

# Introduction

## Welcome Back!

Today's lesson is about JavaScript timers and CSS transitions. Timers are JavaScript events that kill time. This might sound kind of crazy at first. After all, we want everything to run as fast as possible, so why would you want to intentionally slow things down in an app?



Most of the time, you do want things to run quickly. The one exception is transition effects, in which you may want to slow things down so people can see motion or change as it's happening. That's what we're going to look at in this lesson.

Historically, the only way to slow things down in JavaScript was with timers. However, CSS3 offers transitions, which can also be used to control timing and slow things down, so you're no longer entirely dependent on JavaScript for animation effects. As you'll learn in today's lesson, you can use JavaScript timers and CSS transitions to create control CSS transitions on your pages.

We'll start with timers.

## Experimenting with Timers



## Download

Below is the link to the ZIP file which contains the following resources you'll need for this lesson:

- **Timer:** You're welcome to type this page from scratch, if you're so inclined. Or, if you're not in the mood to practice typing right now, feel free to simply save it to your *Intro JavaScript* folder.
- **Timed Slideshow:** Here's another page that is set up with a basic slideshow. We're going to use it to add a timer.
- **Animals:** This folder contains five images of animals from pexels.com. Please save this folder to your *Intro JavaScript* folder.
- **Photo Slider:** Here's another slideshow. We're going to use it to add a responsive slider.
- **People:** This folder contains ten images of people from pexels.com. Please save this folder to your *Intro JavaScript* folder.



[Download Lesson-09\\_ClassProject](#)

34 MB.(Gigabytes) ZIP

If you have your own image files, you're welcome to use them instead.

# Chapter 1: JavaScript Timers

## Understanding Timing Events

### Methods for Adding Timers

JavaScript timers (also called *timing events*) allow you to put time delays on code execution. The basic idea is to tell JavaScript to wait, doing nothing for a period of time, and then execute a specified JavaScript function. So, in a sense, the passage of time (rather than a mouse click or other user activity) triggers an event that causes some other JavaScript code to execute.

There are two methods for using timers:

- **setTimeout("function", milliseconds):** Executes the named function once, after waiting the number of seconds specified by milliseconds.
- **setInterval("function", milliseconds):** Executes code or a function repeatedly at specific time intervals. It's kind of like a loop but with a time delay each time through the loop.

In both cases, you need to replace *function* with the name of the JavaScript function to be executed after the time delay. Replace *milliseconds* with a number indicating how long to wait.



### A millisecond is 1/1000th of a second

So if you want a time delay of one second, then you have to set milliseconds to *1000* (remember, no commas in your numbers). If you want it to wait half a second, then specify *500* as the milliseconds value, and so forth.

Both timers are methods of the window event. You may occasionally see them written with *window* and a *dot* at the start of the statement:

```
window.setInterval("function", milliseconds)
```

... or ...



```
window.setTimeout("function", milliseconds)
```

However, in JavaScript, the window object is assumed if you omit it. So, it's okay to leave off the leading *window* object name and *dot*.

## Methods for Canceling Timers

In order to cancel a timer, you need to associate the timer with a variable name. To assign a variable name to a timer, use this syntax:

```
var name = setInterval("function", milliseconds):
```

Replace *name* with a variable name of your own choosing. The function and milliseconds are the same as before. We used *setInterval* in the example, though it would work the same with *setTimeout*.

Once you've associated the timer with a variable name, you can use *clearInterval* or *clearTimeout* to stop the timer. The syntax is:

```
clearInterval(name)
```

... or ...

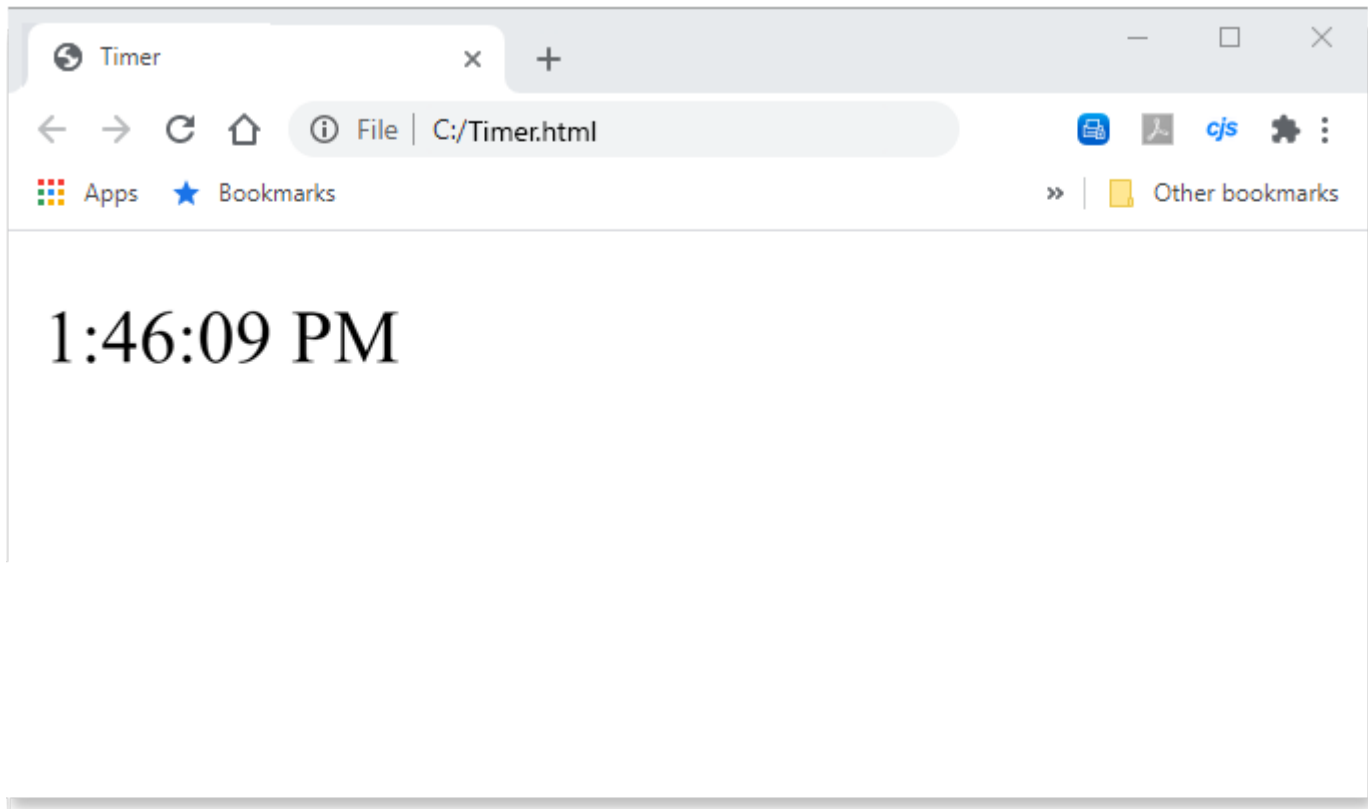
```
clearTimeout(name)
```

Replace *name* with the variable name of the timer that you want to stop.

## Displaying a Timer

### Adding the Timer

We gave provided you with *timer.html* at the beginning of this lesson, go ahead and open it in a browser. This page should be programmed so that you see the current time, updated every one second. Here is static screenshot of what you might see:



Assuming your computer's internal clock is correct, you'll see the current time for your location, updating every one second.

