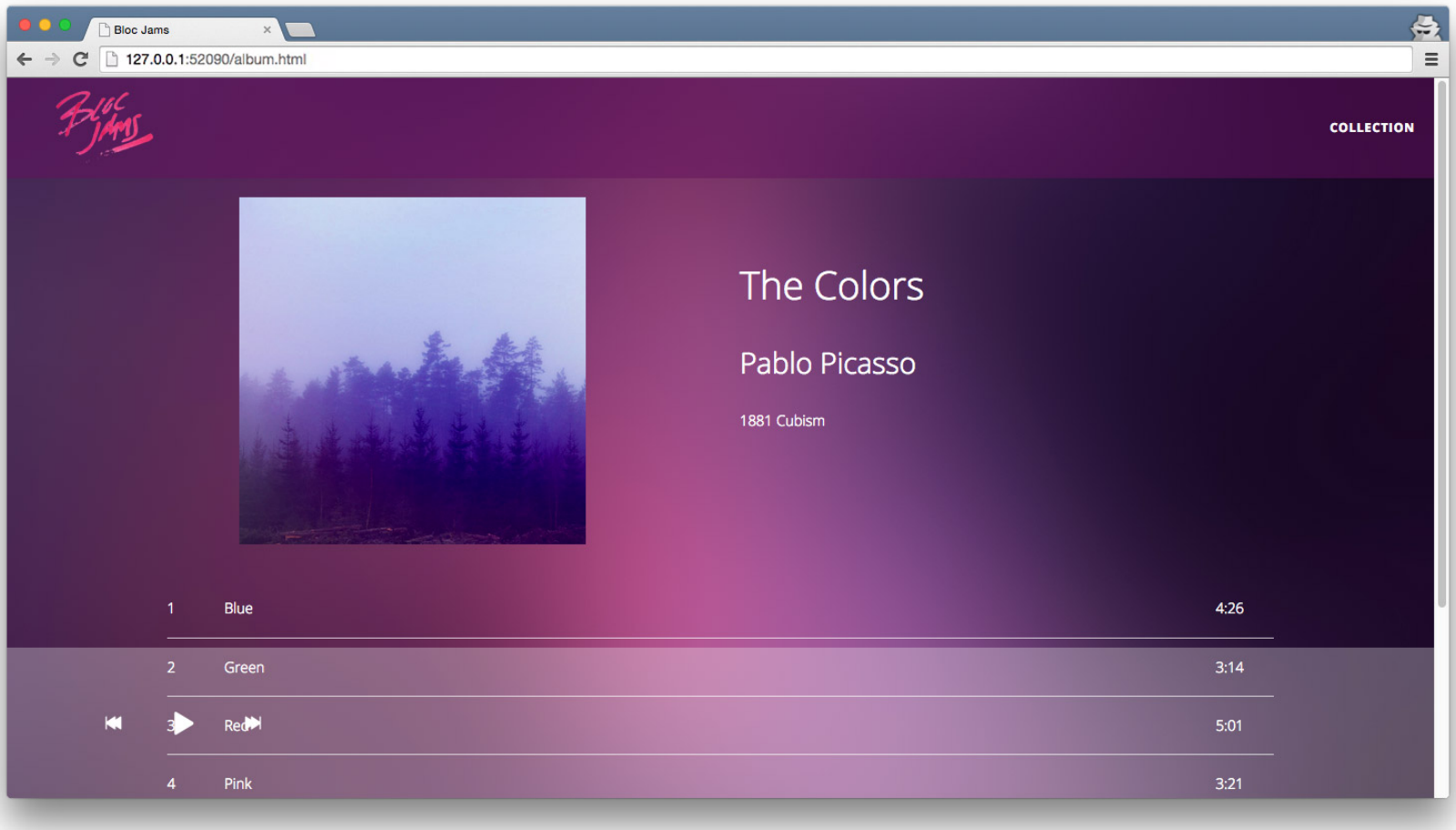
groups, which is why its width property is largest. Additionally, we've fixed the position of the player bar to the bottom of the window and gave it a height of `200px`. To accommodate this fixed element on the album page, add this style to `album.css`:

~/bloc/bloc-jams/styles/album.css

```
  body.album {
      background-image: url(../assets/images/blurred_backgrounds/blur_bg_3.jpg);
      background-repeat: no-repeat;
      background-attachment: fixed;
      background-position: center center;
      background-size: cover;
+     padding-bottom: 200px;
  }
  ...
```

This style adds padding to the bottom of the album page. This allows the user to scroll to the end of the album page and still view all the content, even with the player bar fixed at the bottom. Without this style, the music player may obscure the content beneath it, which would make for a poor user experience.

Add Main Controls



We'll set up the main controls with previous/next and play/pause buttons. Add the following to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
                <div class="control-group main-controls">
+                   <a class="previous">
+                       <span class="ion-skip-backward"></span>
+                   </a>
+                   <a class="play-pause">
+                       <span class="ion-play"></span>
+                   </a>
+                   <a class="next">
+                       <span class="ion-skip-forward"></span>
+                   </a>
                </div>
  ...
```

We use `<span>` tags to display icons that represent the buttons. The play/pause button is initialized with the "play" icon.

In `player_bar.css`, add the following code to style the anchor tags:

~/bloc/bloc-jams/styles/player_bar.css

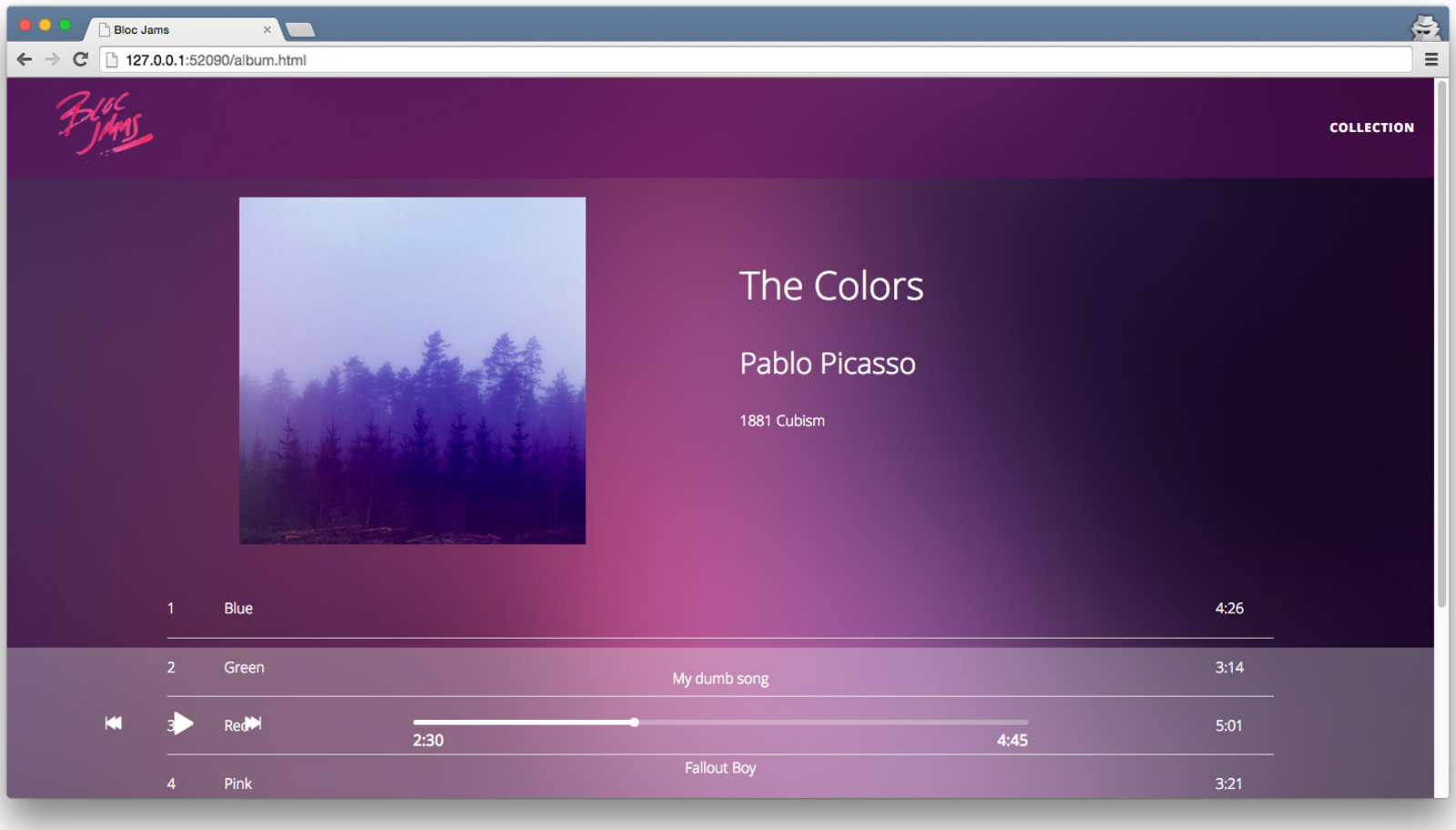
```
  ...
+ .player-bar a {
+     font-size: 1.1rem;
+     vertical-align: middle;
+ }
+
  .player-bar a,
  .player-bar a:hover {
  ...
```

~/bloc/bloc-jams/styles/player_bar.css

```
   ...
   .player-bar .main-controls {
       width: 25%;
       text-align: left;
       padding-right: 1rem;
   }
+
+  .player-bar .main-controls .previous {
+      margin-right: 16.5%;
+  }
+
+  .player-bar .main-controls .play-pause {
+      margin-right: 15%;
+      font-size: 1.6rem;
+  }
   ...
```

We adjust the position of the icons using `margin` and `vertical-align`. We also adjust the size of the icons to make them easier to see.

Add Currently Playing Controls



The center of the player bar contains details about the currently playing song. Add the following to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
                <div class="control-group currently-playing">
+                   <h2 class="song-name">My dumb song</h2>
+                   <h2 class="artist-song-mobile">My dumb song - Fallout Boy</h2>
+                   <h3 class="artist-name">Fallout Boy</h3>
                </div>
  ...
```

We've given each of the details a heading tag, a corresponding class name, and placeholder information. We've created a `.artist-song-mobile` heading that joins the song name and artist in one line to make it more compact. We'll style this element so that it only displays on mobile devices.

