

Introduction to JavaScript Training

Images, Windows and Timers

Lesson 1, Activity 2: Image Rollovers

Image rollovers are commonly used to create a more interesting user experience and to help highlight navigation points. When the user hovers the mouse over an image, the source of the image is modified, so that a different image appears. When the user hovers the mouse back off the image, the original source is restored. The code below shows how to create a simple rollover.

Code Sample:

ImagesWindowsTimers/Demos/SimpleRollover.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Simple Image Rollover</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>
<body>
<div style="text-align:center;">
<h1>Simple Image Rollover</h1>

<p>Who are you calling simple?</p>
</div>
</body>
</html>
```

The mouse-over event is captured with the `img` tag's `onmouseover` event handler. When this happens, the following JavaScript code is called:

```
this.src = 'Images/Hulk.jpg';
```

The `this` object refers to the current object - whatever object (or element) the `this` keyword appears in. In the case above, the `this` object refers to the `img` object, which has a property called `src` that holds the path to the image. The code above sets the `src` to "Images/Hulk.jpg".

Likewise, the mouse-out event is captured with the `img` tag's `onmouseout` event handler. When this happens, the following JavaScript code is called:

```
this.src = 'Images/Banner.jpg';
```

This code sets the `src` to "Images/Banner.jpg," which is what it originally was.

An Image Rollover Function

Image rollovers can be handled by a function as well:

Code Sample:

ImagesWindowsTimers/Demos/SimpleRolloverFunction.html

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Simple Image Rollover Function</title>
<link href="style.css" rel="stylesheet" type="text/css">
<script type="text/javascript">
function imageRollover(img, imgSrc){
    img.src = imgSrc;
}
</script>
</head>
<body>
<div style="text-align:center;">
<h1>Simple Image Rollover Function</h1>


<p>Who are you calling simple?</p>
</div>
</body>
</html>
```

Lesson 1, Activity 4: Preloading Images

When working with files on a local machine, image rollovers like the ones we have seen in previous examples work just fine. However, when the user first hovers over a rollover image, the new image file has to be found and delivered to the page. The time it takes to load the image will depend on its size and the user's connection. It could take a few moments, causing an ugly pause in the rollover effect. This can be prevented by preloading images.

Images can be preloaded by creating an `Image` object with JavaScript and assigning a value to the `src` of that `Image`. A sample is shown below:

Code Sample:

<ImagesWindowsTimers/Demos/PreloadingImages.html>

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Preloading Images</title>
<link href="style.css" rel="stylesheet" type="text/css">
<script type="text/javascript">

var imagePaths = [];
imagePaths[0] = "Images/Hulk.jpg";
imagePaths[1] = "Images/Batman.jpg";

var imageCache = [];

for (var i=0; i<imagePaths.length; i++) {
  imageCache[i] = new Image();
  imageCache[i].src = imagePaths[i];
}

function imageRollover(img, imgSrc) {
  img.src = imgSrc;
}
</script>
</head>
<body>
<div style="text-align:center;">
<h1>Simple Image Rollover Function</h1>


<p>Who are you calling simple?</p>
</div>
</body>
</html>
```

Notice that the code is not in a function. It starts working immediately as follows:

1. An array called `imagePaths` is created to hold the paths to the images that need to be preloaded.

```
var imagePaths = [];
```

2. An array element is added for each image to be preloaded.

```
imagePaths[0] = "Images/Hulk.jpg";  
imagePaths[1] = "Images/Batman.jpg";
```

3. An array called `imageCache` is created to hold the `Image` objects that will hold the preloaded images.

```
var imageCache = [];
```

4. A `for` loop is used to create an `Image` object and load in an image for each image path in `imagePaths`.