Let's talk about how and why it works now.

Text equivalent start.

Topic	Information
<pre>var clock = setInterval("showtime()", 1000);</pre>	This is the first bit of JavaScript code on the page is this, right under the <code>&lt;script&gt;</code> tag. Since it's not in a function, that code executes as soon as the page opens. It creates a variable named <i>clock</i> that is associated with a <code>setInterval</code> timer. The <code>setInterval</code> timer calls a JavaScript function named <i>showtime()</i> every one second (every 1000 milliseconds).
<pre>function showtime() { }</pre>	Next in the JavaScript code is the <code>showtime()</code> function that's being called every second. Inside the curly braces, you'll see how this function is defined.
<pre>var now = new Date();</pre>	This is the first line in the function. It creates a variable named <i>now</i> that is a <i>new Date()</i> object. That syntax always puts the current date and time, from the computer's internal clock, into the variable.
<pre>var time = now.toLocaleTimeString();</pre>	This next line in the function uses JavaScript's built-in <i>toLocaleTimeString()</i> method to take just the time portion of the current date and time, convert it into a string, and place it in a variable named <i>time</i> . In most places that string will be in hh:mm:ss AM/PM format.

Topic	Information
<code>document.getElementById("ptime").innerHTML = time;</code>	The last line in the function looks in the body of the page for an element named <i>ptime</i> and then sets its <i>innerHTML</i> to whatever is stored in the <i>time</i> variable. The <i>innerHTML</i> method puts the time between the <code>&lt;p id="ptime"&gt;</code> and <code>&lt;/p&gt;</code> tags that define that element.
<code>&lt;p id="ptime"&gt;&lt;/p&gt;</code>	If you look in between the <code>&lt;body&gt; . . . &lt;/body&gt;</code> tags, you'll see the element on the page is just a paragraph with an ID. Since the <code>showtime()</code> function gets called every one second, repeatedly, the time showing on the page is updated every one second.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

That's how your basic *setInterval* timer works. In the browser, you see it as the current time being updated every one second.

## Stopping the Timer

If, for whatever reason, you want to give the user the option to stop some timer-controlled activity, you can use a button that calls a function that uses *clearInterval* to stop the timer. As always, we'll use a simple example to see how things work. So let's add a button to our page to stop the timer.

## Here are the Steps

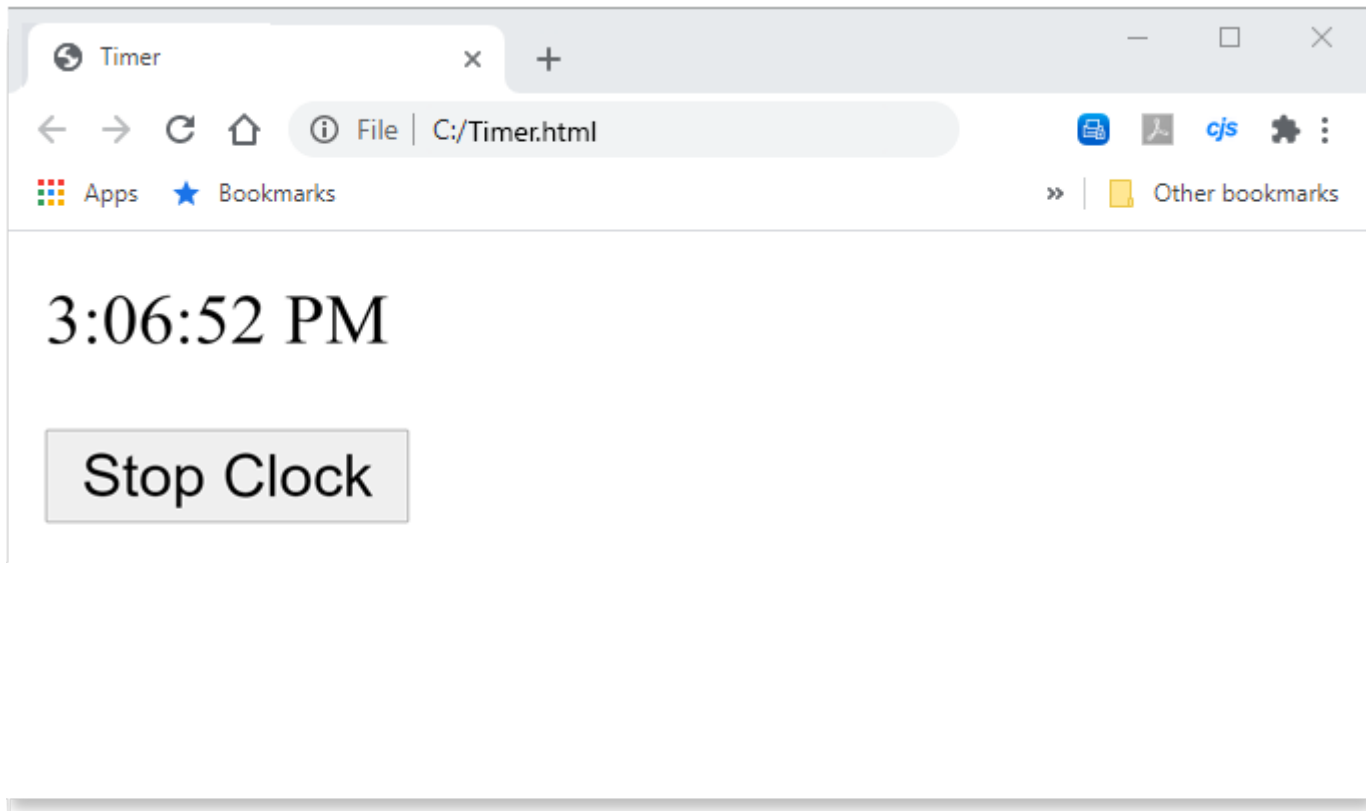
1. Open *timer.html* in your editor.
2. First, add a function named *stoptimer* that applies *clearInterval* to the *clock* variable. To do this copy/paste or type the code below so it's just above the closing `</script>` tag:

```
//Function to stop the timer
function stoptimer() {
    clearInterval(clock);
}
```

3. Next, add a button that calls the *stoptimer* function when clicked. You can copy/paste this HTML code below the `<p> . . . </p>` tag that's currently in the page so it's just above the closing `</body>` tag.

```
<p>
  <!-- Button to stop the timer -->
  <input type="button" value="Stop Clock" onclick="stoptimer()">
</p>
```

4. Save your changes. Then open or reload and refresh the page in a browser.
5. As before, the page should open, and the clock should update once every second. But this time, there will be a **Stop Clock** button below the time.




6. When you click the button, the time will still show. But it'll no longer update every one second because you stopped the timer.

Here's how it works.

Text equivalent start.

Topic	Information
<code>&lt;input type="button" value="Stop Clock" onclick="stoptimer()"&gt;</code>	This line of HTML code displays a button on the page with the words <i>Stop Clock</i> . When the user clicks the button, it calls a function named <i>stoptimer()</i> .