Add the following to `player_bar.css`:

~/bloc/bloc-jams/styles/player_bar.css

```
   ...
   .player-bar .currently-playing {
       width: 50%;
       text-align: center;
       position: relative;
   }
+
+ .player-bar .currently-playing .song-name,
+ .player-bar .currently-playing .artist-name,
+ .player-bar .currently-playing .artist-song-mobile {
+     text-align: center;
+     font-size: 0.75rem;
+     margin: 0;
+     position: absolute;
+     width: 100%;
+     font-weight: 300;
+ }
+
+ .player-bar .currently-playing .song-name,
+ .player-bar .currently-playing .artist-song-mobile {
+     top: 1.1rem;
+ }
+
+ .player-bar .currently-playing .artist-name {
+     bottom: 1.1rem;
+ }
+
+ .player-bar .currently-playing .artist-song-mobile {
+     display: none;
+ }
   ...
```

The "currently playing" control group will also contain a seek bar to allow the user to jump to any point in the song.

Add the following to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
                <div class="control-group currently-playing">
                    <h2 class="song-name">My dumb song</h2>
+                   <div class="seek-control">
+                       <div class="seek-bar">
+                           <div class="fill"></div>
+                           <div class="thumb"></div>
+                       </div>
+                       <div class="current-time">2:30</div>
+                       <div class="total-time">4:45</div>
+                   </div>
                    <h2 class="artist-song-mobile">My dumb song - Fallout Boy</h2>
                    <h3 class="artist-name">Fallout Boy</h3>
                </div>
  ...
```

The seek bar displays the current playback time and the full length duration of the song. For now, we've used placeholder times.

The `fill` class element in `.seek-bar` will display the current song's playback progress. As the song plays, the width of `.fill` increases until it reaches 100%, the full width of `.seek-bar`.

`.thumb` is a visual marker that makes it easier for users to identify the end of the `.fill` element. To make it easy for users to drag and drop the `.thumb`, we'll style it as a small circle.

Add the seek bar styles to `player_bar.css`:

~/bloc/bloc-jams/styles/player_bar.css

```
  ...
  .player-bar .currently-playing .artist-song-mobile {
      display: none;
  }
+
+ .seek-control {
+     position: relative;
+     font-size: 0.8rem;
+ }
+
+ .seek-control .current-time {
+     position: absolute;
+     top: 0.5rem;
+ }
+
+ .seek-control .total-time {
```

```
+     position: absolute;
+     right: 0;
+     top: 0.5rem;
+ }
+
+ .seek-bar {
+     height: 0.25rem;
+     background-color: rgba(255, 255, 255, 0.3);
+     border-radius: 2px;
+     position: relative;
+     cursor: pointer;
+ }
+
+ .seek-bar .fill {
+     background-color: white;
+     width: 36%;
+     height: 0.25rem;
+     border-radius: 2px;
+ }
+
+ .seek-bar .thumb {
+     position: absolute;
+     height: 0.5rem;
+     width: 0.5rem;
+     background-color: white;
+     left: 36%;
+     top: 50%;
      /* #1 */
+     margin-left: -0.25rem;
+     margin-top: -0.25rem;
+     border-radius: 50%;
+     cursor: pointer;
+     -webkit-transition: all 100ms ease-in-out;
+        -moz-transition: all 100ms ease-in-out;
+             transition: all 100ms ease-in-out;
+ }
+
+ .seek-bar:hover .thumb {
+     width: 1.1rem;
+     height: 1.1rem;
+     margin-top: -0.5rem;
+     margin-left: -0.5rem;
+ }
  ...
```

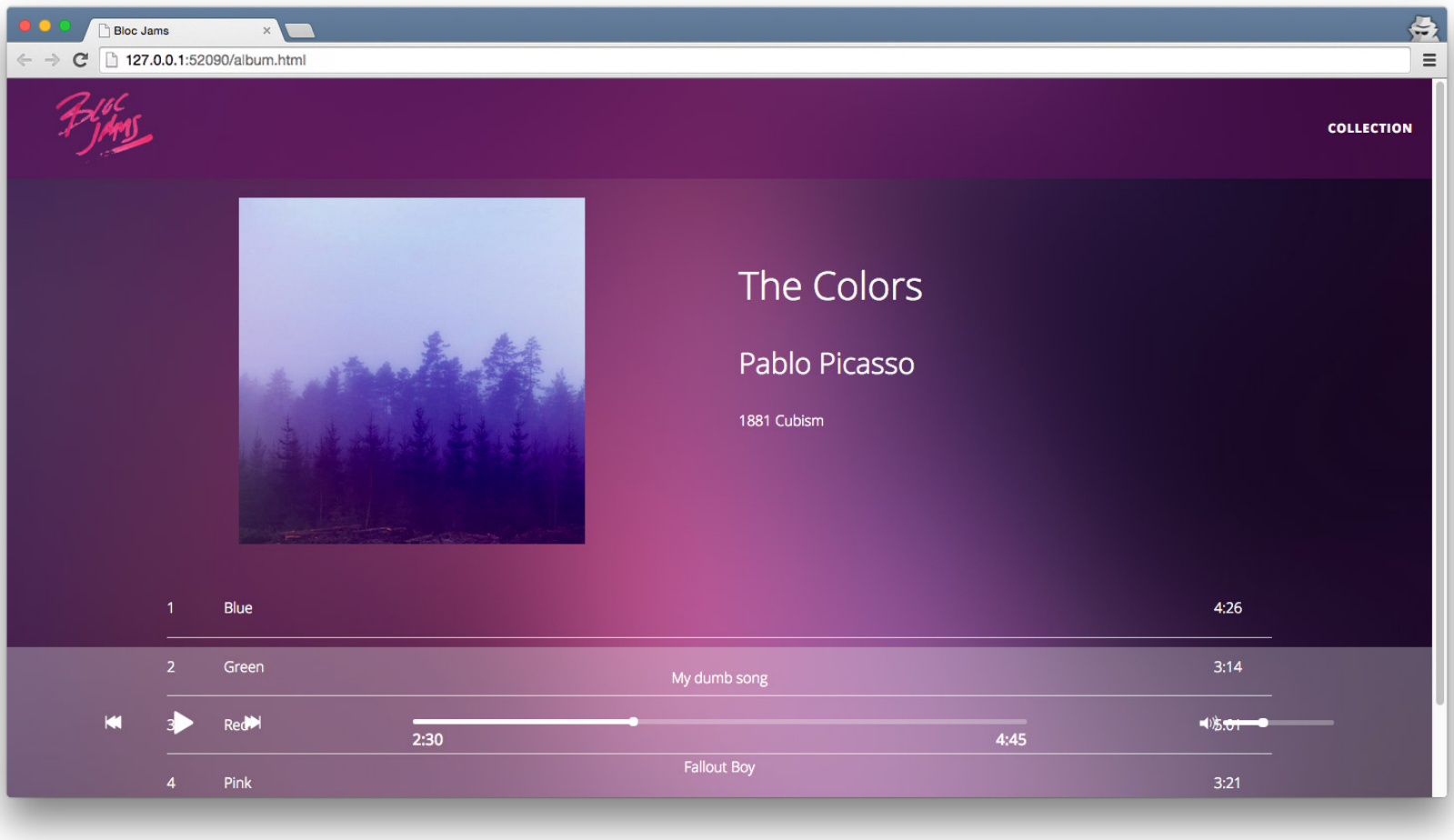
We use properties we've seen before to set position, color, and size of the seek bar elements.

`.fill` is a static element that appears as a line. For the time being, we've given it a `36%` width so that we can see the element when we view the page.

We've set the `border-radius` of `.thumb` to `50%`, causing the corners of the element to round out enough to appear circular. Additionally, we've styled `.thumb` to increase in size when a user hovers over it so it's easier to click and drag. Negative margins (at **#1**) account for the size increase and hold the element's position in place so that the change in size doesn't adjust the position of other elements.

Spend some time to style the seek bar in different ways. Adjust the various properties to see how different values alter the view. When you feel comfortable with these properties, reapply these proposed styles to maintain the original design.

Add Volume Controls



The volume control group contains the volume adjuster. Add the following markup to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
                <div class="control-group volume">
+                   <span class="ion-volume-high icon"></span>
+                   <div class="seek-bar">
+                       <div class="fill"></div>
+                       <div class="thumb"></div>
+                   </div>
                </div>
  ...
```

The volume icon visually informs the user that this control adjusts the volume. We also include another seek bar element so that the user can adjust the volume as desired.

Add the following styles to `player_bar.css`:

~/bloc/bloc-jams/styles/player_bar.css

```
  ...
  .player-bar .volume {
      width: 25%;
      text-align: right;
  }
+
+ .player-bar .volume .icon {
+     font-size: 1.1rem;
+     display: inline-block;
+     vertical-align: middle;
+ }
+
+ .player-bar .volume .seek-bar {
+     display: inline-block;
+     width: 5.75rem;
+     vertical-align: middle;
+ }
```

We style the volume icon so that it's easily viewable.

Experiment with these properties to create a new look for the volume controls. When you feel comfortable with these properties and how they affect elements, revert back to the original styles.

Mobile

The player bar collapses on itself when viewed on smaller screens, rendering it unusable.

We need to change it so that past a certain screen width, the `.control-group` elements stack vertically instead of horizontally.

Add the mobile styles to `player_bar.css`:

~/bloc/bloc-jams/styles/player_bar.css

```
  ...
  .player-bar .volume .seek-bar {
      display: inline-block;
      width: 5.75rem;
      vertical-align: middle;
  }
+
+ @media (max-width: 640px) {
+     .player-bar {
+         padding: 1rem;
+         background-color: rgba(0,0,0,0.6);
+     }
+
+     .player-bar .main-controls,
+     .player-bar .currently-playing,
+     .player-bar .volume {
+         display: block;
+         margin: 0 auto;
+         padding: 0;
+         width: 100%;
+         text-align: center;
+     }
+
+     .player-bar .main-controls,
+     .player-bar .volume {
+         min-height: 3.5rem;
+     }
+
+     .player-bar .currently-playing {
+         min-height: 2.5rem;
+     }
+
+     .player-bar .artist-name,
+     .player-bar .song-name {
+         display: none;
+     }
+
+     .player-bar .currently-playing .artist-song-mobile {
+         display: block;
+     }
+ }
```

We add a semi-transparent black background color to the player bar so that the white text stands out better. We also change each control group element to `display: block`, stacking them vertically, and set `width: 100%`, expanding them to the width of the screen. We give them a minimum height to ensure they have enough space to display well. We hide the individual song and artist titles with `display: none` and reveal the compact, single-line version.

