webmonkey/programming/

Thau's JavaScript Tutorial

by Thau!

In earlier lessons, you've learned:

- How to add JavaScript to your HTML pages
- How to use dialog boxes and variables to store and utilize user input
- How to write HTML to a Web page using JavaScript
- How to let JavaScript make decisions using `if-then` statements
- How to make your Web pages react to users' actions using link events
- How to do a basic image swap

So far I've explained how to do many things, but I haven't described why they work. In the last lesson, for instance, I showed you that `window.document.monkey_image.src = "happy_monkey.gif"` will swap `happy_monkey.gif` into an image named `monkey_image`. But what is that `window.document` stuff? And where does the `.src` come from? Similarly, you've seen `document.writeln("monkey")`. But why is it `document.writeln` and not just `writeln`?

The answer to the above questions can be found in the JavaScript Document Object Model. The DOM is the way JavaScript describes Web pages, and it lies at the heart of all JavaScript programming. This lesson will teach you about the DOM, and the next lesson will teach you the rest of the basics of computer programming. By the end of the next two lessons, you will know all of the major ideas and syntax of JavaScript. All that will be left to learn are details, tricks, and how to avoid snafus.

To start us off along the road to the DOM, let's learn about how to use JavaScript to open and manipulate new browser windows.

Introduction to Window Manipulation

Before learning how to open a window in JavaScript, you should know how to open one using HTML. In most recent browsers, you can open a new window using an `href` statement. For example, clicking on this link will open another window.

The HTML used to do this is:

```
clicking on <a
href="yer_new_window.html" target="yer_new_window">this
link</a> will open another window.
```

The important thing to know about windows opened by targeted links is that the window above now has the name "yer_new_window" associated with it. If you have another `href` that uses "yer_new_window" as the target, and you haven't closed the window yet, whatever URL you put in that link will open in the original window. If you want to see what I mean, click here and then open another HTML page in the same window.

For the purposes of this lesson, I'm going to call the target name, in this case `yer_new_window`, the name of the window.

Now that we've had this brief refresher on `href` targets, it's time to learn about opening windows in JavaScript.

Window Manipulation in JavaScript

While opening windows in HTML is very handy, it's also limiting; the Web browser controls how the window looks. You have no control over the size of the window or what the window looks like. Happily, JavaScript gives you this control.

Here's how:

```
window.open("URL","name","features");
```

This statement opens a window with the URL that you list as the first parameter in the method call. Above it's called "URL," but in an actual call you'd write "http://www.hotwired.com/webmonkey/" or something similar.

The second parameter of the method call is the window's name. This is just like the name we saw in the last page. If you open a window and there's already a window open with the same name, the URL in your open statement will be sent to that open window.

The third parameter, features, is a list of the different components a window can have. It's an optional parameter, so let's do some examples with the first two parameters before checking out features.

Here are some examples of using JavaScript to open windows.

Examples of Opening Windows with JavaScript

Try clicking on these three links and see what happens. Don't close any of the windows until you've clicked on all three links.

Here's a window named javascript_1.
Here's a window named javascript_2.
Here's another HTML page going into javascript_1.

Let's take a look at how this is done. Here's the first line:

```
<a href="#" onClick="window.open('javascript_window_1.html','javascript_1');
return false;">Here's a window named javascript_1</a>.
```

When you click on this link, a new window called javascript_1 gets opened and the HTML page javascript_window_1.html gets put into the window. Because the features parameter is optional, you can just leave it out. This will give you the default window you would have gotten if you'd used a targeted href, as in the previous page.

Notice that I put the call to open the window in an onClick. You don't have to put window.open() calls inside an onClick, it was just convenient to do so in this example. You'll see examples of window.open() inside <script> tags soon.

The second link is almost exactly like the first link. It just opens a window with a different name and loads in a different HTML page:

```
<a href="#" onClick="window.open('javascript_window_2.html','javascript_2');
return false;"> Here's a window named javascript_2</a>.
```

The third link sticks a new HTML page into the first window. This works because the third link opens a window with the same name as the first window: javascript_1.

```
<a href="#" onClick="window.open('javascript_window_3.html','javascript_1');
return false;">Here's another HTML page going into javascript_1</a>.
```

Now the fun begins. We can play with the features parameter to make our windows look very different.