Topic	Information
<pre>function stoptimer() {clearInterval(clock);}</pre>	<p>The <i>stoptimer()</i> function, which you just added to the page, contains the line of code <i>clearInterval(clock);</i> that stops the timer that's associated with the variable name <i>clock</i>.</p> <div><h2>Remember</h2><p>Our timer has that variable name because we used <i>var clock = setInterval("showtime()", 1000);</i> to create the timer in the first line of JavaScript code on the page.</p></div>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

## Restarting the Timer

Of course, if you allow the user to stop a timer, you might also want to give them an option to restart it. Intuitively you might expect there to be some kind of restart method, or something along those lines, in the JavaScript language to make that happen. But there isn't. Instead, you just reset the initial variable name back to a `setInterval` method.

## Here are the Steps

1. Open *timer.html* in your editor.
2. Create a new function named *starttimer* and make it so that the clock variable is set to it's original value. To do this, type or copy/paste this comment and code so it's just above the closing `</script>`



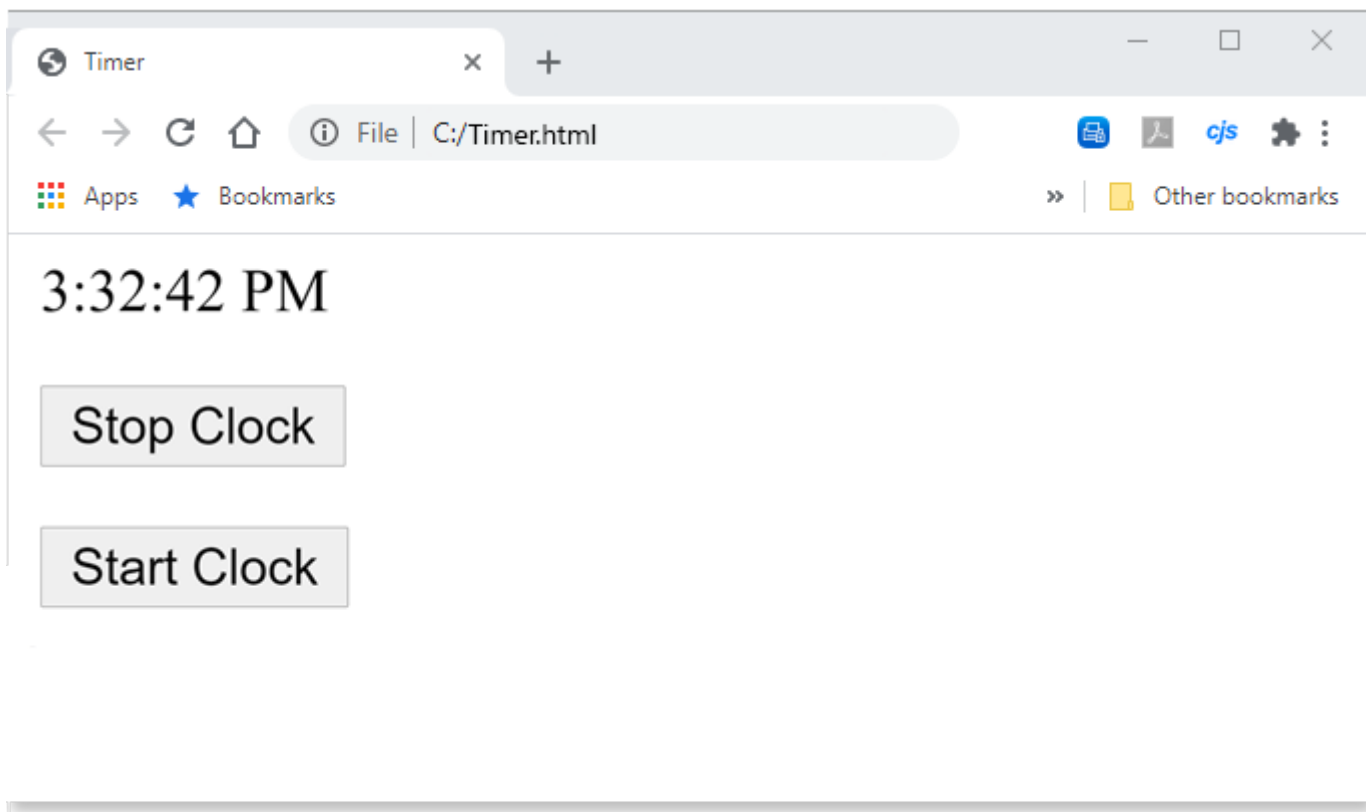
tag:

```
//Function to start the timer  
function starttimer() {  
    clock = setInterval("showtime()", 1000);  
}
```

3. Next, you need to add a **Start Clock** button that calls the *starttimer* function when clicked. Type or copy/paste this code so it's just below the **Stop Clock** button (below the `</p>` tag and above the `</body>` tag near the bottom of the page):

```
<p>  
<!-- Button to restart the timer -->  
<input type="button" value="Start Clock" onclick="starttimer()">  
</p>
```

4. Save the page, and view it in a browser.
5. In the browser, you see the clock ticking along as before, but now there are two buttons below it.



6. Go ahead and clicking the **Stop Clock** button. As before, the time is still showing, but it no longer updates. You've stopped the timer.
7. Now, clicking the **Start Clock** button. Watch as the time jumps forward and starts updating every one second again.

Here's how it works.

Text equivalent start.

Topic	Information
<pre>&lt;input type="button" value="Start Clock" onclick="starttimer()"&gt;</pre>	The <b>Start Clock</b> button works because you used this HTML code to display the button. Clicking that button calls the function named <i>starttimer()</i> .
<pre>function starttimer() {clock = setInterval("showtime()", 1000);}</pre>	The <i>starttimer()</i> function, in turn, executes the line of code that reads <i>clock = setInterval("showtime()", 1000);</i> . Here we're reassociating the original clock variable name with an interval timer that calls the <i>showtime()</i> function every one second. So, the clock starts updating every one second again, just as it did when the page first opened.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

In real life, you probably wouldn't use JavaScript timing events just to show the current time on a webpage. While it's a good exercise in trying things out, it's certainly not the end of the story. Timing events are also used for slideshows and many kinds of animated transition events. Let's look at how you could use a timer to create a simple slideshow on a page.