Recap

Concept	Description
Icons	Icons are used in websites and applications to visually communicate information to the user.
Mobile-Friendly Views	Mobile-friendly designs are important for maximizing usability.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

16 jQuery: Landing Page

Overview and Purpose

This checkpoint introduces jQuery, a JavaScript helper library to simplify development.

Objectives

After this checkpoint, students should be able to:

- Discuss the pros and cons of using a helper library.
- Include jQuery in an application.
- Use jQuery selectors correctly and understand the object(s) returned.
- Recognize and use some basic jQuery methods – such as `.css()` and `.each()`.

To make Bloc Jams more dynamic, we've been using vanilla JavaScript – no bells, no whistles, just JavaScript in its purest form. DOM scripting is "lightweight," as it requires no libraries that may slow page performance.

Anything we can accomplish with a helper library, such as jQuery, we can achieve with JavaScript. However, libraries execute many common, time-consuming tasks that simplify development. Libraries are critical to any developer's success, but knowing which to learn and master can be difficult.

DOM scripting is useful for contributing to open source projects, too. External packages, in particular, often shouldn't depend on other external packages like jQuery.

Experience with DOM scripting is essential, but frontend developers require knowledge of at least a few common libraries. jQuery is the most popular JavaScript library today. We'll use it to refactor the vanilla JavaScript we've implemented for the Bloc Jams landing page and learn why it's so widely used.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Add the jQuery Library

To include the jQuery library in our Bloc Jams project, add `<script>` tags and the jQuery source file in `index.html`:

~/bloc/bloc-jams/index.html

```
  ...
          </section>
+         <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
          <script src="scripts/landing.js"></script>
      </body>
  </html>
```

Here, the jQuery source library is hosted on code.jquery.com, known as a **content delivery network** (or CDN). A CDN is any network that hosts assets you use on your website. Alternatively, we could host the jQuery source ourselves by downloading the source and adding it to the scripts local to our project.

When should I use a content delivery network? There are various pros and cons to using a CDN, and whether or not you should use CDN-hosted assets depends on your project needs. There's no one-size-fits-all answer.

Use jQuery Selectors

We've learned a number of DOM selectors, and a number of methods to select DOM objects (nodes). They aren't supported equally in all browsers, however. jQuery simplifies DOM selection in two ways:

1. jQuery uses a unified selector function written with a dollar sign and parentheses: `$()`.
2. This function works consistently across nearly all browsers.

> Note: `$` is a valid function name in JavaScript, and in jQuery is just an alias for the `jQuery` function.

For example, to select all `<div>` elements using the jQuery selector we would write:

```
$('div')
```

Another advantage of the jQuery selector is that it uses CSS-style syntax. For example, to select all `.cool` elements that have a `#radical` parent element, we would write:

```
$('#radical .cool')
```

We would use this same syntax in a CSS file to style all `.cool` elements that have a `#radical` parent element. This helps simplify element selection. The selector syntax can include pseudo classes like `:hover` as well:

```
$('#radical .cool:hover')
```

Like other selectors, the jQuery selector returns an object. Any time we use `$()` to select an element, the element becomes a **jQuery object**. jQuery *wraps* elements to provide special **jQuery methods** not available on plain JavaScript objects. These additional methods reduce the amount of code required to manipulate the DOM, handle events, and animate elements.

Recall how we styled elements using pure JavaScript:

```
var domElement = document.getElementById("refrigerator");
domElement.style.color = "white";
```

We can perform this same action in jQuery with much less code on our part:

```
$('#refrigerator').css('color', 'white');
```

The jQuery `css()` method gets or sets the style properties of an element. In this case, we *set* the `color` property to `"white"`.

When we want to use a jQuery method on an element we didn't obtain via jQuery selection, we need to wrap the object:

```
var domElement = document.getElementById("refrigerator");
var $newDomElement = $(domElement);
$newDomElement.find('.food'); // Finds all .food elements using the jQuery method
```

> It's a common convention to prefix jQuery variables with a dollar sign so they are easily identified as variables on which jQuery methods can be called.

Refactor `window.onload` with the jQuery Selector

We'll first refactor `window.onload` in `landing.js`. Update the following:

/bloc/bloc-jams/scripts/landing.js

```
  ...
- window.onload = function() {
+ $(window).load(function() {
      if (window.innerHeight > 950) {
  ...
```

We've made the `window` object a jQuery object. We've also changed the `onload` property to the jQuery `load()` method, which takes a function as an argument. When the page loads, the function will execute.

Let's refactor the rest of the function. Make the following changes to `landing.js`:

/bloc/bloc-jams/scripts/landing.js

```
   ...
   $(window).load(function() {
-      // Automatically animates the points on a tall screen where scrolling can't trigg
-      if (window.innerHeight > 950) {
-          animatePoints(pointsArray);
       // #1
+      if ($(window).height() > 950) {
+          animatePoints();
       }

-      var sellingPoints = document.getElementsByClassName('selling-points')[0];
-      var scrollDistance = sellingPoints.getBoundingClientRect().top - window.innerHeig
       // #2
+      var scrollDistance = $('.selling-points').offset().top - $(window).height() + 200

-      window.addEventListener("scroll", function(event) {
       // #3
+      $(window).scroll(function(event) {
-          if (document.documentElement.scrollTop || document.body.scrollTop >= scrollDi
-              animatePoints(pointsArray);
           // #4
+          if ($(window).scrollTop() >= scrollDistance) {
+              animatePoints();
           }
       });
-  }
+  });
```

We add `$()` to convert all instances of `window` into a jQuery object. At **#1** and **#2**, we update the `.innerHeight` property to jQuery's `height()` method, which gets or sets an object's height. Since we pass no arguments to the function, we *get* the height.

At **#2**, we no longer need a separate variable to hold the `.selling-points` element since jQuery can select the element with many fewer characters. We replace `getBoundingClientRect()` with jQuery's `.offset()` method.

At **#3**, the `addEventListener()` method becomes jQuery's `scroll()` method, which takes a function as an argument. jQuery's `scroll()` "method" is still an event handler like `addEventListener()`, but the jQuery wrapper obscures the appearance of events. When the window scrolls, the function executes.

Lastly, at **#4**, we replace `document.documentElement.scrollTop || document.body.scrollTop` with the jQuery equivalent of `$(window).scrollTop()`.

Refactor `animatePoints()`

During the `window.onload` update, we removed the `pointsArray` argument from `animatePoints()`. We'll refactor `animatePoints()` to learn why.

Update the following code in `landing.js`:

/bloc/bloc-jams/scripts/landing.js

```
   // #5
-  var pointsArray = document.getElementsByClassName('point');
-
-  var animatePoints = function(points) {
+  var animatePoints = function() {
-      var revealPoint = function(index) {
-          points[index].style.opacity = 1;
-          points[index].style.transform = "scaleX(1) translateY(0)";
-          points[index].style.msTransform = "scaleX(1) translateY(0)";
-          points[index].style.WebkitTransform = "scaleX(1) translateY(0)";
+      var revealPoint = function() {
           // #7
+          $(this).css({
+              opacity: 1,
+              transform: 'scaleX(1) translateY(0)'
+          });
       };
-      for (var i = 0; i < points.length; i++) {
-          revealPoint(i);
-      }
       // #6
+      $.each($('.point'), revealPoint);
   };
   ...
```

First, we remove the DOM selector (at **#5**) that gets the `.point` elements by class name. The jQuery selection of `.point`, added at **#7**, is terse enough that we don't need to store it in a variable anymore.

Also at **#6**, the `revealPoint` function no longer requires an argument. We replace the `for` loop with the jQuery `$.each()` function. The `$.each()` function iterates over each `.point` element and executes the callback function, `revealPoint`.

At **#7**, we use the jQuery `css()` method in place of the multiple `style` property instances. Due to jQuery's cross-browser compatibility, we don't need to use vendor prefixes on the `transform` property.

Additionally, the `revealPoint` function now refers to `$(this)` instead of a specific `.point` element. To use `this` with jQuery, we must wrap it in a jQuery object. `$(this)` (at **#7**) references a different `.point` element each time jQuery executes the `revealPoint()` callback.

Recap

Concept	Description
Libraries	JavaScript libraries execute many common, time-consuming tasks that simplify development.
jQuery	jQuery is the most popular JavaScript library today. It simplifies many common operations and works across a multitude of browsers.
Content Delivery Network (CDN)	There are **pros** and **cons** to using a CDN.
jQuery Selectors	jQuery selectors allow you to select and manipulate HTML elements using CSS-style syntax.
jQuery API	jQuery has many methods for event handling, animation, document manipulation, etc. • `.css()` • `.load()` • `$.each()`

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

16. jQuery: Landing Page

Assignment | Discussion | Submission

17 jQuery: Collection and Album Views

Overview and Purpose

This checkpoint continues the introduction to jQuery.

Objectives

After this checkpoint, students should be able to:

- Refactor vanilla JavaScript into jQuery.
- Recognize and use a number of jQuery methods – such as `.empty()`, `.append()`, `.text()`, etc.

We'll refactor the collection and album pages to continue exploring jQuery.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Add the jQuery Library

Before we begin refactoring we'll need to link jQuery to our collection and album pages.

