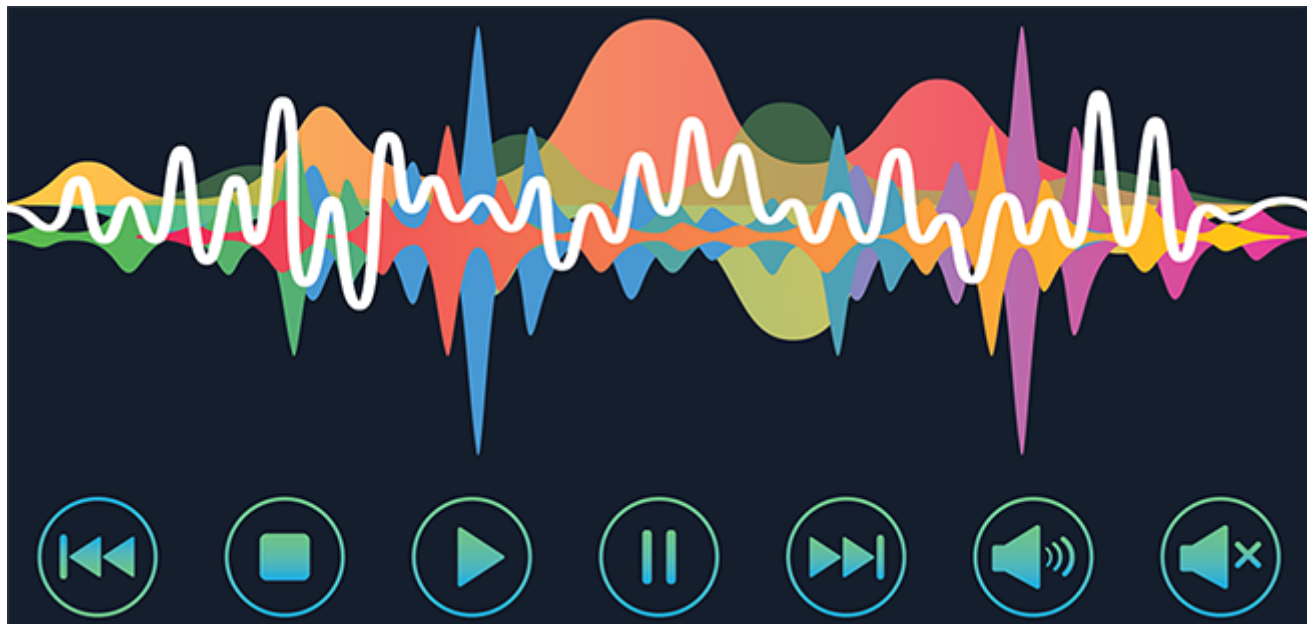


Introduction

Welcome Back!

You might already be familiar with how HTML5 handles audio and video. In HTML, you use `<audio>` and `<video>` tags to play sounds and videos on the page without needing plug-ins like Adobe Flash Player, Apple QuickTime, or Windows Media Player. But you can also use JavaScript to control sound and video.

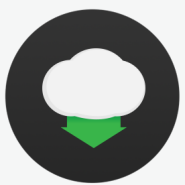


In fact, JavaScript gives you much more freedom and power to use these elements—far beyond simply displaying a player bar for a single sound. With JavaScript, you can add sound effects or background music and create player controls.

As you can imagine, these capabilities are useful in sites that are about music or sound. They also provide a way to develop your JavaScript programming skills. In this lesson, you'll learn how to control the sound capabilities of modern browsers directly from JavaScript.

Fun with Music

Today, we're going to add some sound and music files to a webpage using JavaScript.



Download

Click the link below to download the ZIP file that contains the resources you'll need for this lesson. Included in it are the following files:

- **SoundFX.html:** This is a standard webpage that you'll use to code some sound effects.
- **BackgroundMusic.html:** This new webpage already has code, but you'll be adding some additional controls.
- **Sounds:** We have assembled a number of public domain sample sound files and a speaker icon that we'll be using throughout this lesson.



[Download Lesson-06 ClassProject.zip](#)

1 MB.(Gigabytes) ZIP

Let's get started.

Chapter 1: Sound and JavaScript

Sound Compression

In this chapter, we'll start with the basics of using sound in webpages. There are many formats for sound files. The filename extension on a sound file indicates its format. Some common formats include wave (WAV), AAC, AIFF, AU, MP3, and MP4.

Most formats are simply different ways to compress the sound file to produce a smaller file without compromising its quality. Some of those compression techniques are patented. Ideally, the public Internet should be all about public domain techniques that aren't patented to ensure that patent-infringement lawsuits don't start flying once the format is used widely.

These days, the two audio formats that are free of any patent encumbrances are MP3 and OGG. All HTML5-compatible browsers support at least one of these formats. Virtually all support MP3, and many sites offer only MP3. But it's not all that unusual to see developers offering both, just to ensure the most compatibility across all the many brands and versions of web browser out there on computers, tablets, and phones.

In the interest of being thorough, this lesson will present the safest method of offering two of each sound file, one in MP3 format and exactly the same sound file in OGG format. Be aware that the user, the person hearing the sound, doesn't have to choose one version or the other themselves. Code built into the web browser automatically chooses the best version for itself.



Copyright and Permission

Regardless of whether the format used to store a sound file is patented, the actual content (the song or other sound in the file) might be protected by copyright. For example, any song you can buy on iTunes or Amazon or similar sites is probably copyrighted. You can't distribute or give away such files from your own site without the express written permission of the copyright holder. However, there are some *public domain* sound files you can use freely, including those that we've listed in the Lesson 6 Resources for Further Learning.

So that's the background. Now let's talk about the code.