# Chapter 2: Timed Slideshow

## Adding a Timer to a Slideshow

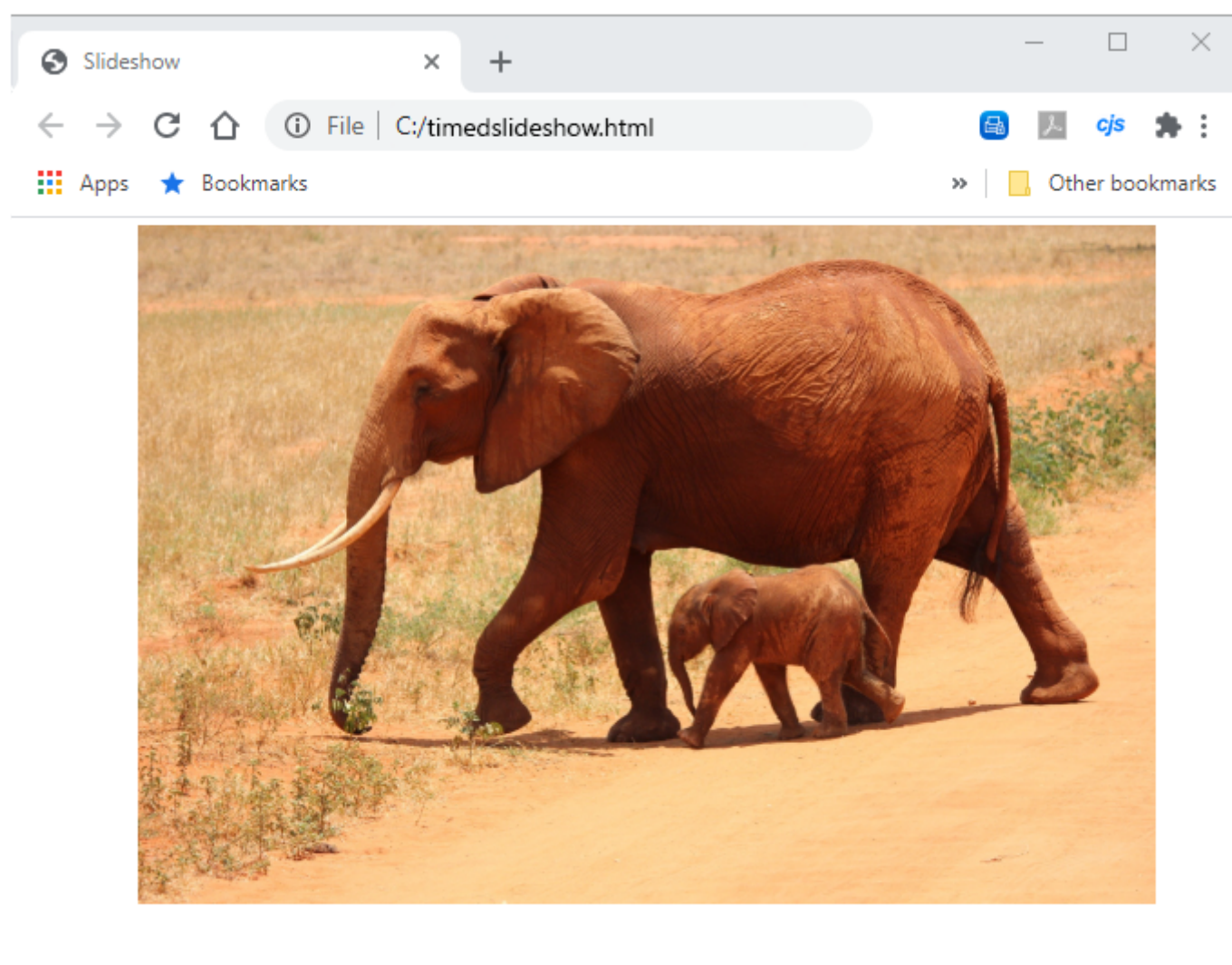
### A Simple Slideshow

You may be wondering why we're looking at another slideshow. However, there are so many ways to do slideshows, they're useful examples for many different kinds of JavaScript programming. Including advancing from one slide to the next automatically using a JavaScript timer, as you'll soon learn. There is no "better" or "faster" with each of the different techniques illustrated. They each have their own application depending on what you're trying to do. You can even use some of these techniques together, so there are no advantages or disadvantages that serve as a hard rule. They're just different ways of coding.

We've already given you all the code with plenty of comments so that you have it for future reference in the *timedslideshow.html* page you downloaded earlier. Unlike previous examples, we don't have to worry about pre-loading images with this one because the time it takes to transition from the first image in the slide show to the second is probably enough time for all the images to download without our having to code the download as we did in previous lessons. So let's get started.

### Here are the Steps

1. Open *timedslideshow.html* in your browser.
2. You should see a picture that changes every five seconds. The image below shows a single picture, but in your own browser that picture should change every few seconds.



3. Now, go ahead and open the page in your editor. Let's talk about how and why the slideshow works.

Of course, like any new code, it may look like alien gobbledygook at first. That's just a natural step in the learning process. But as we pick it apart, hopefully you'll see how it really is a kind of logic flow. We'll walk through the code and explain it so you understand exactly how everything works.

## Breaking Down the Code

### Adding Content and Style

Near the top of the page, in the `<head>` we have the standard HTML5 tags plus a couple of CSS style rules, one to style the div that contains the image shown in the slideshow, the other to style the actual image. The styling here is somewhat arbitrary. You can size and style things to your own taste.

```
<style>

/* CSS code to style the picture container and images */

#slideshow {

/* Height of box should roughly match image heights */

height: 750px;

text-align: center;

}

/* Size each image to fit container height */

#slideshow img {

height: 100%;

}

</style>
```

Down in the body of the page, you'll see the div tag and img tag to which the styling applies. When the page first opens, the img tag displays an image named *elephants.jpg* from the *animals* folder.

```
<!-- HTML to display the pictures -->

<div id="slideshow">



</div><!-- End slideshow div -->
```

In real life, that could be any image you care to show in your own slideshow. The id attribute in the img tag identifies the image as *slideimg* (one image at a time in a slideshow). The name, like any variable name, is just one we made up and can be any name you like.



## Important

While its name doesn't matter, it plays an important role in the code. It's the ID name that JavaScript will use to control what pictures show there.


Now that you understand the HTML5 and CSS framework, let's look at the JavaScript code in the page (which starts right under the `<script>` tag).


---

Text equivalent start.

Topic	Information
-------	-------------



Topic	Information
<pre>//Global variables visible to all code in page //Subscript of first image to show     var subscript = 0; //Number of seconds for each picture to show     var pictime = 5;</pre>	<p>For starters, we define a couple of global variables and assign some values to them. The comments in the code help to explain what each variable is for. As you can see, the variable named <i>subscript</i> is just a number that we initialize at zero. The <i>pictime</i> variable is a number to indicate the number of seconds to show each picture in the slideshow.</p> <div></div> <h2>Remember</h2> <p>As always, the variable names are just names we made up. The variables are <i>global</i> because we created them outside of any specific function. That means the variables are accessible to all the JavaScript functions in the page.</p>

Topic	Information
<pre>//List all of your image filenames. Filenames don't have to follow any pattern var images = new Array()  images[0] = "animals/elephants.jpg"; images[1] = "animals/parrots.jpg"; images[2] = "animals/flamingo.jpg"; images[3] = "animals/donkeys.jpg"; images[4] = "animals/sheep.jpg";</pre>	<p>Next, we create an array and put the names of all the image files for the slideshow into the array. The sample names are just examples.</p> <div></div> <h2>Remember</h2> <p>The filenames can be anything you like and can be in any subfolder you like (those all happen to be in the sample <i>animals</i> folder). You could even show pictures from external websites if you have an appropriate absolute reference for the picture (an http address that points directly to the picture filename).</p> <p>You can have as many images as you like in the slideshow and array. Just make sure your array subscripts are numbered sequentially (0, 1, 2, 3, 4, and so on) as in our working example.</p>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

In a moment, you'll see how we use JavaScript to pick one image at a time to show in the slideshow. But first, we want to call your attention to something else that's new in the code.

## Starting Page Execution on Page Load

One tricky thing about using JavaScript properly is making sure that all of the elements on the page that the JavaScript manipulates are actually on the page before the JavaScript tries to access them. As mentioned, any JavaScript code between the `<script> . . . </script>` tags at the bottom of the page just above the `</body>` tag will be executed only after everything else is downloaded.

Earlier we used a "modern JavaScript" method to delay execution with *addEventListener*, we'll revisit that later. An older and fairly common method is to use *window.onload()* to delay execution until all the HTML and CSS code has executed.