Add the jQuery library to `collection.html`:

~/bloc/bloc-jams/collection.html

```
  ...
          </section>
+         <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
          <script src="scripts/collection.js"></script>
      </body>
  </html>
```

Add the jQuery library to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
          </section>
+         <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
          <script src="scripts/album.js"></script>
      </body>
  </html>
```

Refactor `collection.js`

Make the following changes to `collection.js`:

~/bloc/bloc-jams/scripts/collection.js

```
- var collectionItemTemplate =
  // #1
+ var buildCollectionItemTemplate = function() {
+     var template =
        '<div class="collection-album-container column fourth">'
      + '  <img src="assets/images/album_covers/01.png"/>'
      + '  <div class="collection-album-info caption">'
         ...
      + '  </div>'
      + '</div>'
      ;
+
      // #2
+     return $(template);
+ };

  window.onload = function() {
  ...
```

We change the name of the variable that stores the template from `collectionItemTemplate` to `template`. Although we don't use any jQuery methods, we may later. To support that, we wrap `template` in a jQuery object (at **#2**) to future-proof it.

With that in mind, we wrap the template in a function. This function returns the markup string as a jQuery object, which we'll call a *jQuery template*. Note that when naming action-oriented functions, it's a convention to start the function name with a verb. In that spirit, we name the function `buildCollectionItemTemplate` (at **#1**).

Let's refactor the `window.onload` function as we did for `landing.js`. Update the following code in `collection.js`:

~/bloc/bloc-jams/scripts/collection.js

```
- window.onload = function() {
+ $(window).load(function() {
-     var collectionContainer = document.getElementsByClassName('album-covers')[0];
      // #3
+     var $collectionContainer = $('.album-covers');
-     collectionContainer.innerHTML = '';
      // #4
+     $collectionContainer.empty();

      for (var i = 0; i < 12; i++) {
-         collectionContainer.innerHTML += collectionItemTemplate;
+         var $newThumbnail = buildCollectionItemTemplate();
          // #5
+         $collectionContainer.append($newThumbnail);
      }
- }
+ });
```

As in `landing.js`, we change `window.onload` to its jQuery equivalent of `$(window).load()`. Similarly, at **#3**, we substitute DOM selection with the shorter jQuery alternative. When the element selection becomes a jQuery object, we prefix the `collectionContainer` variable name with a `$`, a convention that identifies jQuery-related variables.

At **#4**, we replace the vanilla DOM scripting `innerHTML` property with the jQuery `empty()` method. The `empty()` method, like many jQuery operations, is literal in what it does – it empties, or removes, any text or other elements from the element(s) it is called on.

Lastly, at **#5**, we replace `+=` in the `for` loop with the `append()` method. With each loop, we append the template content to the collection container.

View the collection page in the browser to confirm that the jQuery performs the same as our original vanilla JavaScript implementation.

Refactor `album.js`

Let's continue exploring jQuery by refactoring `album.js`.

In `album.js`, update the `createSongRow` function to return a jQuery template:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {
      var template =
        '<tr class="album-view-song-item">'
      + '  <td class="song-item-number" data-song-number="' + songNumber + '">' + songN
      + '  <td class="song-item-title">' + songName + '</td>'
      + '  <td class="song-item-duration">' + songLength + '</td>'
      + '</tr>'
      ;

-     return template;
+     return $(template);
  };
  ...
```

Next, refactor the DOM selectors in the `setCurrentAlbum` function:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var setCurrentAlbum = function(album) {
-     var albumTitle = document.getElementsByClassName('album-view-title')[0];
-     var albumArtist = document.getElementsByClassName('album-view-artist')[0];
-     var albumReleaseInfo = document.getElementsByClassName('album-view-release-info')
-     var albumImage = document.getElementsByClassName('album-cover-art')[0];
-     var albumSongList = document.getElementsByClassName('album-view-song-list')[0];
+     var $albumTitle = $('.album-view-title');
+     var $albumArtist = $('.album-view-artist');
+     var $albumReleaseInfo = $('.album-view-release-info');
+     var $albumImage = $('.album-cover-art');
+     var $albumSongList = $('.album-view-song-list');
  ...
```

We replace each instance of `getElementsByClassName` with a jQuery selector and use CSS-style syntax to select the elements. Additionally, we add a `$` to the start of each variable name because they now reference jQuery objects.

Refactor the values assigned to the album detail elements:

~/bloc/bloc-jams/scripts/album.js

```
  ...
-     albumTitle.firstChild.nodeValue = album.title;
-     albumArtist.firstChild.nodeValue = album.artist;
-     albumReleaseInfo.firstChild.nodeValue = album.year + ' ' + album.label;
-     albumImage.setAttribute('src', album.albumArtUrl);
+     $albumTitle.text(album.title);
+     $albumArtist.text(album.artist);
+     $albumReleaseInfo.text(album.year + ' ' + album.label);
+     $albumImage.attr('src', album.albumArtUrl);
  ...
```

We call jQuery's `text()` method to replace the content of the text nodes, instead of setting `firstChild.nodeValue`. We also change the `setAttribute()` method to jQuery's `attr()` method, which changes the element attribute using the same arguments.

When a jQuery selector returns a single element, we can access it without array-index syntax. For example, we can call a jQuery method directly on a selector without recovering the first (and only) item in the array.

Apply the following final changes to `album.js`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
-     albumSongList.innerHTML = '';
+     $albumSongList.empty();

      for (var i = 0; i < album.songs.length; i++) {
-         albumSongList.innerHTML += createSongRow(i + 1, album.songs[i].title, album.s
+         var $newRow = createSongRow(i + 1, album.songs[i].title, album.songs[i].durat
+         $albumSongList.append($newRow);
      }
  };
```

Once again, we refactor with jQuery using the `empty()` and `append()` jQuery methods.

View the album page in the browser to confirm that the jQuery performs the same as our original vanilla JavaScript implementation.

Note that jQuery and JavaScript work harmoniously. Even when refactoring some of the code in `album.js` to jQuery, everything done with basic DOM scripting still functions.

Recap

Concept	Description
`.empty()`	Removes all child nodes of the set of matched elements from the DOM.
`.append()`	Insert content, specified by the parameter, to the end of each element in the set of matched elements.
`.text()`	Get or set the content of each element in the set of matched elements.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

18 jQuery: Play/Pause

Overview and Purpose

This checkpoint continues the introduction to jQuery.

Objectives

After this checkpoint, students should be able to:

- Refactor vanilla JavaScript into jQuery.
- Recognize and use a number of jQuery methods – such as `.find()`, `.click()`, `.hover()`, etc.

The addition of jQuery has made the Bloc Jams codebase more concise. We'll continue to tighten our methods in `album.js` by moving the event handling into the `createSongRow()` function. This will reduce our code in the `$document.ready()` block to a single function call.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Prepare `createSongRow()` to Handle Events

We can attach event listeners to dynamically created elements before we add them to the DOM. As part of our refactor, we'll pass events to the jQuery object we create at the

beginning of `createSongRow()`.

To start, remove the return statement at the end of assigning `template`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {

      ...

-     return $(template);
+     var $row = $(template);
  };
  ...
```

Attach the jQuery equivalent to the three event listeners we've been using thus far to `$row`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {

      ...

      var $row = $(template);
      // #1
+     $row.find('.song-item-number').click(clickHandler);
      // #2
+     $row.hover(onHover, offHover);
      // #3
+     return $row;
  };
  ...
```

The jQuery `find()` method at **#1** is similar to `querySelector()`. We call it here to find the element with the `.song-item-number` class that's contained in whichever row is clicked. jQuery's `click` event listener executes the callback we pass to it when the target element is clicked. Notice that `clickHandler()` no longer takes any arguments, which we'll address in our `clickHandler()` refactor.

The `hover()` event listener at **#2** combines the `mouseover` and `mouseleave` functions we relied on previously. The first argument is a callback that executes when the user mouses over the `$row` element and the second is a callback executed when the mouse leaves `$row`.

At **#3**, we return `$row`, which is created with the event listeners attached.

Revisiting `this`

Remember our old friend `this`? Recall that `this` is a contextual object in JavaScript. Unless otherwise stated using `apply()`, `call()`, or `bind()`, `this` refers to the object that is calling the method which relies on `this`. As we refactor our play and pause button functions during this checkpoint, we'll use `this` to reference the jQuery objects to which we've attached event listeners.

Define `onHover` and `offHover`

The definitions of the `onHover` and `offHover` functions need to move to `createSongRow()`.

> Note that these are currently anonymous functions called in the event listeners for the `mouseleave` and `mouseover` events.

Add the functions just above our event handlers:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {

      ...

      var $row = $(template);
+
+     var onHover = function(event) {
+         // Placeholder for function logic
+     };
+     var offHover = function(event) {
+         // Placeholder for function logic
+     };
+
      $row.find('.song-item-number').click(clickHandler);
      $row.hover(onHover, offHover);
      return $row;
  };
  ...
```

Attempt to write the `onHover` and `offHover` functions.

The implementations will be similar to the code in the corresponding vanilla DOM scripting. Make sure to use jQuery methods instead. Note that we no longer need to use the `getSongItem()` helper because we can use jQuery's `find()` method to get the element with `.song-item-number`. Use `this` to refer to the row.

Here's **our implementation** so you can check your solution. Message your mentor for help before referencing our solution.

After adding the new functions, update the `window.onload()` block to use `$(document).ready()` and delete the event handlers that we refactored inside `createSongRow()`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
- window.onload = function() {
+ $(document).ready(function() {
      setCurrentAlbum(albumPicasso);

-     songListContainer.addEventListener('mouseover', function(event) {
-         if (event.target.parentElement.className === 'album-view-song-item') {
-             var songItem = getSongItem(event.target);
-
-             if (songItem.getAttribute('data-song-number') !== currentlyPlayingSong) {
-                 songItem.innerHTML = playButtonTemplate;
-             }
-         }
-     });
      for (var i = 0; i < songRows.length; i++) {
-         songRows[i].addEventListener('mouseleave', function(event) {
-             var songItem = getSongItem(event.target);
-
-             if (songItem.getAttribute('data-song-number') !== currentlyPlayingSong) {
-                 getSongItem(this).innerHTML = getSongItem(this).getAttribute('data-so
-             }
-         });
-
          songRows[i].addEventListener('click', function(event) {
              clickHandler(event.target);
          });
      }
- };
+ });
```

Refactor `clickHandler`

As noted earlier, the `clickHandler` function will no longer need to take any arguments. We no longer need to reference the `targetElement` or use the `getSongItem()` function because we can use `this` to reference the row. Move an empty function to `createSongRow()`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {

      ...

      var $row = $(template);