Detecting and Playing Sound

Before you add any sound, there are a few important things you need to learn for all kinds of JavaScript-controlled audio playback. To lay the groundwork, there are three key capabilities you need to start with:

- ✓ See if the browser supports HTML5 audio.
- ✓ If it does, define a player for playing audio files.
- ✓ Then, determine whether the browser can play MP3 sound files or OGG sound files.

Let's walk through each of these.

Detecting Sound Compatibility

First, you'll need to detect whether the user's browser can handle HTML5 audio. You'd do this using *object detection* to determine whether or not the user's browser is capable of handling the code you intend to use. Fortunately, it's pretty easy. The code syntax is:

```
if (window.HTMLAudioElement) {  
    
}
```

That's it!

The `window.HTMLAudioElement` property returns *true* if the browser can handle HTML5 audio. Any code inside the curly braces of the *if {}* will be executed if (and only if) the current browser supports HTML5 audio. If the browser doesn't support HTML5 audio, `window.HTMLAudioElement` returns *false*, and code inside the curly braces is ignored.

Creating the Player

If you want to be able to control sound purely through JavaScript (with no visible player on the screen), the page will need an *HTML5 audio object*.

This is basically an audio player that doesn't appear on the screen. In this case, your code controls the sound rather than the person viewing the page. Creating an audio object is easy. The syntax is:

```
var name = document.createElement('audio');
```

All you need to do is replace *name* with a variable name of your choosing.



Remember

Like with all JavaScript names, just make sure that it follows these guidelines:

- It starts with a letter
- It doesn't contain blank spaces.
- It doesn't contain punctuation.
- It's case-sensitive through the code.

You can put the object detection code and the code to create the player right in the head of the page. We'll do that next.

Here are the Steps

1. Open *SoundFX.html* in your editor.
2. Add your `<script>...</script>` tags below the `<title>SoundFX</title>` line but above the `</head>` tag.
3. Now type or copy and paste the following comments in between your script tags:

```
// This code runs when page first opens  
// If the browser supports HTML5 audio, it creates a player
```

4. Below the comments, add the sound detection code:

```
if (window.HTMLAudioElement) {  
    
}
```

5. Then, between the curly braces for the if statement, add your invisible player. For simplicity, let's use the name *player* because it's a short, descriptive name that's easy to remember.


```
var player = document.createElement('audio');
```

6. Save your progress.

This code says, "If this browser supports HTML5 audio, create an HTML5 audio player and name it *player*." If the current browser doesn't support HTML5 audio, then the code is ignored, and the page won't attempt to create the audio player.

Determine Which File Type the Browser can Play

We haven't told the player yet what sound to play yet, so that code won't make any sound. It just creates a player that we can control with more JavaScript code. The next step is to determine which types of audio files the browser can play. Fortunately, there's a built-in JavaScript property for that.

Once you've defined an audio player in your code, you can use the *.canPlayType* property of the *audio* object to determine which type of file the browser can play. Since MP3 and OGG seem destined to be the two mainstream audio formats now, we can test for both types. The basic syntax is:

```
name.canPlayType('mimetype').
```

To make the code work, you need to:

- Replace *name* with the name you gave your player.
- Replace *mimetype* with the file type for which you're testing.
 - **To test for MP3**, use *audio/mpeg* (because MP3 is the audio side of the larger mpeg format).
 - **To test for OGG**, you have to use the more complex *audio/ogg; codecs="vorbis"*. (The longer syntax with the *codecs=* is recommended for OGG because the video side of OGG uses a *theora codec*, and we specifically want the *vorbis codec* for OGG audio.)



codec

The word *codec* is short for compressor/decompressor. It's the component of the format that indicates the algorithm used to compress the file. The same algorithm decompresses the file for playback.

The *.canPlayType* property returns a string value:

- It returns `""` (a zero-length string) if the browser can't play that file type.
- It returns *maybe* or *probably*, if the browser can play that file type.

We usually test for compatibility using code that says "If `.canPlayType(mimetype)` does *not* return a zero-length string, then use that mime type."

In JavaScript code, the basic syntax for testing for MP3 would be an *if* block like this:

```
if (name.canPlayType('audio/mpeg').length > 0) {  
    //Code for MP3 sound file goes here  
}
```

Similarly, the test for OGG compatibility looks like this:

```
if (name.canPlayType('audio/ogg; codecs="vorbis"').length > 0) {  
    //Code for ogg sound file goes here  
}
```

In both examples, you'll replace *name* with the name of the audio object (which would be *player* in our example). You'll then add code for setting the source of the audio object to the appropriate sound file for that mime type under the comment inside the curly braces.

Let's add this last bit of code before we move on.

Here are the Steps

1. Open *SoundFX.html* in your editor.
2. Put the cursor after the line that reads *var player = document.createElement('audio');* but above the closing *}* for the *if* statement.

3. Type or copy and paste the following comment:

```
// Choose sound file to play based on browser capabilities
```

4. Now, add the code to test for MP3 below the comment:

```
if (player.canPlayType('audio/mpeg').length > 0) {  
    //Code for mp3 sound file goes here  
}
```

5. Below that, add the code to test for OGG:

```
if (player.canPlayType('audio/ogg; codecs="vorbis"').length > 0) {  
    //Code for ogg sound file goes here  
}
```

6. Save your progress.

You still don't have a sound file, so no sound will play. But don't worry, we'll show you how to specify and play a sound file next. It requires yet another new JavaScript property.

Chapter 2: Adding Sound Files

Using the setAttribute Method

How the setAttribute Method Works

To continue from where we left off, you'll need to learn about another JavaScript method named *setAttribute*. As its name implies, it lets you set the attribute of any HTML element.



Remember

As you may recall, HTML attributes are things inside HTML tags that provide information about that tag. Attributes always use the syntax *name="value"* where *name* is the built-in name of the attribute (you don't get to make up your own). The *value* is the value you assign to the attribute.