## Note

There were some inconsistencies across browser brands and versions with the earlier methods, which caused them to fall out of favor at one point. However, the *window.onload()* method is generally considered reliable so you may see it in other peoples' code. Here's the syntax:

```
window.onload = function() {  
    JavaScript code to execute on page load;  
}
```

You type all of the code exactly as shown. You have to replace the placeholder *JavaScript code to execute on page load;* with the JavaScript code to execute when the page is loaded. That can be any code you like.

Make sure you put it between the `<script> . . . </script>` tags. Most people will put it in the head of the page with other JavaScript code, usually just above the `</script>` tag for that code. For our slideshow example, we defined a timer inside the curly braces of a `window.onload`, as below:

```
//Make sure everything is loaded before starting timer  
window.onload = function () {  
    var stimer = setInterval("nextimg()", pictime * 1000);  
}
```

So, basically, that chunk of code says that once the page is fully downloaded, create a timer named *stimer*. Have that timer call a function named *nextimg()* every five seconds.

So now, let's look at the `nextimg()` function and see how it changes the image that's showing when it's called.

## The nextimg() Function

The `nextimg()` function is the real meat of the slideshow code, as it's the one that, when called, changes the image that's showing in the slideshow. First, here's that code in its entirety:

```
    //Show the next image in the array
function nextimg() {
    //Calculate next subscript (uses ternary operator)
    subscript = (subscript == images.length - 1) ? 0 : subscript + 1;
    var imagefile = images[subscript];
    //Find img tag and change its src= attribute
    var img = document.getElementById("slideimg");
    img.src = imagefile;
}
```


Like any JavaScript function, this one has a name that we made up. The name is *nextimg()*. The trickiest and most advanced line of code here is the first one:

```
subscript = (subscript == images.length - 1) ? 0 : subscript + 1;
```

This statement uses a ternary operator to calculate the subscript of the next image to show (the subscript is the number between square brackets in the array elements). Let's dissect it a bit since it's somewhat advanced and abstract:

Text equivalent start.

Topic	Information
subscript =	This says "set the value of the variable named subscript."

Topic	Information
<pre>(subscript == images.length - 1)</pre>	<p>This is the conditional ("if") part of the ternary expression. That says, "If subscript is equal to the length of the images array minus 1."</p> <div></div> <p>In an array of 5, the last subscript is 4.</p> <p>The last subscript in an array is always one less than the length of the array. That's because the first item is item 0, not item 1. If you count the number of items in the array, you can see there are indeed five of them ([0], [1], [2], [3], [4]).</p>
<pre>? 0</pre>	<p>This says, "If the condition is true, set the variable to 0."</p>
<pre>: subscript + 1;</pre>	<p>This part says, "Otherwise, set the subscript number equal to whatever value it has right now, plus 1." That decision is made each time the function is called. So the subscript numbers just keep cycling through the pattern 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4 for as long as the page remains open.</p>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

Logically, that line of code reads as follows:

Text equivalent start.

Full Line of Code	<pre>subscript = (subscript == images.length - 1) ? 0   subscript + 1;</pre>		
1st Section	<pre>subscript = (subscript == images.length - 1)</pre>	<i>If</i>	The current subscript is equal to the number of items in the images array
2nd Section	<pre>  ? 0</pre>	<i>Then</i>	Set the subscript back to zero



3rd Section	<code>subscript + 1;</code>	<i>Else</i>	Set the subscript to whatever it is now, plus 1
End		<i>End If</i>	
Breaking Down the Code			

Text equivalent stop.

The rest of the code in the `nextimg()` function should look a little more familiar. The line that reads *`var imagefile = images[subscript];`* sets the name of the *imagefile* variable to whatever array element the subscript variable says to set it to. So:

- When the subscript contains 0, the *imagefile* variable receives a value of *`animals/elephants.jpg`*.
- When the subscript contains 1, the *imagefile* variable receives a value of *`animals/parrots.jpg`*.
- When the subscript contains 2, the *imagefile* variable receives a value of *`animals/flamingo.jpg`*.
- When the subscript contains 3, the *imagefile* variable receives a value of *`animals/donkeys.jpg`*.
- When the subscript contains 4, the *imagefile* variable receives a value of *`animals/sheep.jpg`*.

The next line of code in the `nextimage` function, *`var img = document.getElementById("slideimg");`*, locates the tag that contains *`id="slideimg"`*, which, of course, is the `<img>` tag on the page that shows a slideshow image.

Then the last line of code in that function, *`img.src = imagefile;`*, sets the `src=` attribute of that `img` tag to whatever is in the *imagefile* variable, and so the image that's showing on the page is now that next image in the array. That's how you get the slideshow effect, by having JavaScript change the `src=` attribute to a different image filename every five seconds.

When you watch the slideshow, you might notice that the change from one image to the next is very abrupt. It might look a little nicer if we put in a gentler fade from one image to the next. Fortunately, we can do that too with just a little more CSS and a little more JavaScript. We'll show you how, up next.

# Chapter 3: Creating a JavaScript Slider

## Adding a Responsive Slider

### What is a Responsive Slider?

In this chapter we will create yet another slideshow. Having slides advance automatically (meaning the user doesn't have to click anything to go from one picture to the next) is a great example of using a JavaScript slider. To make it even more professional, we'll add two effects:

- We'll set it so each image slides into view from left to right, as opposed to just changing or fading.
- We'll make the slideshow responsive, so that it fits perfectly on any screen size and never loses its shape.

Here is an example of a responsive slider:

The HTML and CSS code you'll need to start with is in the *photoslider.html* page you downloaded earlier. Opening the page in a browser won't show a slideshow yet, because we'll need a JavaScript timer to run that.



### Note

We've provided some references to the CSS opacity and transition properties in the **Resources for Further Learning** for this lesson, in case you want more information on either. You can also Google *css opacity* or *css transition* to research on your own.

### Creating the Slider

To get the sliding effect, we can't just change the source of an image. We have to apply the `.slideout` class to the image that's showing, and the `.slidein` class to the image that's coming in. Images that aren't showing at all are actually just to the right of the slideshow, but have no visibility until a JavaScript timer makes the next image visible and slides it in from the right. This may seem a bit abstract still, but some of the JavaScript code may look familiar as we step through the process.

# Here are the Steps

1. Open *photoslider.html* in your browser.
2. To start with, please add the following code just under the opening `<script>` tag in the page:

```
//Just a generic counter for the next() function to determine the next image to show
var counter = 0;

// Name of subolder containing images
var picfolder = "people"

// Image filenames in the array
var picfiles = ["image01.jpg", "image02.jpg", "image03.jpg", "image04.jpg", "image05.jpg",
"image06.jpg", "image07.jpg", "image08.jpg", "image09.jpg", "image10.jpg"];

//Placeholder for any image's path (folder/filename) populated later
var imgsourc = "";

// Number of images in array is length minus one (because first is always zero)
var imgcount = picfiles.length - 1;
```



## Setting Up Variables

In this code we're just setting up some variables to contain information the JavaScript will need. Comments explain what's going on. We used the people folder, and images with imagexx.jpg file names. However, you can use any image file names, and as many or as few images as you like. Just make sure the picfolder variable contains the name of the subfolder in which the image are stored, and the picfiles array accurately lists the file names of the images you intend to use.