+     var clickHandler = function() {
+         // clickHandler logic
+     };
+
      var onHover = function(event) {
  ...
```

Attempt to refactor the function using jQuery on your own. Reference **our implementation** for assistance. Message your mentor if you need help before consulting our solution.

Remove Deprecated Code

Remove the code for `findParentByClassName()`, `getSongItem()`, and the previous `clickHandler()`. jQuery renders these methods unnecessary:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var createSongRow = function(songNumber, songName, songLength) {
      ...
  };

- var findParentByClassName = function(element, targetClass) {
-     var currentParent = element.parentElement;
-     while (currentParent.className != targetClass) {
-         currentParent = currentParent.parentElement
-     }
-     return currentParent;
- };
-
- var getSongItem = function(element) {
-     switch (element.className) {
```

```
-         case 'album-song-button':
-         case 'ion-play':
-         case 'ion-pause':
-             return findParentByClassName(element, 'song-item-number');
-         case 'album-view-song-item':
-             return element.querySelector('.song-item-number');
-         case 'song-item-title':
-         case 'song-item-duration':
-             return findParentByClassName(element, 'album-view-song-item').querySelect
-         case 'song-item-number':
-             return element;
-         default:
-             return;
-     }
- };
-
- var clickHandler = function(targetElement) {
-     var songItem = getSongItem(targetElement);
-
-     if (currentlyPlayingSong === null) {
-         songItem.innerHTML = pauseButtonTemplate;
-         currentlyPlayingSong = songItem.getAttribute('data-song-number');
-     } else if (currentlyPlayingSong === songItem.getAttribute('data-song-number')) {
-         songItem.innerHTML = playButtonTemplate;
-         currentlyPlayingSong = null;
-     } else if (currentlyPlayingSong !== songItem.getAttribute('data-song-number')) {
-         var currentlyPlayingSongElement = document.querySelector('[data-song-number='
-         currentlyPlayingSongElement.innerHTML = currentlyPlayingSongElement.getAttrib
-         songItem.innerHTML = pauseButtonTemplate;
-         currentlyPlayingSong = songItem.getAttribute('data-song-number');
-     }
- };
  ...
```

Finally, remove the `click` event listener from the `$(document).ready()` call, and remove the variables that selected the song rows and song list container. The operations in `createSongRow()` replace their functionality:

~/bloc/bloc-jams/scripts/album.js

```
  ...
- var songListContainer = document.getElementsByClassName('album-view-song-list')[0];
- var songRows = document.getElementsByClassName('album-view-song-item');

  var playButtonTemplate = '<a class="album-song-button"><span class="ion-play"></span>
  var pauseButtonTemplate = '<a class="album-song-button"><span class="ion-pause"></spa

  var currentlyPlayingSong = null;

  $(document).ready(function() {
      setCurrentAlbum(albumPicasso);
-     for (var i = 0; i < songRows.length; i++) {
-         songRows[i].addEventListener('click', function(event) {
-             clickHandler(event.target);
-         });
-     }
  });
```

Recap

Concept	Description
`this`	References the object to which it is scoped, depending on the context of that scope.
`.find()`	jQuery's equivalent to `querySelector()`. It can be called on a selected jQuery-wrapped element to find any child that matches the specified selector.
`.click()`	jQuery's event handler for click events.
`.hover()`	jQuery's shorthand for an event that handles both `mouseenter` and `mouseleave` behavior. In the case of event delegation, it uses `mouseover` and `mouseout`.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

18. jQuery: Play/Pause

Assignment | Discussion | Submission

19 jQuery: Next and Previous Buttons

Overview and Purpose

This checkpoint uses jQuery to implement complex JavaScript behavior for moving between songs.

Objectives

After this checkpoint, students should be able to:

- Assess a problem and attempt to devise a solution in pseudo-code or plain English.
- Translate pseudo-code or plain English actions into JavaScript.
- Recognize and use a number of jQuery methods – such as `.html()`, `.text()`, etc.

We've used jQuery selectors to retrieve the song data using methods like `attr()`. As we add features that require us to use the same data in multiple places, finding data via selectors becomes inefficient. We want data in our application to populate the interface, not the other way around. To streamline the source of song and album data, we'll add functions that keep track of this information using JavaScript objects and arrays.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Move Album Data

These album objects at the top of the `album.js` file are taking up a lot of space. They only provide fake data for us to use in the application, so they're essentially a fixture. Fixtures are objects that provide quick and easy ways of loading your application with fake data. So we want to keep these around, but just not in our `album.js` file. Let's give them their own home.

Create a new file called `fixtures.js`:

~/bloc/bloc-jams/

```
$ touch scripts/fixtures.js
```

Move the album objects from `album.js` into the new `fixtures.js` file:

~/bloc/bloc-jams/scripts/fixtures.js

```
+ var albumPicasso = {
+     title: 'The Colors',
+     artist: 'Pablo Picasso',
+     label: 'Cubism',
+     year: '1881',
+     albumArtUrl: 'assets/images/album_covers/01.png',
+     songs: [
+         { title: 'Blue', duration: '4:26' },
+         { title: 'Green', duration: '3:14' },
+         { title: 'Red', duration: '5:01' },
+         { title: 'Pink', duration: '3:21' },
+         { title: 'Magenta', duration: '2:15' }
+     ]
+ };
+
+ var albumMarconi = {
+     title: 'The Telephone',
+     artist: 'Guglielmo Marconi',
+     label: 'EM',
+     year: '1909',
+     albumArtUrl: 'assets/images/album_covers/20.png',
+     songs: [
+         { title: 'Hello, Operator?', duration: '1:01' },
+         { title: 'Ring, ring, ring', duration: '5:01' },
+         { title: 'Fits in your pocket', duration: '3:21' },
+         { title: 'Can you hear me now?', duration: '3:14' },
+         { title: 'Wrong phone number', duration: '2:15' }
+     ]
+ };
```

Delete the albums from the top of `album.js`:

~/bloc/bloc-jams/scripts/album.js

```
- var albumPicasso = {
-     title: 'The Colors',
-     artist: 'Pablo Picasso',
-     label: 'Cubism',
-     year: '1881',
-     albumArtUrl: 'assets/images/album_covers/01.png',
-     songs: [
-         { title: 'Blue', duration: '4:26' },
-         { title: 'Green', duration: '3:14' },
-         { title: 'Red', duration: '5:01' },
-         { title: 'Pink', duration: '3:21' },
-         { title: 'Magenta', duration: '2:15' }
-     ]
- };
-
- var albumMarconi = {
-     title: 'The Telephone',
-     artist: 'Guglielmo Marconi',
-     label: 'EM',
-     year: '1909',
-     albumArtUrl: 'assets/images/album_covers/20.png',
-     songs: [
-         { title: 'Hello, Operator?', duration: '1:01' },
-         { title: 'Ring, ring, ring', duration: '5:01' },
-         { title: 'Fits in your pocket', duration: '3:21' },
-         { title: 'Can you hear me now?', duration: '3:14' },
-         { title: 'Wrong phone number', duration: '2:15' }
-     ]
- };
  ...
```

Include the `fixtures.js` file in a script tag at the bottom of `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
+     <script src="scripts/fixtures.js"></script>
      <script src="scripts/album.js"></script>
  ...
```

Track Current Song Data

Here are the goals of this checkpoint's code:

- We want to use the album data stored in `fixtures.js` to track our current song and album by storing them in variables.
- We want to match the currently playing song's object with its corresponding index in the `songs` array.
- When we call the *next* and *previous* functions in our application, they should increment or decrement the index of the current song in the array, respectively.

Store Current Album and Song Information

> We want to use the album data stored in `fixtures.js` to track our current song and album by storing them in variables.

Thus far, Bloc Jams has used `currentlyPlayingSong` to track information about the music playing. We need to store more data about the currently playing music to track the song properly for *next* and *previous* functionality. We can store the current album information by adding a variable above the `$document.ready()` block called `currentAlbum`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var currentlyPlayingSong = null;
+ var currentAlbum = null;

  $(document).ready(function() {
  ...
```

Then, assign the variable to `album`, the argument in the existing `setCurrentAlbum()` function:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var setCurrentAlbum = function(album) {
+     currentAlbum = album;
      var $albumTitle = $('.album-view-title');
  ...
```

We've used the `currentlyPlayingSong` variable to store the number of the current song, but we'll rename it to `currentlyPlayingSongNumber` to be more explicit. Add the new variable to `album.js`, along with a `currentSongFromAlbum` variable that will hold the currently playing song object from the `songs` array:

~/bloc/bloc-jams/scripts/album.js

```
  ...
- var currentlyPlayingSong = null;
  // #1
  var currentAlbum = null;
+ var currentlyPlayingSongNumber = null;
+ var currentSongFromAlbum = null;

  $(document).ready(function() {
  ...
```

We now have a set of variables in the global scope that hold current song and album information (at **#1**).

Update Methods to Use the New Variables

The `clickHandler()`, `onHover`, and `offHover` methods relied on `currentlyPlayingSong` to determine their behavior. We also need to update the `currentSongFromAlbum` variable when a new song number is established.

Attempt to refactor the `clickHandler()`, `offHover()`, and `onHover()` on your own using the new variables. Consult your mentor for help if you have trouble. Check your work against **our sample implementation** after your attempt.

Use the Song Data to Update the Player Bar

> The player bar information updating when we play a new song.

With the new variables holding the current song information, we can easily share this information with the player bar. Remove our beloved placeholder song from `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
-                     <h2 class="song-name">My dumb song</h2>
+                     <h2 class="song-name"></h2>
  ...
```

~/bloc/bloc-jams/album.html

```
  ...
-                     <h2 class="artist-song-mobile">My dumb song - Fallout Boy</h2>
-                     <h3 class="artist-name">Fallout Boy</h3>
+                     <h2 class="artist-song-mobile"></h2>
+                     <h3 class="artist-name"></h3>
  ...
```

Write a function called `updatePlayerBarSong()` that updates the text of the `<h2>` tags that contain the song name and the artist name. Reference data from the current song variables to populate them. **Try it on your own first.** Message your mentor with questions, and check your answer with **our sample implementation**.

Toggle the Player Bar "Play" Button

> The player bar toggling between play and pause with the individual song rows.

In a music player like Bloc Jams, the player bar should reflect whether a song is playing or paused.

Add two new templates that hold the Ionicon for the play and pause button, so we can easily set the HTML of the player bar when we've played a new song:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var playButtonTemplate = '<a class="album-song-button"><span class="ion-play"></span>
  var pauseButtonTemplate = '<a class="album-song-button"><span class="ion-pause"></spa
+ var playerBarPlayButton = '<span class="ion-play"></span>';
+ var playerBarPauseButton = '<span class="ion-pause"></span>';

  var currentAlbum = null;
  ...
```

And add a line of code to the end of `updatePlayerBarSong()` that updates the HTML of the play/pause button to the content of `playerBarPauseButton`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var updatePlayerBarSong = function() {

      ...

+     $('.main-controls .play-pause').html(playerBarPauseButton);
  };
  ...
```

Finally, add a call to `updatePlayerBarSong()` in the `clickHandler()` conditional when a new song is played, and revert the HTML of the element to the `playerBarPlayButton` template when the song is paused:

~/bloc/bloc-jams/scripts/album.js

```
  ...
          if (currentlyPlayingSongNumber !== songNumber) {
              $(this).html(pauseButtonTemplate);
              currentlyPlayingSongNumber = songNumber;
              currentSongFromAlbum = currentAlbum.songs[songNumber - 1];
+             updatePlayerBarSong();
          } else if (currentlyPlayingSongNumber === songNumber) {
              $(this).html(playButtonTemplate);
+             $('.main-controls .play-pause').html(playerBarPlayButton);
              currentlyPlayingSongNumber = null;
              currentSongFromAlbum = null;
          }
  ...
```

Track the Index of the Current Song

> We want to match the currently playing song's object with its corresponding index in the `songs` array.