For example, consider this HTML tag:

```
<a href="http://www.google.com" target="_blank">
Google
</a>
```

In that tag, *href* is an attribute, and *http://www.google.com* is the value assigned to that attribute. The word *target* is also an attribute, and its value is *_blank*.

Here's another example, using a simple HTML *img* tag for displaying a picture:

```

```

Here, *src* is an attribute, and *mypic.png* is the value assigned to that attribute. The *alt* is also an attribute, and its value is *picture*.

So if you've been writing code in HTML, you've been using attributes and values all along, even if you weren't aware of it. The `setAttribute` method lets you set the value of any attribute in any tag via JavaScript. The syntax is:

```
element.setAttribute(attributeName, value)
```

In your own code, you need to replace the following placeholders:

- ***element***: The element (tag) for which you want to set or change the attribute. Typically this is a variable for a specific element that's already been identified with a `getElementById()` attribute.
- ***attributename***: The attribute to which you want to assign a value. It must be a valid attribute for the tag type. For example, an `href` attribute for an `<a>` tag or `src` attribute for an `` tag.
- ***value***: The value you want to assign to the attribute.

In terms of sound, the `setAttribute` property will come in handy because when you use HTML5 audio tags (rather than JavaScript) to play sound, you can specify the sound file to play using this syntax:

```
<audio src="soundfile">
</audio>
```

You replace *soundfile* with the relative or absolute path to the sound file you want to play. That tag syntax is important because any attribute you can set in a tag, you can also set using `setAttribute` in JavaScript.



Note

This means we can use *setAttribute* to tell our JavaScript audio player what sound file to play.

Adding Sound Files with `setAttribute`

What sound file you play is, of course, up to you. But for the sake of our example, we'll use the *whoosh* file from the *Sounds* file you added to your *Intro JavaScript* folder. You'll see that we've put in two of every sound file; one with an MP3 extension and one with an OGG extension. That's to get maximum support across browsers.

Since you put that *Sounds* folder in your Intro JavaScript folder, the relative path to any sound file in that subfolder is *sounds/filename* where *filename* is the exact filename (including the extension) of the sound file.

In Chapter 1, we left off with some code that could decide whether the browser can play MP3 or OGG files. Now we'll use *setAttribute* in conjunction with that kind of logic to load either an MP3 or OGG file into the player.

Here are the Steps

1. Open *SoundFX.html* in your editor.
2. Put the cursor at the end of the comment line that reads *//Code for mp3 sound file goes here.* but above the closing *}* for the *if* statement and press **ENTER**.
3. Type or copy and paste this code on the new line you created:

```
player.setAttribute('src', 'sounds/whoosh.mp3');
```

4. Now, place the cursor at the end of the comment line that reads *//Code for ogg sound file goes here.* but above the closing *}* for the *if* statement and press **ENTER**.
5. Type or copy and paste this code on the new line you created:

```
player.setAttribute('src', 'sounds/whoosh.ogg');
```

6. Save your progress.



Note

Notice the logic of the code:

- The first *if {}* says, "If this browser can play audio or mpeg files, then set the src= attribute of the audio player to the sound named *whoosh.mp3*."
- The second *if {}* says, "If this browser can play OGG (vorbis) files, then set the src= attribute of the player to the *whoosh.ogg* file."

So the JavaScript code makes a decision about which sound file to use based on the browser's capabilities.

This JavaScript code goes between the `<head> . . . </head>` tags of the page. It doesn't go inside a JavaScript function. So it executes as soon as the page opens in a browser. However, the code still doesn't actually play the sound. If you opened the page in a browser right now, you'd hear nothing. This is because, so far, we've only created the player object and specified a sound file. We haven't actually told the player to play the sound.

Sound Effects and Music

Playing Sound Effects

What you've learned so far can be used in a number of ways, as you'll see. One the simplest is to play the sound in response to some event, such as the click of a button or other activity on the page. We'll use the click of a button as an example, since that would probably be most common.

Here are the Steps

1. Open *SoundFX.html* in your editor.
2. First, we'll need a button, so put your cursor between the `<body>` and `</body>` tags.
3. Type or copy and paste the following code:

```
<button onclick="playsound()">Woosh</button>
```



This puts a visible button on the page.

When clicked, the button calls a JavaScript function named *playsound()*. Or, it will once we create that function. Let's do that now!

4. Put the cursor just above the `</script>` tag but below the `}` that ends the *if* statement above.
5. Press **ENTER** once or twice to add a blank line or two.

6. Type or copy and paste the following comment:

```
//This function only runs when called. It plays the sound file in the player, if the player exists
```

7. Then type or copy and paste the following function below that:

```
function playsound() {}
```

8. Place your cursor between the curly braces for this function, and press **ENTER**.

9. Then type or copy and paste the following if statement:

```
if (window.player) {}
```

10. Place your cursor between the curly braces for this if statement, and press **ENTER**.

11. Then type or copy and paste the following code:

```
player.play();
```

How it Works

This new code is in a function named *playsound()*. That code will execute only when some event on the page (like our button) calls the function. When it's called, it does a little object detection to make sure the page contains an object named *player*. That's what the *if (window.player) {}* is about.

- It returns *true* if the page contains an object named *player*.
- It returns *false* if the page doesn't contain such an object (which would mean the player never got created, because the browser doesn't support HTML5



audio).

But assuming the page does contain an object named *player*, the function executes `player.play()`. The *.play()* is the JavaScript method for telling an HTML5 audio player to play whatever sound has loaded. Of course, *player* in this case is the HTML5 audio player we created and loaded up with a sound file as soon as the page loaded.

12. Save your progress.

13. To try it out, open *SoundFX.htm* in an HTML5 compatible browser. You should see a button on the page, as below.

Whoosh

14. Clicking that button should play the *whoosh.mp3* or *whoosh.ogg* file. If you hear the sound, then congratulations, you've done your first HTML5 audio page!