3. Next, add the following code below the code you just entered, so it's just above the closing `</script>` tag:

```

//Wait until the page is fully loaded before setting source of the image.

window.addEventListener('load', (event) => {

    for (var i = 0; i < picfiles.length; i++) {

        // Create image path (folder/filename)

        imgsource = picfolder + '/' + picfiles[i];

        // Create an image element

        var img = document.createElement("img");

        //Make the source of that image the next path in the loop

        img.src = imgsource;

        //Need to give that image an id

        img.id = "img" + ('0' + i.toString()).slice(-2)

        // Append the image to the window element

        document.getElementById("window").appendChild(img);

        //Need to start first image as slidein class or it looks wonky on first
image

        if (i == 1) {

            document.getElementById(img.id).classList.add('slidein');

        }

    }

    //All the setup is done, so call next() function every 3 seconds

    var timer = setInterval(function () {

        next()

    }, 3000);

})

```



## Waiting for the Page to Load

This code waits for the page to load, because it needs to be able to find the element named *window*, which is just a div in the body of the page. It contains an `<img>` tag, which is what shows the actual image, one image at a time. Once the page is loaded, we fire up a loop that repeats once for each filename in the *picfiles* array. After that code completes, the timer starts with `var timer=...` but it doesn't work yet. The timer has to call a function named `next()` to start the process, and we haven't put that code in yet.

4. Place the following code just under the global variable (the line that reads *var imgcount = picfiles.length - 1;*):

```

//Figures out which image to show next, restarts at 1 after imgcount reached.

function next() {

    //Increment the counter (or restart at 1)

    counter += 1;

    //Current image id number is equal to counter.

    var currentimage = counter % imgcount;

    //The next image is counter + 1

    nextimage = (currentimage + 1) % imgcount;

    //Generate the id name, needs leading zero if < 10.

    thisimgid = "img" + ('0' + currentimage.toString()).slice(-2);

    nextimageid = "img" + ('0' + nextimage.toString()).slice(-2);

    //Pass the ids of the images to the slide function.

    slide(thisimgid, nextimageid);

}

```

5. The code won't work without a function named slide. You can put that in now, right below the closing curly brace for the next() function, by copy/pasting in this code:

```

//i1 is the image showing, i2 is the next image to show.

function slide(i1, i2) {

    //Slide out the current image

    document.getElementById(i1).className = "slideout";

    //Slide in the next image

    document.getElementById(i2).className = "slidein";

    //Wait, then recycle the last slid-out image over to the pile to the left
of #window

    setTimeout(function () {

        hideit(i1)

    }, 1100);

}

```

6. This code requires a function called hideit().But you can add it now, right below the code you just added by typing or copying/pasting it from here:



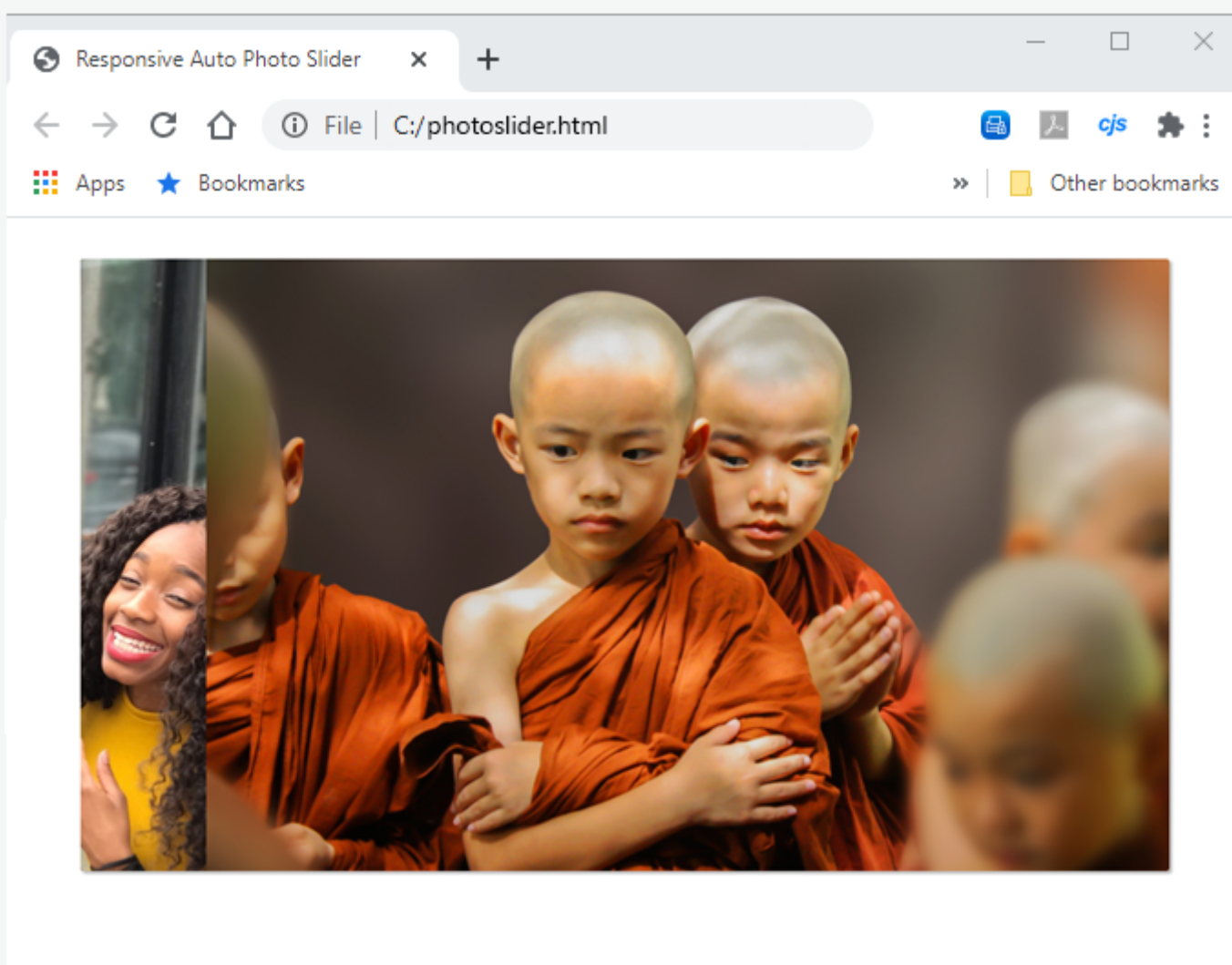
```
//Removing the class name moves it back to the left of the show where it's hidden.  
  
function hideit(nm) {  
  
    document.getElementById(nm).className = "";  
  
}
```

7. Save your progress.

8. Now, go ahead and open *photoslider.html* in your browser.

## Sneak Peak

You'll now notice that the slideshow cycles through the images by sliding them in and out of the page on their own (without buttons).



## Breaking Down the Code

Window.addEventListener()

```
//Wait until the page is fully loaded before setting source of the image
window.addEventListener('load', (event) => {
    for (var i = 0; i < picfiles.length; i++) {
        // Create image path (folder/filename)
        imgsource = picfolder + '/' + picfiles[i];
        // Create an image element
        var img = document.createElement("img");
        //Make the source of that image the next path in the loop
        img.src = imgsource;
        //Need to give that image an id
        img.id = "img" + ('0' + i.toString()).slice(-2)
        // Append the image to the window element
        document.getElementById("window").appendChild(img);
        //Need to start first image as slidein class or it looks wonky on first
image


        if (i == 0) {
            document.getElementById(img.id).classList.add('slidein');
        }
    }
}
```

Let's walk through this code.