Following the goal we set earlier, create a helper method with two arguments, `album` and `song`, that returns the index of a song found in album's `songs` array:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var setCurrentAlbum = function(album) {
      ...
  };
+
+ var trackIndex = function(album, song) {
+     return album.songs.indexOf(song);
+ };
  ...
```

Implement the `nextSong()` Function

> When we call the *next* and *previous* functions in our application, they should increment or decrement the index of the current song in the array, respectively.

Let's expand this requirement and be more specific. The `nextSong()` function should:

- Know what the previous song is. This includes the situation in which the next song is the first song, following the final song in the album (that is, it should "wrap" around).
- Use the `trackIndex()` helper function to get the index of the current song and then increment the value of the index.
- Set a new current song to `currentSongFromAlbum`.
- Update the player bar to show the new song.
- Update the HTML of the previous song's `.song-item-number` element with a number.
- Update the HTML of the new song's `.song-item-number` element with a pause button.

Given the details above, **try to write the `nextSong()` function on your own, first.** Consult your mentor for help if needed. Check your work against **our own implementation**.

Implement the `previousSong()` Function

The implementation of the `previousSong()` function follows the same steps, except that the first detail will be different when switching to a previous song.

Also, after getting the index of the current song in this function, we should *decrement* it, not increment it.

Follow the same procedure as the prior section: **try writing `previousSong()`**, message your mentor with questions, and check it **against our implementation**.

Add Event Handlers

Create variables at the bottom of `album.js` to hold jQuery selectors for the next and previous buttons:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var currentSongFromAlbum = null;
+
+ var $previousButton = $('.main-controls .previous');
+ var $nextButton = $('.main-controls .next');

  $(document).ready(function() {
  ...
```

Add jQuery click event handlers on each respective variable to the `$(document).ready()` block:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  $(document).ready(function() {
      setCurrentAlbum(albumPicasso);
+     $previousButton.click(previousSong);
+     $nextButton.click(nextSong);
  });
```

Enforce Consistent Data Types

> An error occurs from inequality in types between `songNumber` and `currentlyPlayingSongNumber`.

If we hover over a song row after clicking next or previous, we find that the play button switches back to the song number of the row. This behavior occurs because the type of `songNumber` and `currentlyPlayingSongNumber` are different, and the conditional in the `onHover` and `offHover` functions uses **strict equality** to evaluate the conditional statements.

The jQuery `attr()` method retrieves the numbers with a string data type, whereas `currentlyPlayingSongNumber` is either type object or number. Add this line to the `offHover()` function to test the types:

~/bloc/bloc-jams/scripts/album.js

```
  console.log("songNumber type is " + typeof songNumber + "\n and currentlyPlayingSongNu
```

Use the `parseInt()` function to convert all song number references to integers. Wrap every variable assignment involving a song number in a `parseInt()` call. For example, we would refactor the first variable assignment in `clickHandler()` to:

~/bloc/bloc-jams/scripts/album.js

```
var songNumber = parseInt($(this).attr('data-song-number'));
```

Wrap every variable assignment involving a song number in a `parseInt()` call.

Test the `onHover` and `offHover` behavior after, and the next and previous functionality should work as shown below.

Recap

Concept	Description
"Fixtures" data file	A file used for holding static data, often used to supply sample data for an application.
`parseInt()`	One of few JavaScript functions for converting data to numbers.
`indexOf()`	Returns the first index at which a given element can be found in the array, or -1 if it is not present.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

20 jQuery: Buzz Library

Overview and Purpose

This checkpoint introduces the Buzz music library for playing audio files.

Objectives

After this checkpoint, students should be able to:

- Include the Buzz music library in an application.
- Play an audio file using the Buzz API.
- Explain what the `buzz.sound` constructor function returns.
- Use a number of Buzz methods to manage audio file playback – such as `.play()`, `.pause()`, `.stop()`, `isPaused()`, etc.

Bloc Jams has been a silent music player so far, and it's time to change that. We'll use the **Buzz music library** to play and pause music in our application. We've already written many of the functions required to update the interface, so we'll focus more on helper methods that use Buzz to change song playback, volume, and state.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Download mp3s and Add the Buzz Library

Create a subdirectory in `assets` called `music` to hold the files we'll add to Bloc Jams:

~/bloc/bloc-jams/

```
$ mkdir assets/music
```

Download the .zip of the mp3s and move them into `assets/music`. There should be five songs that match the names of the songs in the `albumPicasso` object.

Add an `audioUrl` property to each song in the `songs` array of `albumPicasso`, and set the value to the corresponding path of the mp3:

~/bloc/bloc-jams/scripts/fixtures.js

```
  var albumPicasso = {
      title: 'The Colors',
      artist: 'Pablo Picasso',
      label: 'Cubism',
      year: '1881',
      albumArtUrl: 'assets/images/album_covers/01.png',
      songs: [
-         { title: 'Blue', duration: '4:26' },
-         { title: 'Green', duration: '3:14' },
-         { title: 'Red', duration: '5:01' },
-         { title: 'Pink', duration: '3:21'},
-         { title: 'Magenta', duration: '2:15'}
+         { title: 'Blue', duration: '4:26', audioUrl: 'assets/music/blue' },
+         { title: 'Green', duration: '3:14', audioUrl: 'assets/music/green' },
+         { title: 'Red', duration: '5:01', audioUrl: 'assets/music/red' },
+         { title: 'Pink', duration: '3:21', audioUrl: 'assets/music/pink' },
+         { title: 'Magenta', duration: '2:15', audioUrl: 'assets/music/magenta' }
      ]
  };
  ...
```

> Note that we don't add the `.mp3` extension to the end of the audio files. We'll specify the file type when we use the files with Buzz.

Add Buzz library's CDN link to `album.html`:

~/bloc/bloc-jams/album.html

```
  ...
      <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
+     <script src="https://cdnjs.cloudflare.com/ajax/libs/buzz/1.1.10/buzz.min.js"></sc
      <script src="scripts/fixtures.js"></script>
      <script src="scripts/album.js"></script>
  ...
```

> We put the script reference before `fixtures.js` and `album.js` because the Buzz functionality needs to load before the code in which reference it.

Wrap Song in a Buzz Object

To use Buzz's API on audio files, we need to wrap an audio file in the `buzz.sound` constructor function. The function returns a Buzz `sound` object, which is instantiated using the `new` keyword. It requires at least one argument, a link to an audio file (or array of audio files), but also takes an optional settings object.

Before we use one of our songs in a `buzz.sound` call, create a global variable called `currentSoundFile`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var currentlyPlayingSongNumber = null;
  var currentSongFromAlbum = null;
+ var currentSoundFile = null;
  ...
```

We'll store the `sound` object in this variable when we set a new current song. The `setSong()` function handles the assignment of the current song, so we should create the `sound` object there and assign it to `currentSoundFile`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
  var setSong = function(songNumber) {
      currentlyPlayingSongNumber = parseInt(songNumber);
      currentSongFromAlbum = currentAlbum.songs[songNumber - 1];
      // #1
+     currentSoundFile = new buzz.sound(currentSongFromAlbum.audioUrl, {
          // #2
+         formats: [ 'mp3' ],
+         preload: true
+     });
  };
  ...
```

At **#1**, we assign a new Buzz `sound` object. We've passed the audio file via the `audioUrl` property on the `currentSongFromAlbum` object.

At **#2**, we've passed in a settings object that has two properties defined, `formats` and

`preload`. `formats` is an array of strings with acceptable audio formats. We've only included the `'mp3'` string because all of our songs are mp3s. Setting the `preload` property to `true` tells Buzz that we want the mp3s loaded as soon as the page loads.

Play Music Using Existing Functions

We want to implement play and pause behavior with music. To do so, we need to redefine what our application has understood "pause" to mean so far.

Before adding music, we couldn't *actually* pause songs because pausing a song means *freezing the song at a specific time*. If we don't have music files to play, then no time elapses to establish a moment where we can pause them.

With the addition of music, we now have three states:

State	Description
Stopped	The *stopped* state means that `currentlyPlayingSongNumber`, `currentSongFromAlbum`, and `currentSoundFile` are all `null`. It only exists before we've clicked a song.
Playing	*Playing* means that `currentlyPlayingSongNumber`, `currentSongFromAlbum`, and `currentSoundFile` are defined based on the song whose play button was clicked.
Paused	In the *paused* state, the current time elapsed in the song is greater than zero, and `currentlyPlayingSongNumber`, `currentSongFromAlbum`, and `currentSoundFile` are all defined. It may only exist after we've played a song.

Refactor `clickHandler()`

We need to refactor `clickHandler` to reflect this new playback approach. That means:

- In the second conditional statement, when the user clicks a song that is not the currently playing song, we need to play the `currentSoundFile` after calling `setSong()`.
- In the third conditional statement, when the user clicks the pause button for the same song that is playing, we need to get rid of the logic that sets the `currentlyPlayingSongNumber` and `currentSongFromAlbum` to `null`. We should replace it with a conditional statement that checks if the `currentSoundFile` is paused:

- If it is, we need to start playing the song again and revert the icon in the song row and the player bar to the pause button.
- If it isn't paused, we need to pause it and set the content of the song number cell and player bar's pause button back to the play button.

Buzz has `play()` and `pause()` methods that will play and pause audio files. For the third conditional in `clickHandler()`, we'll use Buzz's `isPaused()` method on `currentSoundFile` to check if the song is paused or not. **Try implementing the new `clickHandler()` behavior on your own based on the requirements we've defined above.**

Reference the **Methods section of the Buzz documentation** for clarification on the `buzz.sound` APIs. Message your mentor with any questions, and compare your answer to **our implementation**.

Prevent Multiple Songs From Playing Concurrently

We need to make an addition to the `setSong()` function to prevent concurrent playback. If we click to play a different song before a the current song is finished, we need to stop the current song before we set a new one. Add a conditional statement to the beginning of `setSong()` that checks for a defined `currentSoundFile` and then runs `currentSoundFile.stop()` if `true`:

~/bloc/bloc-jams/scripts/album.js

```
   ...
   var setSong = function(songNumber) {
+      if (currentSoundFile) {
+          currentSoundFile.stop();
+      }
+
       currentlyPlayingSongNumber = parseInt(songNumber);

       ...