Next, we'll look at how you can use JavaScript to add background music and custom player controls to a page.

Playing Background Music

As some of you may know, HTML5 has an `<audio>` tag that can display a default player on the page for playing music. As an alternative to relying on the default audio player, you can create your own buttons and sliders for controlling playback. You've seen how to use JavaScript with buttons, dropdown lists, and other common form controls. Here we'll show you how to create and use a slider (called a *range* control) to allow the user to control sound volume. Some of the code for accomplishing such feats is similar to code you already saw. So to speed things along, we've given you a complete sample page, and we'll focus on those parts of the page that'll help you learn new JavaScript coding.

Here are the Steps

1. Open *BackgroundMusic.html* in your browser (you downloaded this file to your *Intro JavaScript* folder at the beginning of the lesson).

Background Music:

Play



2. Click the **Play** button. You should hear background music begin to play, and the button should now say **Pause**.
3. Now use your mouse to move the slider to the right of the **Play/Pause** button. You'll notice that the volume changes as you do.
4. Open *BackgroundMusic.html* in your editor.
5. Now, let's take a look at the JavaScript code that runs when the page first opens. (Hint: This code is in the `<script>` tags in the `<head>`.) You'll notice that it's similar to the code you created for *SoundFX.html*:

```
// This code runs as soon as page opens
//If browser supports HTML5 audio set up sound
if (window.HTMLAudioElement) {
  var player = document.createElement('audio');
  // Choose sound file to play based on browser capabilities
  if (player.canPlayType('audio/mpeg').length > 0) {
    player.setAttribute('src', 'sounds/landofpeace.mp3');
  }
  if (player.canPlayType('audio/ogg; codecs="vorbis"').length > 0) {
    player.setAttribute('src', 'sounds/landofpeace.ogg');
  }
  // Sets the loop attribute so music plays continuously
  player.setAttribute('loop', true);
}
```

What's Different?

There are two main differences between *BackgroundMusic.html* and *SoundFX.html*:



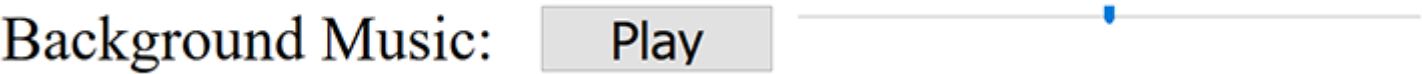
1. **The sound/music file.** In *BackgroundMusic.html* you'll notice that we're using a music file named *landofpeace* (rather than *whoosh*). This is a longer-running music track so you can adjust the volume slider as it's playing. You're free to use any music you like, so long as you're not infringing on someone else's copyright.
2. **Loop attribute.** You'll also notice there is a third *player.setAttribute* line near the end of that code:

```
player.setAttribute('loop', true);
```

This line turns on the loop attribute of the audio player, which makes the music restart each time it ends. (If you want your music file to play just once, you can remove that line.)

Controlling How the Music Plays

As you already know, when you open *BackgroundMusic.html* in an HTML5-compatible browser, you should see the text *Background Music*, a button, and a slider. The exact appearance of these custom player controls on the page depends on your browser.



- **Play/Pause Button:** The button allows you to play and pause the music.
- **Volume Control:** The slider control lets you control the volume.

Of course, the page could contain any other content you want. But right now we're concerned only with the background music. Let's break down how these custom controls work.

The Play/Pause Button

The Play/Pause button is a standard HTML button that's displayed in the body of the page with this tag:

```
<input type="button" id="playpause" value="Play" style="width: 60px" onclick="stopgo(this)">
```

Let's break this down:

Text equivalent start.

Topic	Information
<code>input type="button"</code>	This creates a button input.

Topic	Information
<code>id="playpause"</code>	This is just a made up name. We want the button to say Play when the music is off and Pause when the music is playing, hence the name <i>playpause</i> .
<code>value="Play"</code>	The button initially shows <i>Play</i> as the button text. For accessibility purposes, many browsers won't automatically play music when a page first opens. So, we'll start by assuming the sound isn't playing when the page opens.
<code>style="width: 60px"</code>	This sets the width of button. It doesn't impact the button height or the font size.
<code>onclick="stopgo(this) "</code>	When the user clicks the button, it uses the code <i>stopgo(this)</i> to call a JavaScript function named <i>stopgo</i> . The word <i>this</i> in this context is an actual JavaScript keyword that refers to the control that's calling the JavaScript. As you'll see, the JavaScript code can then refer back to the calling control to inspect its properties or make changes.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

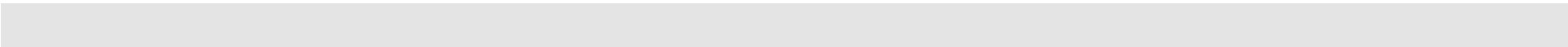
The *stopgo()* function


The *stopgo()* function that the button calls in the *BackgroundMusic.html* page consists of the following code:

```
//Play and pause the player
function stopgo(btn) {
  //Play or pause and change button text depending on current state
  if (btn.value == "Pause") {
    player.pause()
    btn.value = "Play"
  } else {
    player.play()
    btn.value = "Pause"
  }
}
```

Let's break this down.

Text equivalent start.