Text equivalent start.

Topic	Information
imgsource = picfolder + '/' + picfiles[i];	Within this loop we create a variable named imgsource which ends up containing the complete path to one image. For example people/image01.jpg on the first pass through the loop.
var img = document.createElement("img");	Then this line creates an image element (basically an empty image tag with no position on the page yet).
img.src = imgsource;	This line then assigns the actual image file name as the source (src) of that image.

Topic	Information
<pre>img.id = "img" + ('0' + i.toString()).slice(-2)</pre>	<p>To that same image we need to assign an id name, because the javascript code accesses the images by those id names. This line assigns an id name based on the current number.</p> <p>For images with numbers less than 10 we need to stick a 0 to the left side. The slice(-2) tells the code we want "the two rightmost characters". So when a two-digit number like 10 gets a zero added to the left, it ends up being 010 but we don't need the leading zero on a two-digit number, so the <i>slice</i> just takes the last two digits, 10.</p>
<pre>document.getElementById("window").appendChild(img);</pre>	<p>Finally, whatever image that name results in get appended to the window div with this line of code. The <i>appendChild</i> method just ensures that each image is contained within the div named window, which is here all our images need to be for later JavaScript code to work properly.</p>

Topic	Information
<pre>if (i == 0) {document.getElementById(img.id).classList.add('slidein');}</pre>	<p>On the first pass through the loop, we add the CSS class name <i>slidein</i>, which is the CSS code that allows the image to slide in.</p> <div><h3>Note</h3><p>The complete name of that CSS class is <code>#window img.slidein{}</code> but that's not a problem because the image is, indeed, inside the div with the id of window. That CSS class sets the left edge of the image to the left side of the window (<code>left:0</code>), but it takes one second to get there because of <code>transition-duration: 1s</code>;. This is what causes the left-to-right motion of the image sliding into view.</p></div>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

The tricky thing with this code is that we need to know the ids of two images, and it's the `next()` function that provides those names.

## Function `next()`


```
//Figures out which image to show next, restarts at 1 after imgcount reached
function next() {
    //Increment the counter (or restart at 1)
    counter += 1;
    //Current image id number is equal to counter
    var currentimage = counter % imgcount;
    //The next image is counter + 1
    nextimage = (currentimage + 1) % imgcount;
    //Generate the id name, needs leading zero if < 10
    thisimgid = "img" + ('0' + currentimage.toString()).slice(-2);
    nextimageid = "img" + ('0' + nextimage.toString()).slice(-2);
    //Pass the ids of the images to the slide function
    slide(thisimgid, nextimageid);
}
```

Now, let's take a closer look at this code.

Text equivalent start.

Topic	Information
<code>counter += 1;</code>	This line increments the global counter variable by one.
<code>var currentimage = counter % imgcount;</code>	We set a variable named <i>currentimage</i> equal to the counter number, but use modulo to divide by the number of images in the array, which keeps the number from going higher than 10.



Topic	Information
<pre>nextimage = (currentimage + 1) % imgcount;</pre>	<p>We do the same thing for nextimg, which will be the image that's slding in. It's just the current image number plus 1, but again we use modulo to keep the numbers in the 1-10 range.</p> <div></div> <h3>Note</h3> <p>No matter how many times the code runs, both currentimage and neximage will be in the range of 1 to 10 because there are 10 images in our array, as indicated by the variable named imgcount.</p>
<pre>thisimgid = "img" + ('0' + currentimage.toString()).slice(-2); nextimageid = "img" + ('0' + nextimage.toString()).slice(-2);</pre>	<p>Now that we have those two umbers, we can use them define the complete ids of the current and next image, which will be something like img01, img02, or whatever, up to img10. Again we have to ensure only two digits are added to the "img" portion of the name, so we use .slice(-2) which means "the last two characters in the string". Those names are stored in variables named thisimid and nextimgid.</p>
<pre>slide(thisimgid, nextimageid);</pre>	<p>Finally, we pass those two image id names to a (currently non-existent) function named <i>slide</i>, via this line.</p>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

# Function slide()

```
//i1 is the image showing, i2 is the next image to show

function slide(i1, i2) {

    //Slide out the current image
    document.getElementById(i1).className = "slideout";

    //Slide in the next image
    document.getElementById(i2).className = "slidein";

    //Wait, then recycle the last slid-out image over to the pile to the left of
    #window

    setTimeout(function () {


        hideit(i1)

    }, 1100);

}
```

Let's take a stroll through this code.

Text equivalent start.

Topic	Information
<code>document.getElementById(i1).className = "slideout";</code>	What happens there is that the first image has the class named <i>slideout</i> added to it, which makes it slide out of view off the left.
<code>document.getElementById(i2).className = "slidein";</code>	Then immediately the second image gets the <i>slidein</i> class name added to it, which causes it to slide into view.
<code>setTimeout(function () {hideit(i1)}, 1100);</code>	<div>The image that is sliding out of view has to be moved back over to the right side of the window. But if you do that too quickly, you may see some activity mixed in with the other sliding in. So we wait 1.1 seconds before doing that with this code.</div> <div><div></div><div><h3>Note</h3><p>That code won't work without the function named <i>hideit()</i>.</p></div></div>

Text equivalent stop.

---

Using *.className* to apply a CSS property that contains a transition event will make the transition event play out on the screen. However, if some other transition needs to take place after that, then you may need to use *.setTimeout* to apply the first transition event to play out before applying the next style class.

You may feel a bit overwhelmed right now, but we've included many comments in the code to explain what's going on, so hopefully those will help when you refer to the code again in the future. Your slideshow should work now if you open *photoslider.html* in a browser.

Let's take a break now and review what you've learned in today's lesson.

# Review

Today's lesson was mainly about JavaScript timing events, which are handled with the `setInterval` and `setTimeout` methods. You also got to practice writing some code with arrays and ternary operators from Lesson 8. Then we went all out and used advanced CSS3 animation techniques with advanced JavaScript techniques to create a fully automated responsive slider that will play and look good in any browser on any size screen.