   };
   ...
```

Play Songs When Skipping

Add `currentSoundFile.play()` to `nextSong()` and `previousSong()` after the `setSong()` call:

~/bloc/bloc-jams/scripts/album.js

```
   ...
   var previousSong = function() {
       ...
       setSong(currentSongIndex + 1);
+      currentSoundFile.play();
       ...
   };

   var nextSong = function() {
       ...
       setSong(currentSongIndex + 1);
+      currentSoundFile.play();
       ...
   };
   ...
```

Set Song Volume

Buzz `sound` objects also have methods for handling volume that are based on a scale from 0-100, with 100 as the maximum volume. Create a `currentVolume` variable and set its initial value to 80:

~/bloc/bloc-jams/scripts/album.js

```
   ...
   var currentSongFromAlbum = null;
   var currentSoundFile = null;
+  var currentVolume = 80;
   ...
```

Add a `setVolume()` function that takes one argument, a volume value, and wraps the Buzz `setVolume()` method with a conditional statement that checks to see if a `currentSoundFile` exists:

~/bloc/bloc-jams/scripts/album.js

```
   ...
   var setSong = function(songNumber) {

       ...

       currentSoundFile = new buzz.sound(currentSongFromAlbum.audioUrl, {
           formats: [ 'mp3' ],
           preload: true
       });
+
+      setVolume(currentVolume);
   };
+
+  var setVolume = function(volume) {
+      if (currentSoundFile) {
+          currentSoundFile.setVolume(volume);
+      }
+  };
   ...
```

Click any song's play button to enjoy some killer tunes.

Recap

Concept	Description
Buzz	A JavaScript audio library which uses the HTML5 audio element API.
Buzz `sound` **Object**	A constructor function that wraps audio files and provides methods and events for enabling, tracking, and manipulating them.
Buzz `setVolume()` **Method**	Set the volume of the sound. The range is 0-100.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

20. jQuery: Buzz Library

Assignment | Discussion | Submission

21 jQuery: Seek Bars

Overview and Purpose

This checkpoint implements code to adjust the song progress and volume seek bars.

Objectives

After this checkpoint, students should be able to:

- Discuss different mouse fire events.
- Explain what `Math.min()` and `Math.max()` do.
- Discuss jQuery event properties such as `pageX` and methods such as `offset()`.

With playable music in our application, it's time to program usable seek bars to adjust the current time of the song and its volume. We'll develop seek bars that update as the song plays, and respond to mouse events that click or drag the thumb to a specific location.

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Get Song Durations in Seconds

To implement functional seek bars, we need to have the song duration for each of our album tracks in seconds instead of fake strings. Replace the strings stored in each `length`

property with the following values:

~/bloc/bloc-jams/scripts/fixtures.js

```
  var albumPicasso = {
      title: 'The Colors',
      artist: 'Pablo Picasso',
      label: 'Cubism',
      year: '1881',
      albumArtUrl: 'assets/images/album_covers/01.png',
      songs: [
-         { title: 'Blue', duration: '4:26', audioUrl: 'assets/music/blue' },
-         { title: 'Green', duration: '3:14', audioUrl: 'assets/music/green' },
-         { title: 'Red', duration: '5:01', audioUrl: 'assets/music/red' },
-         { title: 'Pink', duration: '3:21', audioUrl: 'assets/music/pink' },
-         { title: 'Magenta', duration: '2:15', audioUrl: 'assets/music/magenta' }
+         { title: 'Blue', duration: 161.71, audioUrl: 'assets/music/blue' },
+         { title: 'Green', duration: 103.96, audioUrl: 'assets/music/green' },
+         { title: 'Red', duration: 268.45, audioUrl: 'assets/music/red' },
+         { title: 'Pink', duration: 153.14, audioUrl: 'assets/music/pink' },
+         { title: 'Magenta', duration: 374.22, audioUrl: 'assets/music/magenta' }
      ]
  };
  ...
```

A Generic Method for Updating Seek Bars

Before writing any code, consider the functionality required of a generic method to update any seek bar:

- The function must take two arguments, one for the seek bar to alter (either the volume or audio playback controls) and one for the ratio that will determine the `width` and `left` values of the `.fill` and `.thumb` classes, respectively.
- The ratio must be converted to a percentage so we can set the CSS property values as percents.
- The percentage must be passed into jQuery functions that set the `width` and `left` CSS properties.

With those constraints in mind, add the following code to `album.js` below the `setCurrentAlbum()` function:

~/bloc/bloc-jams/scripts/album.js

```
   ...
+  var updateSeekPercentage = function($seekBar, seekBarFillRatio) {
+     var offsetXPercent = seekBarFillRatio * 100;
       // #1
+     offsetXPercent = Math.max(0, offsetXPercent);
+     offsetXPercent = Math.min(100, offsetXPercent);
+
       // #2
+     var percentageString = offsetXPercent + '%';
+     $seekBar.find('.fill').width(percentageString);
+     $seekBar.find('.thumb').css({left: percentageString});
+  };
   ...
```

We start by multiplying the ratio by 100 to determine a percentage. At **#1**, we use the built-in JavaScript `Math.max()` function to make sure our percentage isn't less than zero and the `Math.min()` function to make sure it doesn't exceed 100.

> While we expect you to give 110% effort to the Bloc checkpoints, giving 110% to determine the position of our seek bar causes weird visual bugs and ugly errors.

At **#2**, we convert our percentage to a string and add the `%` character. When we set the `width` of the `.fill` class and the `left` value of the `.thumb` class, the CSS interprets the value as a percent instead of a unit-less number between 0 and 100.

Configure the Seek Bars

`updateSeekPercentage()` is useless until we have a method for determining the `seekBarFillRatio`. Create a function called `setupSeekBars()` below `updateSeekPercentage()` and consider the following code which uses a `click` event to determine the fill width and thumb location of the seek bar:

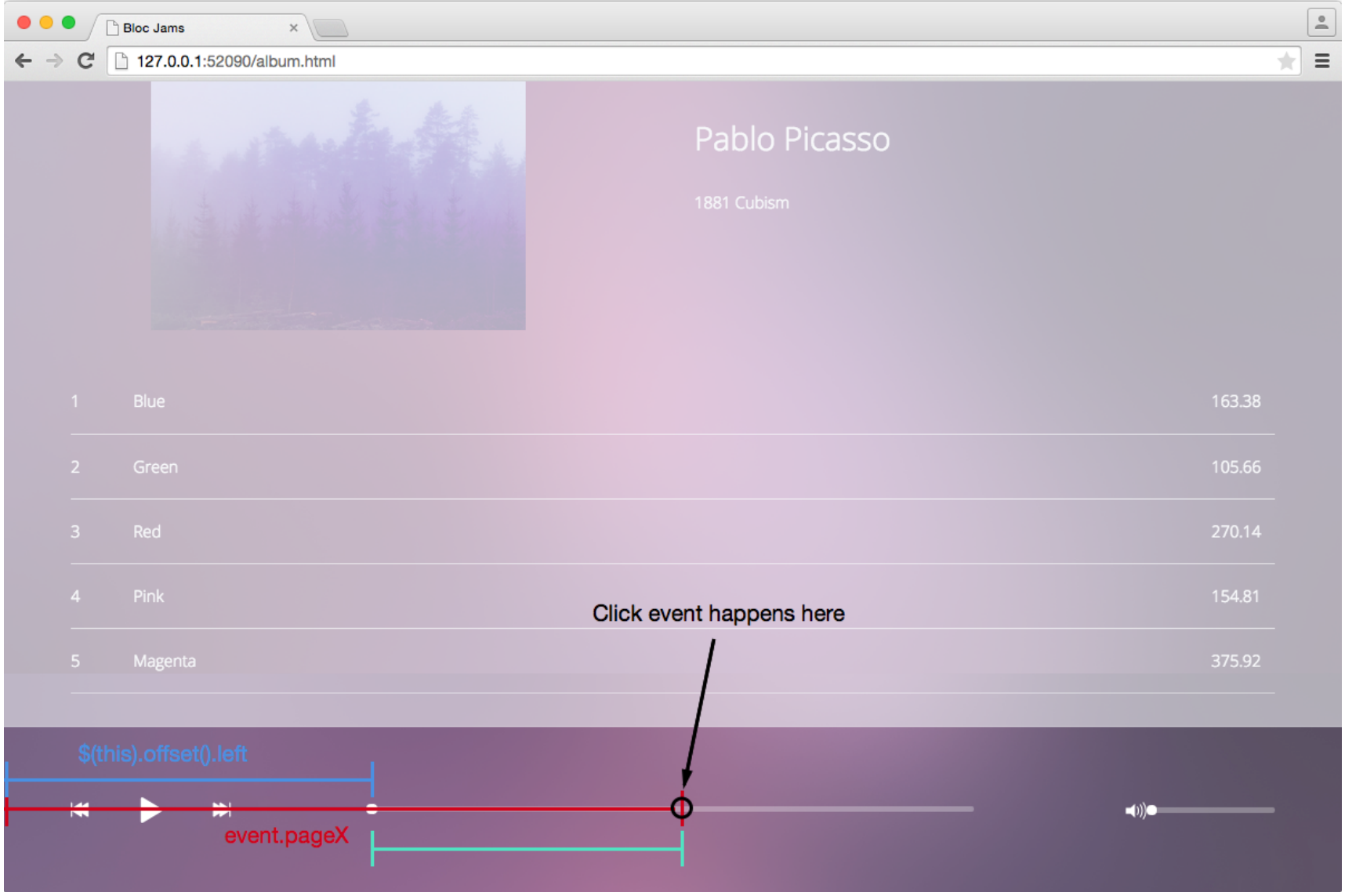
~/bloc/bloc-jams/scripts/album.js