Topic	Information
<pre>function stopgo(btn) {</pre>	<p>Unlike the previous functions we've worked with, you'll notice that there is some text, <i>btn</i>, inside the parentheses. This text is usually called a <i>parameter</i>. A parameter, in this case <i>btn</i>, is just a variable name, and like any other variable name—it can be anything you want. It could have been <i>X</i> or <i>goober</i> or <i>howdy</i> or any other valid JavaScript variable name. The only thing that's different about that variable is that it gets its value from the code that's calling the function.</p> <div><div></div><div><h2>Parameter vs. Argument</h2><p>In our <i>BackgroundMusic.html</i> example, that calling code also has something between its parentheses:</p><pre>onclick="stopgo(this) "</pre><p>There, <i>stopgo</i> is the name of the function being called, and <i>this</i> is the value (also called the <i>argument</i>) that's being passed to the <i>stopgo</i> function. The <i>stopgo</i> function stores the value of <i>this</i> in its own <i>btn</i> variable, because <i>btn</i> is the name that we put between the parentheses in the function definition.</p><p>So again, looking at the function definition and the calling code:</p><pre>function stopgo(btn) onclick="stopgo(this) "</pre></div></div>

Topic	
	<p>The btn name is a <i>parameter</i>, which means its a variable that gets its value from whatever code calls the function. In the calling code, stopgo(this) the keyword <i>this</i> is the <i>argument</i>, which contains information about the button that was clicked. That argument's information is automatically stored in the <i>btn</i> variable. It is not necessary to use a var statement to define btn as a variable again later in the code. That variable gets is value the moment the function is called.</p>
<pre>if (btn.value == "Pause") {</pre>	Once the function is called, and the <i>btn</i> variable has its value, the function can make decisions based on information in the btn variable. This line means "If the button is currently showing the word Pause . . ."
<pre> player.pause()</pre>	If that's the case, then this line executes to pause (stop) the music that's playing.
<pre> btn.value = "Play"}</pre>	This is the next line that executes. This line changes the button text to <i>Play</i> . So now the music has stopped playing, and the button shows the word <i>Play</i> .
<pre>} else {</pre>	Now, suppose that when you clicked the button, it wasn't showing <i>Pause</i> . In that case, the music must've been paused already (because it's either playing or paused; it can't be anything else). So the code under the <i>} else {</i> line executes.
<pre> player.play()</pre>	In this instance, this first line to execute makes the music start again.
<pre> btn.value = "Pause"}</pre>	This is the next line that executes changes the word text to Pause.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

If it's hard to picture how that plays out in real life, open *BackgroundMusic.html* in a browser that can play HTML5 audio. Click the **Play/Pause** button (slowly) a few times. Then you can actually experience what the code is doing.


The Volume Control


To give the user the ability to control the sound volume, we used this tag to display an HTML5 *range* control on the page:

```
<input type="range" min="0" max="100" oninput="volume(this.value/100)" title="Volume">
```

Let's break this down:

Text equivalent start.

Topic	Information
<code>input type="range"</code>	<p>Even though it's just another <code><input></code> tag, the <i>type="range"</i> makes the control appear as a slider (in modern browsers).</p> <div><div></div><div><h3>Note</h3><p>In older browsers that don't fully support HTML5, it'll look like a text box and won't function as a slider at all. There's no workaround for that other than to completely hide the control, as we'll do later in this lesson.</p></div></div>
<code>min="0"</code>	<p>In the syntax for the slider, the <i>min</i> value is the minimum value, which the slider has when the slider box or highlight is all the way to the left. We have <i>min="0"</i> (that's the number zero, not the letter "O").</p>
<code>max="100"</code>	<p>The <i>max</i> is the maximum value, the value when the slider is all the way to the right. Other values, of course, increase from left to right. We have and <i>max="100"</i>.</p>

Topic	Information				
<pre>oninput="volume(this.value/100)"</pre>	<p>Moving the box or bar inside the slider triggers an <i>oninput</i> event and assigns a value to the control. In this example, the value will range from 0 to 100.</p> <div><div></div><div><h3><i>volume()</i></h3><p>To change the volume of sound, you use the <i>volume()</i> property of the audio element.</p><p>The basic syntax is:</p><pre>name.volume=x</pre><p>Replace <i>name</i> with the name of the audio control and x with a number between 0 and 1 to indicate the value. For example, <i>0</i> (zero) means no volume, <i>0.5</i> means half volume, and <i>1</i> means full volume.</p><p>We handle it in our JavaScript code by using <i>oninput="volume(this.value/100)</i> in the input tag of the range control. That calls a JavaScript function named <i>volume</i> every time the slider value changes. It passes to that function the value of the slider control divided by 100. So when the slider is near the middle of the control and its value is 50, it passes to the volume function <i>0.5</i>, which is mid-volume.</p></div></div> <tr><td><pre>title="Volume"</pre></td><td><p>This just tells you that the name of this particular range input is <i>Volume</i>.</p></td></tr> <tr><td colspan="2">Read the topic in the first column. Then read the second column for the information.</td></tr>	<pre>title="Volume"</pre>	<p>This just tells you that the name of this particular range input is <i>Volume</i>.</p>	Read the topic in the first column. Then read the second column for the information.	
<pre>title="Volume"</pre>	<p>This just tells you that the name of this particular range input is <i>Volume</i>.</p>				
Read the topic in the first column. Then read the second column for the information.					

Text equivalent stop.

The *volume()* function

In *BackgroundMusic.html*, the JavaScript *volume()* function looks like this:

```
function volume(amt) {  
    //Adjust the volume  
    player.volume = amt  
}
```

That volume function is actually very simple. Let's take a look:

Text equivalent start.

Topic	Information
<code>function volume(amt) {</code>	This tells us that the function <i>volume</i> stores whatever value is passed to it from the HTML code in a variable named <i>amt</i> (short for <i>amount</i> , but as always, this is just made up).
<code>//Adjust the volume</code>	This JavaScript comment describes what the executable code does.
<code>player.volume = amt}</code>	Then the one and only line of executable code within the function, <i>player.volume = amt</i> , applies that value to the volume of player, which is the name of the audio control that plays the music in our page.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

And that's how we control sound volume from HTML and JavaScript.