OK already, let's play with windows features!

Windows Features

The third parameter of the window.open() method is a list of features that you'd like your window to have. As you've seen, if you don't include this parameter at all, the window will contain all the features of a default browser window.

However, if you specify any features in the third parameter, just those features will appear. The way to specify that you'd like your window to have certain features is to list them as a comma-separated list.

For example, if you write ...

```
window.open
("some_url","window_name","location,menubar");
```

... you'll get a window with just the location box (the place in your browser where you type in a URL) and a menu bar (File, Edit, etc.). **Note:** that these code samples may wrap to fit here in your browser, but they should be all on one line, no spaces, when you actually use them.

Here's another example:

```
window.open("some_url","window_name",
"location,height=100,width=100");
```

This will open a window that is 100 pixels high and 100 pixels wide and has no features other than a location field. Notice again that there are no spaces in the string.

If you'd like to open a window that has almost all the features, you can specify which features you don't want by setting those features equal to no. For example:

```
window.open("some_url","window_name",
"location=no,status=no");
```

This will open a window that has all the features except the location field and the status bar. Here's a list of the features that you can include in the feature string:

`menubar`
: This is the row of functions that appears on most software applications. Normally it includes `File`, `Edit`, and a few other items.

`status`
: This is the message bar at the bottom of your window. When you move your mouse over an HTML link, the URL appears in the status bar. You may have seen pages that use JavaScript to turn this status bar into a scrolling marquee. I'm **not** going to show you how to do this. If you want to know, you have to figure it out yourself. "Down with marquees," the monkey cried!

`scrollbars`
: This allows scrollbars to appear when necessary.

`resizable`
: If `resizable` is listed, the window can be resized. Be careful of the spelling. I always get it wrong.

`width`
: The width of the window in pixels.

`height`
: The height of the window in pixels.

`toolbar`
: The browser toolbar, which contains the `Back` and `Forward` buttons, the `Stop` button, and the `Home` button, among others.

`location`
: The text area of a browser into which you can type URLs.

`directories`
: The directories that Netscape browsers have called "What's new," "What's cool," and so on.

Here are some examples of different types of windows.

Once you've checked out the examples, and maybe tried bringing up some windows of your own, it's time to learn how to mess with the contents of the windows.

The JavaScript Document Object Model

Now that you know how to open the window of your choice, it's time to learn how to manipulate things inside that window. To really get control over the things that appear in a window, you have to learn about the JavaScript Document Object Model (DOM). And before you learn about the DOM, it helps to learn a little bit about object-oriented programming.

A Brief Overview

Object-oriented programming — especially the JavaScript version of it — isn't so hard to understand. The main idea is that information is organized in terms of objects. JavaScript is wonderful because it comes with a built-in library of objects. For example, a window is an object. Whenever I refer to one of the default JavaScript library objects, I will capitalize it (Window). Specific instances (a particular window) will be lowercase.

Object Properties

Objects have properties that describe them. Some of the properties of a Window object are its name, the words in its status bar, the URL of the document inside the window, and the window's document itself, which includes the words, images, and hyperlinks inside the window.

In JavaScript, you are given a default Window object called, of all things, `window`. One of the properties of a window is the words in its status bar. Here's how you can find out what's in the status bar of the default Window:

```
var the_status = window.status;
```

This says: "Find the `status` property of the Window object called `window`, and load it into the variable `the_status`." In addition to reading what those words are, you can also change them. The way to set an object's property is this:

```
window.status = "I'm monkeying around!";
```

Now let's take a look at how we can mess around with the status bar.

Messing with the Status Bar

Move your mouse over the following two links and keep your eye on the bottom of the page.

Who do we love?
Why do we love them?