```
   ...
+  var setupSeekBars = function() {
+      var $seekBars = $('.player-bar .seek-bar');
+
+      $seekBars.click(function(event) {
           // #3
+          var offsetX = event.pageX - $(this).offset().left;
+          var barWidth = $(this).width();
           // #4
+          var seekBarFillRatio = offsetX / barWidth;
+
           // #5
+          updateSeekPercentage($(this), seekBarFillRatio);
+      });
+  };
   ...
```

We emphasized the importance of making our functions work for any seek bar, and we're satisfying that goal by selecting either seek bar with our `$seekBars` selector. The seek bar that updates will be determined by the target of the event.

At **#3**, we see a new property on the `event` object called `pageX`. This is a jQuery-specific event value, which holds the X (or horizontal) coordinate at which the event occurred (think of the X-Y coordinate plane that you hated in Algebra class).

We subtract the `offset()` of the seek bar held in `$(this)` from the left side. This may be difficult to visualize, so we've created a diagram below:



The diagram illustrates how subtracting `$(this).offset().left` (the blue line) from the `event.pageX` value (the red line) leaves us with a resulting value that is a proportion of the seek bar (the green).

At **#4**, we divide `offsetX` by the width of the entire bar to calculate `seekBarFillRatio`.

Finally, at **#5**, we pass `$(this)` as the `$seekBar` argument and `seekBarFillRatio` for its eponymous argument to `updateSeekBarPercentage()`.

Add `setupSeekBars()` to `$(document).ready()` and click the seek bars to test the new functionality.

~/bloc/bloc-jams/scripts/album.js

```
  ...
  $(document).ready(function() {
     setCurrentAlbum(albumPicasso);
+    setupSeekBars();
  ...
```

> Seek bars with click functionality.

Drag Thumb Position Using Mouse Events

Dragging the seek bar's thumb involves three new events:

1. `mousedown` which fires when the user presses their mouse down
2. `mousemove` which fires anytime the mouse moves, regardless of which event precedes it
3. `mouseup` which fires when the user releases their hold on the mouse

Add the following code below the `click()` event:

~/bloc/bloc-jams/scripts/album.js

```
   var setupSeekBars = function() {
      // #6
       var $seekBars = $('.player-bar .seek-bar');

       $seekBars.click(function(event) {
          ...
       });
       // #7
+      $seekBars.find('.thumb').mousedown(function(event) {
           // #8
+          var $seekBar = $(this).parent();
+
           // #9
+          $(document).bind('mousemove.thumb', function(event){
+              var offsetX = event.pageX - $seekBar.offset().left;
+              var barWidth = $seekBar.width();
+              var seekBarFillRatio = offsetX / barWidth;
+
+              updateSeekPercentage($seekBar, seekBarFillRatio);
+          });
+
           // #10
+          $(document).bind('mouseup.thumb', function() {
+              $(document).unbind('mousemove.thumb');
+              $(document).unbind('mouseup.thumb');
+          });
+      });
   };
```

At **#6**, we are using jQuery to find all elements in the DOM with a class of `"seek-bar"` that are contained within the element with a class of `"player-bar"`. This will return a jQuery wrapped array containing both the song seek control and the volume control.

At **#7**, we `find` elements with a class of `.thumb` inside our `$seekBars` and add an event listener for the `mousedown` event. A `click` event fires when a mouse is pressed and released quickly, but the `mousedown` event will fire as soon as the mouse button is pressed down. In contrast to this, the `mouseup` event is the opposite: it fires when the mouse button is released. jQuery allows us access to a shorthand method of attaching the `mousedown` event by calling `mousedown` on a jQuery collection.

At **#8**, we are taking the context of the event and wrapping it in jQuery. In this scenario, `this` will be equal to the `.thumb` node that was clicked. Because we are attaching an event to both the song seek and volume control, this is an important way for us to determine which of these nodes dispatched the event. We can then use the `parent` method, which will select the immediate parent of the node. This will be whichever seek bar this `.thumb` belongs to.

#9 introduces a new way to track events, **jQuery's `bind()` event**. `bind()` behaves similarly to `addEventListener()` in that it takes a string of an event instead of wrapping the event in a method like we've seen with all other jQuery events thus far. We use `bind()` because it allows us to **namespace** event listeners (we'll discuss namespacing, shortly). The event handler inside the `bind()` call is identical to the `click` behavior.

We've attached the `mousemove` event to `$(document)` to make sure that we can drag the thumb after mousing down, even when the mouse leaves the seek bar. This allows for a smoother experience for seeking to a song position. If we didn't take this approach, and instead attached the event handler directly to `$seekBar`, we would see the behavior shown in this video:

Notice that we can't drag the thumb outside of the seek bar element, which creates a very narrow interface for adjusting song position. However, when we attach the `mousemove` event to `$(document)`, we can continue to drag the thumb as long as the mouse remains down:

The event handler on `$(document)` is the reason we need to use `bind()` with namespacing. In this case, namespacing is a technique to make the event more specific by attaching a string to it after a period. This way, if we ever attach another event listener for the `mousemove` event, the seek bar would only move if we also included the `.thumb` string.

> The resemblance to the notation for a `.thumb` CSS selector is incidental. All jQuery event namespaces are offset with a period and followed by a string. We could have called it `.thumbOnSeekBar`, and it would still function the same way.

Finally, at **#10**, we bind the `mouseup` event with a `.thumb` namespace. The event handler uses the `unbind()` event method, which removes the previous event listeners that we just added. If we fail to `unbind()` them, the thumb and fill would continue to move even after the user released the mouse. Comment out this block to demonstrate the unintended behavior.

Refactoring the Transition

Dragging the thumb works, but it looks visually awkward because the thumb delays when it seeks:

The culprit is the `transition` property we set on the `.seek-bar .thumb` selector in `player_bar.css`, which animates all properties of the thumb for 100ms. To fix this, we'll limit transitions to the properties which the `:hover` selector manipulates:

~/bloc/bloc-jams/styles/player_bar.css

```
  ...
  .seek-bar .thumb {
-     transition: all 100ms ease-in-out;
+     transition: width 100ms ease-in-out,
+                 height 100ms ease-in-out,
+                 margin-top 100ms ease-in-out,
+                 margin-left 100ms ease-in-out;
  }
  ...
```

`transition` can take multiple arguments that define transitions on specific properties. Now we only apply the transition to the `width`, `height`, `margin-top`, and `margin-left` properties. This removes the transition on the `left` property that caused the unwanted animation.

Check the result and the thumb should animate more cleanly:

Update the Seek Bar When a Song Plays

The interface for the seek bars works, but it doesn't affect the song position or volume. To address that need, add a function called `updateSeekBarWhileSongPlays()` with the following code above `updateSeekPercentage()`. Consider its implementation below:

~/bloc/bloc-jams/scripts/album.js

```
  ...
+ var updateSeekBarWhileSongPlays = function() {
+     if (currentSoundFile) {
          // #10
+         currentSoundFile.bind('timeupdate', function(event) {
              // #11
+             var seekBarFillRatio = this.getTime() / this.getDuration();
+             var $seekBar = $('.seek-control .seek-bar');
+
+             updateSeekPercentage($seekBar, seekBarFillRatio);
+         });
+     }
+ };

  var updateSeekPercentage = function($seekBar, seekBarFillRatio) {
  ...
```

At **#10**, we `bind()` the `timeupdate` event to `currentSoundFile`. `timeupdate` is a custom Buzz event that fires repeatedly while time elapses during song playback.

At **#11**, we use a new method for calculating the `seekBarFillRatio`. We use Buzz's

`getTime()` method to get the current time of the song and the `getDuration()` method for getting the total length of the song. Both values return time in seconds.

> Find the documentation for `getTime()` and `getDuration()` on the Buzz **sound API page**.

We need to trigger this method whenever a song plays, so **add `updateSeekBarWhileSongPlays()` to `clickHandler()`, `nextSong()`, and `previousSong()` after the `currentSoundFile.play()` call.**

Play a song and the seek bar moves with each passing second.

Seek to Parts of a Song

The seek bar updates while a song plays, but clicking a new location doesn't seek to the corresponding position in the song. We need to create a method that can change the current song's playback location. Name it `seek()` and add it to `album.js` above `setVolume()`:

~/bloc/bloc-jams/scripts/album.js

```
  ...
+ var seek = function(time) {
+     if (currentSoundFile) {
+         currentSoundFile.setTime(time);
+     }
+ }
  ...
```

`seek()` uses the Buzz `setTime()` method to change the position in a song to a specified time.

Differentiate Behavior Based on the Seek Bar

`setupSeekBars()` configures the behavior for both the volume and playback seek bars. Add a conditional statement to each event handler with the following behavior:

- Checks the class of the seek bar's parent to determine whether the current seek bar is changing the volume or seeking to a song position
  - If it's the playback seek bar, seek to the position of the song determined by the `seekBarFillRatio`
  - Otherwise, set the volume based on the `seekBarFillRatio`

Add the code that meets the requirements to `setupSeekBars()`. Message your mentor for help if you find yourself struggling with it. Compare your answer with **our implementation**.

Setting Volume on Initial Song Play

Starting a song for the first time sets a volume, but doesn't update fill and thumb of the volume seek bar. **Update** `clickHandler()` **to set the CSS of the volume seek bar to equal the** `currentVolume`**.** Message your mentor for help and compare it against **our solution**.

Congratulations!

After completing the assignment below, you will have completed the Foundation Phase for Frontend Web Development. Take a moment and reflect on how far you've come: you've learned how to structure and style a webpage with HTML and CSS; you learned about fundamental programming concepts using JavaScript; you've applied that JavaScript knowledge by dynamically altering the DOM; and you've conquered your first libraries in jQuery and Buzz. You are now well positioned to proceed to the Project Phase of the program and build some amazing things. You should be proud of how much you've learned, and also embrace the challenge ahead. Maintain the good habits you've developed through the Foundation Phase, and you **will** succeed in the Project Phase.

And again, we offer you a sincere congratulations:

Recap

Concept	Description

`Math.min()` and `Math.max()`	Each function takes two numbers as arguments and returns the minimum or maximum between the two.
`pageX`	A property on jQuery events that stores the location of the X coordinate where an event occurs in the browser window. `pageX` may be a **native browser-event property in the future**, but for now, it's only a property on jQuery-wrapped events.
`offset()`	A method for finding the distance between an element and the edge of the browser window.
`mousedown`, `mousemove`, and `mouseup`	Mouse events that track when the user has pressed the mouse down, moved the mouse, or released the mouse, respectively.
`bind()`	A jQuery and Buzz method that attaches behavior to an event. Allows namespacing.
`timeupdate`	A Buzz event that fires repeatedly during audio playback.
Namespacing	A technique for making a string or name more specific by adding a prefix or suffix. Used in Bloc Jams to more specifically identify events bound to the `document`, like `mousemove` and `mouseup`.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

How would you rate this checkpoint and assignment? ☺ ☹

Assignment | Discussion | Submission