In addition to all you've seen so far, *the BackgroundMusic.html* page can also prevent the music and hide the player controls in browsers that aren't fully HTML5 compliant. It does that using JavaScript. Before we explain how, you'll need to learn a couple more JavaScript tricks.

Chapter 3: Using JavaScript with Input Types and CSS

Browser Compatibility with HTML5

HTML has been evolving for many years. The current version, HTML5, offers some controls (input types) that weren't available in older versions. We're using one of the newer types, the range control, in our *BackgroundMusic.html* page because it's ideal for adjusting sound volume.

While most people use browsers that are modern enough to handle the HTML5 audio and range controls, there'll still be a few with older browsers. This means we must make some concessions. We can help older browsers keep from stumbling over this modern code by using some input type detection and CSS manipulation via JavaScript.

Unfortunately, just because a browser can play HTML5 audio doesn't mean it can also display range controls. Any given browser version may not support all HTML5 features. Additionally, HTML5 may not be fully supported for many years to come. To play it safe, if you're using HTML5 audio and a range control for volume, your code should test for compatibility with both things.

If incompatibility is detected, it's easiest to simply not show the controls or play the music when someone is using an older browser. They won't get the full experience of the page, but it's better than having items on the page that don't work. There's not really a whole lot you can do about that.

Compatibility Detection

You already saw how you can use *if (window.HTMLAudioElement)* to test whether the browser can handle HTML5 audio. When detecting input controls, the syntax is a little different. You have to take the following steps:

1. Use `CreateElement` to create an input element.
2. Use `setAttribute` to define whatever input type you want it to be.
3. Determine if JavaScript accepted the type you assigned or defaulted to something else (like text) that the browser recognizes.

If you look at the code in *BackgroundMusic.html*, you'll see the following code in the `<script>` tags withing the page `<body>`:

```
//See if browser supports range control
var temp = document.createElement("input");

temp.setAttribute("type", "range");


//supportsrage will be true if browser supports the range control.
var supportsrage = (temp.type == "range");
```

This code sets a variable named *supportsrage* to *true* in any browser that supports the input range control and to *false* in any browser that doesn't support that input type.

Let's break it down.

Text equivalent start.

Topic	Information
var temp = document.createElement("input");	This line creates an <i>input</i> control and stores it in the <i>temp</i> variable. The variable name <i>temp</i> is short for <i>temporary</i> because we're just using the variable as a temporary placeholder to perform the test. The <i>input</i> control is a JavaScript object to be used for our test and has no visible presence on the page.
temp.setAttribute("type", "range");	This next linesets the <i>type=</i> attribute of the input element to the word <i>range</i> to make it a range (slider) control.

Topic	Information
<pre>var supportsrange = (temp.type == "range");</pre>	<p>Here is the tricky part, this line creates a variable named <i>supportsrange</i>. Like all variables, <i>supportsrange</i> is just a name we made up. The test, <i>temp.type=="range"</i>, is a comparison. The <i>temp.type</i> part uses the JavaScript type property to determine what the browser thinks the temp control's type is.</p> <p>Modern browsers that support the HTML5 range control will correctly return "range" for temp.type. In that case, the test temp.type=="range" proves true, and so the variable named <i>supportsrange</i> also gets a value of <i>true</i>.</p> <p>However, browsers that don't support range controls will default to "text" for temp.type. In this case, the test <i>temp.type=="range"</i> will prove false, since "text" doesn't equal "range". So the <i>supportsrange</i> variable will get a value of <i>false</i> with those nonsupporting browsers.</p> <div><div></div><div><h3>Text Type</h3><p>The <i>.type</i> property for an input control returns <i>"text"</i> for a textbox as well as for any unrecognized input type. It defaults to type="text" because that's one of the oldest and most widely used input types.</p></div></div>

Read the topic in the first column. Then read the second column for the information.

Text equivalent stop.

With this code, the variable named *supportsrange* will end up containing *true* if the browser supports the range input type or will contain *false* if the browser doesn't support that browser type. Which means we can use that for additional decision-making.

CSS Properties in JavaScript

As you know, CSS is a rich and powerful language for styling your websites. Because you can also control it from JavaScript, you can do "intelligent styling" in the sense the styles are applied, or removed, based on JavaScript's ability to make decisions. The syntax for using CSS with JavaScript is pretty simple, too. It goes like this:

```
element.style.property=value
```

Replace *element* with the specific element to which you want to apply the styling. That name is usually in some variable, like you've seen in previous examples. Replace *property* with the CSS property you're trying to apply. Replace *value* with the value that you want to assign to the CSS property.

Unfortunately, the property name you use in JavaScript doesn't always exactly match the property name in CSS. If the CSS property name contains a hyphen, in JavaScript you capitalize the first letter after the hyphen and remove the hyphen. Of course, as is usually the case in code, there are no blank spaces. Here are some examples:

CSS property	JavaScript name
background	background
background-color	backgroundColor
background-image	backgroundImage
border-style	borderStyle
Color	color
display	display
font-family	fontFamily
font-size	fontSize
line-height	lineHeight
Visibility	visibility

Making Controls Invisible

You can apply CSS styling via JavaScript. There's a whole lot you can do with that, but one of the best things is that you can have your JavaScript code decide whether to make something visible or invisible on the page based on its decision-making capabilities.

To make an element invisible, the CSS display property accepts *none* as a value for the CSS *display* property:


```
element.style.display="none";
```

When executed, that code hides whatever *element* you applied it to, so that element doesn't appear on the page. This means JavaScript can make decisions about what to show and what not to show on a page.

Now let's get back to *BackgroundMusic.html* and see how you can use these new features. Under the `<body>` tag in the page, we have a paragraph of code with an ID of *"soundcontrols"*:



Remember

That's the paragraph that creates the label, button, and volume control.