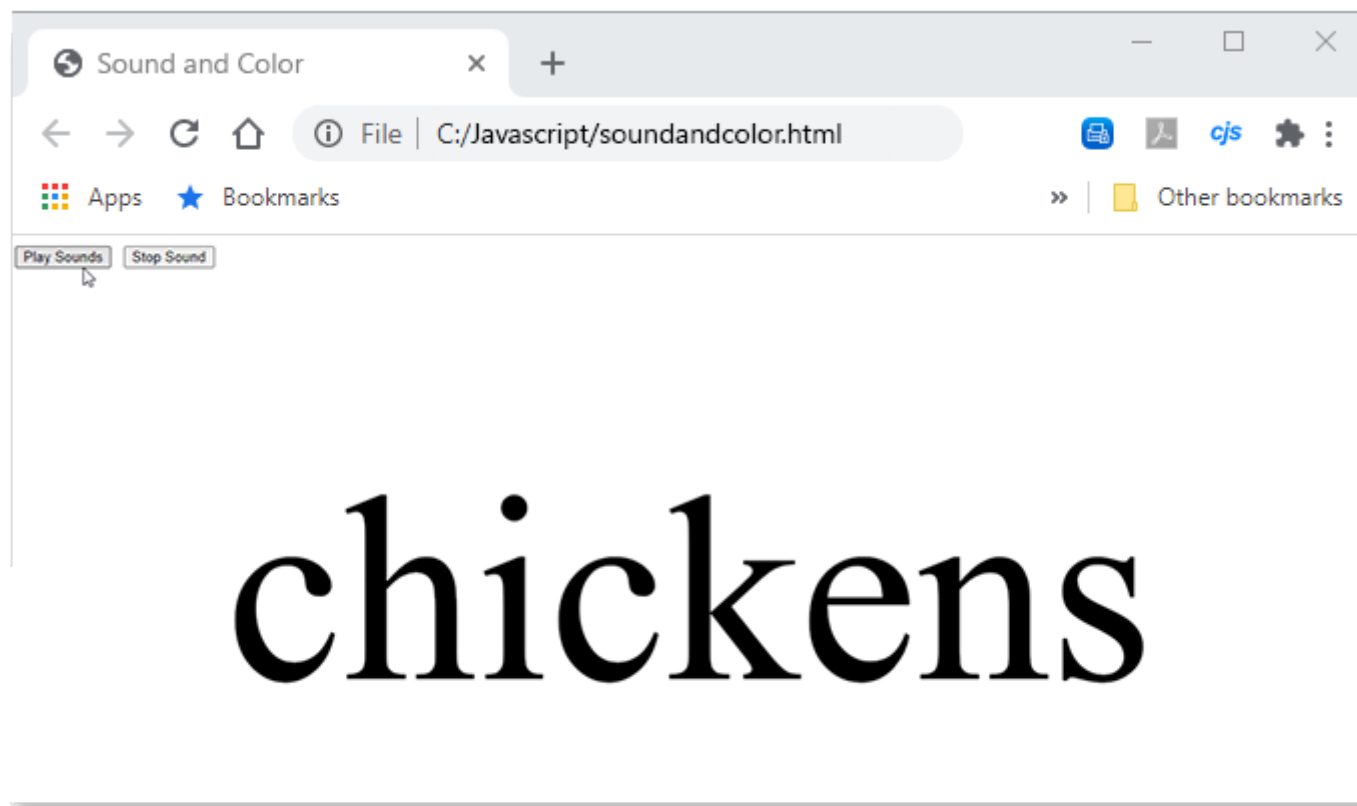
Needless to say, there's a lot more you can do. The things you learned today are a good starting point for more-advanced activities, so let's take a moment to recap the most important points:

- **JavaScript Timers:** You found that the *.setTimeout* method allows JavaScript to call some function once, after a time delay, while the *.setInterval* method allows JavaScript to call some function repeatedly, with a time delay between each call.
- **Timed Slideshow:** Here you learned that you can use JavaScript timers to create a slideshow that navigates through images without buttons.
- **Creating a JavaScript Slider:** In this section, you discovered how to create a responsive slider. The *JavaScript .className* property allows JavaScript to get, or set, the name of a style class that's applied to some tag. CSS3 offers a *transition:* property that allows you to animate the application of a CSS property by extending the time it takes for the property to be applied.

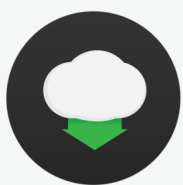
Using timing events and animation effects can be challenging, even for experienced JavaScript programmers. So challenging, in fact, that many JavaScript programmers have written entire libraries of such code to simplify the process. The most famous and most widely-used JavaScript library is named jQuery, and some of you may have come across that name when researching JavaScript on your own online.

# Lesson 9 Assignment

## Timing with Sound and Color



For this assignment we'll have some fun with a page that plays a sound effect once every 3 seconds. It will also show the name of the sound file, in a randomly-selected color, on the screen.



### Download

So you don't have to start from scratch, we've constructed a webpage that is already partially programmed for you. Below is the link to the ZIP which contains following files you'll need:

- **Sound and Color:** This file contains *soundandcolor.html* and *soundandcolor.txt* which contains the HTML and CSS, and most of the JavaScript code. The code will use the same audio files from the *sounds* folder you downloaded earlier.



[Download L09-Assignment.zip](#)

1.8 KB.(Gigabytes) ZIP

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

Here's what you need to do:

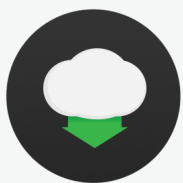
- **Complete function `playall()`**

- Write the code to start an interval timer called *playsound()* every three seconds.
- Use the name *clock* as the timer name, as that name is already declared as a global variable.
- Put the command under the comment that reads *//Then play every 3 seconds*.

- **Finish function `shutup()`**

- Stop the interval timer by adding the appropriate code in the function named *shutup()* under the comment that reads *//Stop the timer*.

First try It without peeking. If you get stuck, you can see both functions, with the appropriate code filled in, below.



## Download

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



[Download L09-AssignmentAnswers.zip](#)

1.8 KB.(Gigabytes) ZIP

# Lesson 9 Resources for Further Learning

## **JavaScript Timing Events** ([https://www.w3schools.com/js/js\\_timing.asp](https://www.w3schools.com/js/js_timing.asp))

[https://www.w3schools.com/js/js\\_timing.asp](https://www.w3schools.com/js/js_timing.asp)

Click this link for a brief but useful description of JavaScript timers and some examples. This is a good candidate for your browsers Bookmarks or Favorites.

---

## **JavaScript Timers** (<https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php>)

<https://www.tutorialrepublic.com/javascript-tutorial/javascript-timers.php>

Here you will find another good article on using JavaScript timers.

---

## **JavaScript Date Object** ([https://www.w3schools.com/jsref/jsref\\_obj\\_date.asp](https://www.w3schools.com/jsref/jsref_obj_date.asp))

[https://www.w3schools.com/jsref/jsref\\_obj\\_date.asp](https://www.w3schools.com/jsref/jsref_obj_date.asp)

This is a handy reference page for the JavaScript Date() object and its many methods. It's another very good reference to add to your browser's Bookmarks or Favorites because there are too many to remember, and you want an easy way to look them up on the fly.

---

## **onload Event** ([https://www.w3schools.com/jsref/event\\_onload.asp](https://www.w3schools.com/jsref/event_onload.asp))

[https://www.w3schools.com/jsref/event\\_onload.asp](https://www.w3schools.com/jsref/event_onload.asp)

Here's yet another short handy reference worthy of inclusion in your browser Bookmarks or Favorites. Even though this page also shows using onload= in a page tag, that approach doesn't have great support across the many brands and versions of browsers out there. The window.onload syntax we used in this lesson is more reliable.

---

## **CSS Viewport Units: A Quick Start** (<https://www.sitepoint.com/css-viewport-units-quick-start/>)

<https://www.sitepoint.com/css-viewport-units-quick-start/>

See this page for a great tutorial on vw and vh units of measure for responsive sizing.

---

## **CSS Animations** ([https://www.w3schools.com/css/css3\\_animations.asp](https://www.w3schools.com/css/css3_animations.asp))

[https://www.w3schools.com/css/css3\\_animations.asp](https://www.w3schools.com/css/css3_animations.asp)

Here you'll find a summary of all the animations capabilities of modern CSS.

---



**Animation** (<https://css-tricks.com/almanac/properties/a/animation/>)

<https://css-tricks.com/almanac/properties/a/animation/>

Here's another good article on CSS animations.

---

**Transition** (<https://css-tricks.com/almanac/properties/t/transition/>)

<https://css-tricks.com/almanac/properties/t/transition/>

Here's a good article on using CSS transitions to control animation.