```
for (var i=0; i<imagePaths.length; i++) {  
    imageCache[i]=new Image();  
    imageCache[i].src=imagePaths[i];  
}
```

Lesson 1, Activity 5: Creating a Slide Show

Duration: 20 to 30 minutes.

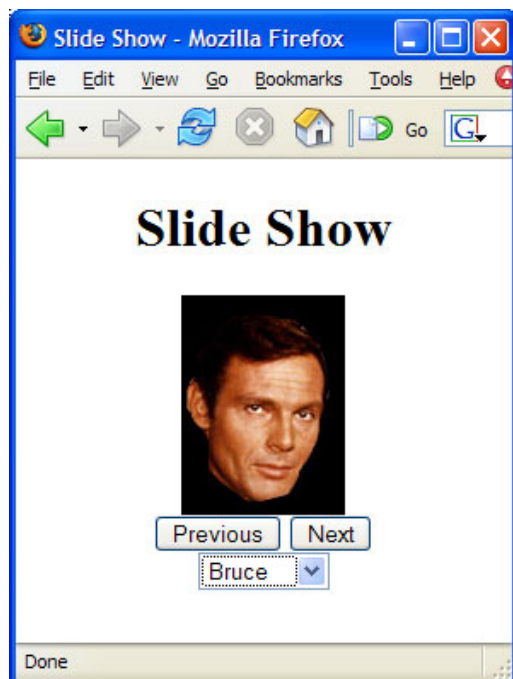
In this exercise, you will practice working with images by creating a slide show.

1. Open [ImagesWindowsTimers/Exercises/SlideShow.html](#) for editing.
2. Write code to preload [Images/Banner.jpg](#), [Images/Hulk.jpg](#), [Images/Bruce.jpg](#), and [Images/Batman.jpg](#). The paths to the images should be stored in an array called `imagePaths`. The `Image` objects should be stored in an array called `imageCache`.
3. Create a function called `changeSlide()` that takes one parameter: `dir`, and behaves as follows:
 - If `dir` equals 1, it changes the slide to the next image as stored in the `imagePaths` array.
 - If `dir` equals -1, it changes the slide to the previous image as stored in the `imagePaths` array.
 - If the user is viewing the last slide and clicks the `Next` button, the first slide should appear.
 - If the user is viewing the first slide and clicks the `Previous` button, the last slide should appear.
4. Test your solution in a browser.

Challenge

1. Add a dropdown menu below the `Previous` and `Next` buttons that contains the names of the images without their extensions: "Banner", "Hulk", "Bruce" and "Batman".
2. When the user selects an image from the dropdown, have that image appear above.
3. When the user changes the image above using the `Previous` and `Next` buttons, have the dropdown menu keep in sync (i.e, show the name of the current image).

The solution should look like the screenshot below.



Solution:

[ImagesWindowsTimers/Solutions/SlideShow.html](#)

---- C O D E O M I T T E D ----

```

<script type="text/javascript">
var imagePaths = [];
imagePaths[0] = "Images/Banner.jpg";
imagePaths[1] = "Images/Hulk.jpg";
imagePaths[2] = "Images/Bruce.jpg";
imagePaths[3] = "Images/Batman.jpg";

var imageCache = [];

for (var i=0; i<imagePaths.length; i++) {
    var s = imagePaths[i];
    imageCache[i] = new Image();
    imageCache[i].src = s;
}

var curSlide = 0;
function changeSlide(dir){
    curSlide += dir;
    if (curSlide < 0) {
        curSlide = imageCache.length - 1;
    } else if (curSlide >= imageCache.length) {
        curSlide = 0;
    }
    document.Slide.src = imagePaths[curSlide];
}
</script>
---- C O D E   O M I T T E D ----

```

Challenge Solution:

[ImagesWindowsTimers/Solutions/SlideShow-challenge.html](#)

```

---- C O D E   O M I T T E D ----

for (var i=0; i<imagePaths.length; i++) {
    var s = imagePaths[i]
    imageCache[i] = new Image();
    imageCache[i].src = s;
    imageCache[i].name = s.substring(s.lastIndexOf("/") + 1, s.lastIndexOf("."));
}

var curSlide = 0;
function changeSlide(dir){
    curSlide += dir;
    if (curSlide < 0) {
        curSlide = imageCache.length - 1;
    } else if (curSlide >= imageCache.length) {
        curSlide = 0;
    }
    document.Slide.src = imagePaths[curSlide];
    document.forms[0].imageSelector.selectedIndex=curSlide;
}

function goToSlide(slide){
    document.Slide.src = imagePaths[slide];
    curSlide = slide;
}
</script>

```

---- C O D E O M I T T E D ----

```
<input type="button" onclick="changeSlide(1);" value="Next"><br>
<select name="imageSelector" onchange="goToSlide(this.selectedIndex);">
<script type="text/javascript">
for (var i=0; i<imageCache.length; i++) {
  document.write("<option value='" + i + "'">"
    + imageCache[i].name + "</option>");
}
</script>
</select>
```

---- C O D E O M I T T E D ----

Lesson 1, Activity 6: Windows

These days, popup windows are generally frowned upon; however, they can be useful in some cases. We'll look at a couple of examples, but first, let's see how to open a new window.

Syntax