Below that, we have some JavaScript code. (It's at the bottom of the page, just before the `</body>` tag):

```
<script>

//Runs at end of page load, after everything else is rendered.

//Prevents sound and controls in incompatible browsers.

//See if browser supports range control.


var temp = document.createElement("input");

temp.setAttribute("type", "range");


//supportsrage will be true if browser supports the range control.
var supportsrage = (temp.type == "range");

// Show sound controls only in browsers that play sound and support range control.
if (supportsrage && window.HTMLAudioElement) {

    player.play()
} else {

    document.getElementById("soundcontrols").style.display = "none";

}

</script>
```

When executed, that code determines if the current browser supports range controls, as we discussed at the start of this chapter. If it does support range controls, the variable named *supportsrage* will be *true*; otherwise, *supportsrage* will be *false*.

Next, the code makes a decision on whether or not to display *soundcontrols* using the code under the JavaScript comment *// Show sound controls only in browsers that play sound and support range control*.

```
// Show sound controls only in browsers that play sound and support range control.
if (supportsrage && window.HTMLAudioElement) {

    player.play()
} else {

    document.getElementById("soundcontrols").style.display = "none";

}
```

Let's break it down.

Text equivalent start.

Topic	Information

Topic	Information
<pre>if (supportsrange && window.HTMLAudioElement) {</pre>	Recall that && is the logical operator for <i>and</i> . So this line of code says, "If this Web browser supports the range control and also supports HTML5 audio . . ."
<pre> player.play()</pre>	If true, this tells the browser to start playing the song that was already loaded into the player via the JavaScript code in the page header area.
<pre>} else {</pre>	If false, meaning the browser can't display range control or can't play HTML5 music, the following action will occur.
<pre> document.getElementById("soundcontrols").style.display = "none";}</pre>	This line of code sets the CSS display property of the entire <i>soundcontrols</i> paragraph to "none", which keeps that paragraph, including the text and player controls inside it, from appearing on the page.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

Since the code to start the playback was passed over before the *} else {*, the music never started playing either. So there's no music and no playback controls. This is the safest bet for any browser that's not quite ready for HTML5.

The element must come before the JavaScript styling.



This particular piece of code must be placed at the bottom of the page because JavaScript can only change the style of an element if that element already exists on the page. An element doesn't actually exist on the page until the code that defines that element executes. Code executes left to right, top to bottom in a page (the order we read in).

So, in order to hide the element named *soundcontrols* it must first exist on the page. This means that the code that creates the *soundcontrols* element (the `<p id="soundcontrols"> . . . </p>` tags) must execute first. Which means those tags must be above the JavaScript code that executes to find and change the CSS styling of the element in the page.

In other words, you must first create the element named *soundcontrols* before you can execute JavaScript code to hide it.

Let's review what you've learned.

Review

We examined a whole bunch of JavaScript capabilities in this lesson. Some were focused on sound. But others, like `setAttribute` and input type detection and controlling CSS with JavaScript, have wide-ranging possibilities for creativity. Let's review some key points:

- **Sound and JavaScript:** Here you learned that HTML5 lets you play audio from the browser without plug-ins. The HTML5 `<audio> . . . </audio>` tags allow you to create an audio object that can be controlled through JavaScript properties and methods. You also discovered that the *window.HTMLAudioElement* object and property return *true* in any browser that supports HTML5 audio. It returns *false* in browsers that don't support HTML5 audio. Properties and methods like *.canPlayType()*, *.play()*, and *.pause()* allow you to control what the player plays and when it plays it.
- **Adding Sound Files:** Here you learned that with `<audio>` tags you can link sound files to attributes. The JavaScript *.setAttribute* allows you to assign a value to any HTML attribute in any HTML tag. You also used if/else statements so that the button displays *Play* when the sound is off and *Pause* when the sound is on.
- **Using JavaScript with Input Types and CSS:** In this section you learned how to detect input type compatibility and hide controls using JavaScript to apply CSS styling. The JavaScript *.style* property allows you to apply CSS styling to any element on a page via JavaScript.

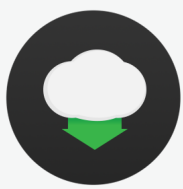
In the next lesson, you'll learn still more capabilities of the JavaScript language and build a useful and fun picture carousel in the process. See you there!

Lesson 6 Assignment Part 1

CSS Styling with JavaScript Decision-Making

Changing Background Colors

For this assignment, we'd like you to try your hand at changing the background color of an element in response to a mouse click. This has nothing do with sound, however, it does provide some practice with JavaScript decision-making and changing CSS styles through code.



Download

So you don't have to start from scratch, we've constructed a webpage that is already programmed to display a box that changes colors when you click it.

The following files you'll need are inside the download below:

- ColorCycle.html
- ColorCycle.txt



[Download L06-Assignment.zip](#)

1 KB.(Gigabytes) ZIP

You can retype or copy and paste if you'd like to run through the code, or you can simply download and save the files to your *Intro to JavaScript* folder.

Here's our challenge for this assignment:

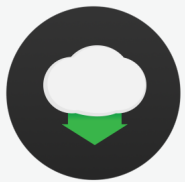
- Change the second *if* statement so that if the background color was green, it changes to blue.
- Add a third *if* statement that says if the background is blue, then change it to red.

The easiest way to solve this is to copy the code that is already there, and then modify it. Copying is easiest because you don't have to retype the code, and so you're much less likely to make common mistakes like forgetting a parenthesis or curly brace.

Here are the Steps

1. Open *ColorCycle.html* in your editor.
2. Copy the *if {}* statement that detects if it was green and then changes it to red.
3. Paste the code below the *if {}* statement you just copied. You'll have two identical *if {}* statements in a row.
4. Then, set the original *if {}* statement so it detects for green and then changes to blue.
5. Change the new *if {}* statement so it checks for blue and changes to red.
6. Save the changes and open it in a browser.
7. Click the box and watch the color change. With each click, it should change from red to green to blue to red in a continuous loop.

Try to see if you can do it on your own without peeking at the answer below.



Download

To check your code, you can download the files that contain the correct code here:



[L06-Answers.zip](#)











1KB.(Gigabytes) ZIP

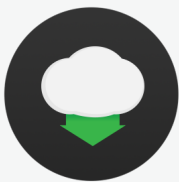
Lesson 6 Assignment Part 2

Customizing an Audio Library

For this part of the assignment, you'll have the opportunity to customize a sound library template that was created using the techniques you learned in this lesson.

Click a name or speaker to hear the sound. To download, right-click the mp3 or ogg link and choose Save Target As or Download Linked File, or whatever you normally use in your browser to download files.

 Alien	mp3	ogg
 Canary	mp3	ogg
 Chickens	mp3	ogg
 Crow	mp3	ogg
 Duck	mp3	ogg
 Effect1	mp3	ogg
 Effect2	mp3	ogg
 Raygun	mp3	ogg
 Sputnik	mp3	ogg
 Tiger	mp3	ogg



Download

We've created a page where users can play and download sample sound files that you can modify for your own purposes.

You'll need the following files:

- DownloadSounds.html
- DownloadSounds.txt



[Download L06-P2-Assignment.zip](#)

2.5 KB.(Gigabytes) ZIP

You can retype or copy and paste if you'd like to run through the code, or you can simply download and save the files to your *Intro to JavaScript* folder.

In the *Sounds* subfolder of your working folder, there is a little image named *speakericon.png* that you can use as the icon for playing a sound. If you look at the <body> you'll notice that the sample page uses that icon in each table row:

```
<tr>

<td onclick="play('alien')">Alien</td>

<td><a href="sounds/alien.mp3" download="alien.mp3" title="Download">mp3</a></td>

<td><a href="sounds/alien.ogg" download="alien.ogg" title="Download">ogg</a></td>

</tr>
```

Additionally, each of the sound files from the *Sounds* folder has been added to the table row as a download. You can replace those with sound files of your own.

Now, let's take a look at the code in the <script> tags in the <head>:

```
<script>

//Code that's executed as soon as the page opens

//Line below eliminates need for <audio>...</audio> tags

if (window.HTMLAudioElement) {

var player = document.createElement('audio');

var ext = ""

}

//Tries to play whatever name is pasted in

function play(filename) {

if (window.HTMLAudioElement) {

// Currently canPlayType(type) returns: "", "maybe" or "probably"

if (player.canPlayType('audio/mpeg').length > 0) {

ext = "mp3"
```

```
}

if (player.canPlayType('audio/ogg; codecs="vorbis"').length > 0) {

    ext = "ogg"

}

//Stop what's playing now

player.pause();

//Set src= attribute to desired sound filename

player.setAttribute('src', 'sounds/' + filename + "." + ext);

//Play the sound

player.play();

} else {

    //Incompatible browsers get an alert

    alert("Sorry, your browser can't preview HTML5 audio")

}

}

</script>
```

You'll notice that a player has been set up for the sounds. Additionally, just like we did earlier the code looks to see if the browser is compatible with HTML5 audio. If it's not, the user will get an alert that tells them their browser can't preview HTML5 audio. We won't go through all of the code step-by-step here. But there's really nothing in there in terms of the JavaScript code that hasn't already been covered in this course.

Another area you might want to look at is the code in the <style> tags in the <head>:

```
<style>

.downloads {

border-collapse: collapse;

}

.downloads tr:hover {

background-color: yellow;

}

.downloads td {

vertical-align: middle;

border: solid 1px silver;

padding: 8px;

cursor: pointer;

}

.downloads td img {

padding-right: 8px;

vertical-align: middle;

}

</style>
```

Here the code adds styling to the table. For instance, you'll notice that when you hover over a table row the background color changes to yellow.

To flex your coding muscles, go ahead and try making the following changes:

- Change the color that highlights the table row when you hover over it.
- Change the border style of the table.
- Change the font family and style when you hover over a table row.

- Change the text the alert displays if the browser doesn't support HTML5 audio.
- Add your own sound files to the table.
- Try making changes of your own.

After you're done experimenting, take a minute to go to the Discussion Area and answer some questions.

Lesson 6 Resources for Further Learning

Free Sound Effects (<https://www.youtube.com/audiolibrary/music>)

<https://www.youtube.com/audiolibrary/music>

From the Microsoft Developer Network (MSDN), this page provides extensive documentation on using JavaScript with the HTML5 audio object.

HTML Audio/Video Methods (https://www.w3schools.com/tags/ref_av_dom.asp)

https://www.w3schools.com/tags/ref_av_dom.asp

From the Microsoft Developer Network (MSDN), this page provides extensive documentation on using JavaScript with the HTML5 audio object.

Detecting HTML5 Features (<http://diveintohtml5.info/detect.html#input-types>)

<http://diveintohtml5.info/detect.html#input-types>

Though somewhat advanced, this page offers a lot of useful information on how you can use JavaScript to detect a browser's ability to handle different HTML5 features. Not really light reading, but it's probably worth keeping in your browser's Bookmarks or Favorites for future reference.

setAttribute() Method (https://www.w3schools.com/jsref/met_element_setattribute.asp)

https://www.w3schools.com/jsref/met_element_setattribute.asp

Here's some quick documentation on the JavaScript .setAttribute method.

Change CSS With JavaScript (<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style>)

<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style>

Here's the Mozilla Developer's Network documentation for changing CSS with JavaScript code.

Free Music and Sound Effects (https://alansimpson.me/public_domain/)

https://alansimpson.me/public_domain/

Look under *Categorized Collections* on this page for free public domain sound effects and music for your web site and other creative projects.