```
var newWin = window.open(URL, name, features, replace);
```

All four parameters are options:

1. URL - the URL of the page to load. If it is left blank, a blank window is open and can be written to with `newWin.document.write()`.
2. name - the target attribute of the window. This can be used to reuse an existing window if it is open.
3. features - a comma-delimited list of window features. Some of the most common are:
 1. height - the height of the window
 2. width - the width of the window
 3. left - the left position of the window
 4. top - the top position of the window
 5. location - whether or not to include the location bar
 6. menubar - whether or not to include the menubar
 7. resizable - whether or not the window should be resizable
 8. scrollbars - whether or not to include scrollbars
 9. status - whether or not to include the status bar
 10. toolbar - whether or not to include the toolbar
4. replace - true or false. If set to true, the new page replaces the current page (if there is one) in the browser window's history.

The height, width, left, and top features should be set in pixels.

The location, menubar, resizable, scrollbars, status, and toolbar features are boolean values: "true" or "false", "yes" or "no", or "1" or "0" will work.

The [HTML5 specification](#) advises browsers to ignore the features arguments completely, and some modern browsers do choose to ignore all but the size and positioning features.

The example below shows how to open a new window:

Code Sample:

[ImagesWindowsTimers/Demos/window.html](#)

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>New Window</title>
<script type="text/javascript">
var eula;

function openWin() {
    eula = window.open("eula.html", "eula", "height=200,width=300,left=100,top=100");
    eula.focus();
}
```

```

function eulaChecked() {
  if (document.getElementById("confirmEula").checked && typeof eula == "undefined") {
    alert("Come on now. Don't you think you should read the EULA first?");
    document.getElementById("confirmEula").checked = false;
    openWin();
  } else if (document.getElementById("confirmEula").checked && !eula.closed) {
    eula.close();
  } else if (!document.getElementById("confirmEula").checked) {
    openWin();
  }
}
</script>
</head>
<body>
<form action="process.xyz">
<input type="checkbox" id="confirmEula" name="confirmEula" onclick="eulaChecked()"> Check to confirm...
</form>
</body>
</html>

```

Things to notice:

1. We make `eula` a global variable so that we can access the window object from within multiple functions.
2. In the `openWin()` function, we call `eula.focus()`; after opening the window. That's to bring the window to the foreground if it's already been opened.
3. In the `eulaChecked()` function:
 1. We first check to see if the `confirmEula` checkbox has been checked before the EULA window has been opened (i.e., before a window object has been assigned to the `eula` variable). In this case, we scold the user and pop up the EULA window.
 2. We then check to see `confirmEula` checkbox has been checked and the EULA window is left open, in which case we close the EULA window via the window object's `close()` method.
 3. Finally, if the `confirmEula` checkbox has been unchecked, we pop the EULA window back open.

Lesson 1, Activity 8: Timers

Timers are started and stopped with the following four methods of the window object:

1. `setTimeout(code_to_execute, wait_time_in_milliseconds)`
2. `clearTimeout(timer)`
3. `setInterval(code_to_execute, interval_in_milliseconds)`
4. `clearInterval(interval)`

Let's take a look at how `setTimeout()` and `clearTimeout()` work first:

Code Sample:

<ImagesWindowsTimers/Demos/Timer.html>

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Timer</title>
<script type="text/javascript">
  var timer;
  function changeBg(color) {
    timer = setTimeout(function() { document.bgColor=color; }, 1000);
  }

  function stopTimer() {
    clearTimeout(timer);
  }
</script>
</head>
<body>
<button onclick="changeBg('red')">Change Background to Red</button>
<button onclick="changeBg('white')">Change Background to White</button>
<button onclick="stopTimer()">Wait! Don't do it!</button>
</body>
</html>
```

Things to notice:

1. We make `timer` a global variable so that we can access the timer object from within multiple functions.
2. In the `changeBg()` function, we create the timer. Note that we need to place `document.bgColor=color;` in a function; otherwise, the code will execute immediately. In this case, we use an anonymous function, which works just like a regular function. Note that because the function is nested within the `changeBg()` function, it has access to the local variable `color`.
3. The `stopTimer()` function simply clears the timer `timer` using `clearTimeout()`.

The `setInterval()` and `clearInterval()` methods work the same way. The only difference is that the code gets executed repeatedly until the interval is cleared.

Lesson 1, Activity 10: Popup Timed Slide Show

Duration: 15 to 25 minutes.

In this exercise, you will create a popup slideshow that runs by itself.

1. Open [ImagesWindowsTimers/Exercises/Timed-SlideShow.html](#) in your editor.
2. Add code to the `startShow()` function to pop up [popup-show.html](#) in a new 200px by 300px window. Make sure it comes to the foreground.
3. Open [ImagesWindowsTimers/Exercises/popup-show.html](#) in your editor.
4. Notice the `play()` function is called when the window loads. Add code to the `play()` function so that it changes the slide every second (1000 milliseconds).
5. Add code to the `stop()` function to stop the slideshow.

Solution:

[ImagesWindowsTimers/Solutions/Timed-SlideShow.html](#)

```

---- C O D E   O M I T T E D ----

<script type="text/javascript">
var showWin;

function startShow() {
  showWin = window.open("popup-show.html","showWin","height=300,width=200,left=300,top=100");
  showWin.focus();
}
</script>
---- C O D E   O M I T T E D ----

```

Solution:

[ImagesWindowsTimers/Solutions/popup-show.html](#)

```

---- C O D E   O M I T T E D ----

var showInterval;
function play() {
  showInterval = setInterval(function() { changeSlide(1); },1000);
}

function stop() {
  clearInterval(showInterval);
}
</script>
---- C O D E   O M I T T E D ----

```