Here's the source code for the first link:

```
<a href="#" onMouseOver="window.status='Monkeys!';return true;">Who do we
love?</a>
```

This says, "When the mouse is moved over this link, change the status bar." If you look closely, you'll notice a `return true` inside the `onMouseOver`. If you don't put that there, the browser will do what it normally does when you move your cursor over a link: It'll look at the URL in the href and put it in the status bar, replacing the text you cleverly put there. But by putting `return true` in the `onMouseOver`, you stop the browser from doing this. It's sort of like the `return false` you put inside `onClicks` to stop the browser from trying to load a Web page:

```
<a href="#" onClick="return false;">this link does nothing and goes
nowhere!</a>
```

Why it's `return false` in an `onClick` but `return true` in an `onMouseOver` is one of those mysteries of life that are better left mysteries.

Objects Have Methods

In addition to properties, objects also have methods. Methods are the actions an object knows how to take. For example, Windows know how to open other Windows: `window.open("URL,""name,""features")`. This tells JavaScript to use the open method of the Window object to open a new window.

So, as in the example above, a `method` of an object is called the same way that a `property` is accessed: the object name, a period, and then the method. The main difference is that methods are always followed by parentheses that contain the parameters of the method. Even if the method takes no parameters, you still need the parentheses. For example, remember this?

```
var italic_hippy = hippy_monkey.italics();
```

Yup, strings are actually objects, and `italics()` is a method of the String object.

You've seen several other examples of Window methods and haven't known it. The dialog box calls `alert` and `prompt` are actually methods of the Window object. If you call:

```
window.alert("Viva la primate!");
```

You'll get an alert box that reads "Viva la primate!" The reason that we were able to just say `alert("Viva la primate!")`, as in earlier examples, is because `window`, being the default top-level object, can be left out. This means that `open("URL","name,""features")` does the same thing as `window.open("URL,""name,""features")`.

I'm not going to go into all the `methods` of Window objects here, just two more: `focus` and `blur`. The `focus` method moves a window that's behind other windows to the front. The `blur` method does the

reverse — it moves a window behind other windows. Unfortunately, blur and focus don't work with Internet Explorer 3.0 or Netscape 2.0, so again, if you're using either of these, it might be time to upgrade.

Try blurring this window.

The code inside the onClick for this link is:

```
onClick="window.blur();return false;"
```

The window.blur() tells the browser to send the window to the back. This is very nice, but it seems sort of silly to just blur the window that you're looking at. In order to use focus and blur more usefully, you need to know how to invoke method calls on other windows.

Communicating between Windows

Although it doesn't make sense to blur or focus the window you're on, you might well want to move another window to the fore. In order to communicate with a window using JavaScript, you need a reference to that window. Look at this example and then come back here for an explanation.

Getting and using a window reference.

There are a few key lines in this JavaScript. First, we open a new window and get a reference to it:

```
var new_window = window.open("hello.html","html_name","width=200,height=200");
```

This opens a little window and assigns the variable new_window to refer to it. Just as variables can contain numbers and strings, variables can also contain references to objects — in this example, a Window object. Now the variable new_window will behave just like the default Window object. You can call methods on new_window just like you could on window.

The next line shows you an example of calling a method on new_window:

```
new_window.blur();
```

Not too tricky, it's just like window.blur() from the last page. These two lines appear in the head of the HTML page. View Source on the page if you want to see the whole thing. As I've mentioned, I tend to put my JavaScript code in the head of HTML pages. That way I can find it when I'm looking for it. These lines of JavaScript could just as easily have gone in the body.

Now, moving to the body we see two links that will move the new window forward or backward:

```
<a href="#" onMouseOver="new_window.focus();">Bring it forward</a>
<a href="#" onMouseOver="new_window.blur();">Put it backward</a>
```

Get it? Let's see ... it's time for an exercise. Try writing this extension to the previous window-reference example. This exercise is pretty tricky; to get it working, you'll have to figure some things out. But give it a try before you View Source to check the answer.

After you're done, come back here and we'll go on to discuss more about the JavaScript Object Model.

More about the JavaScript Document Object Model

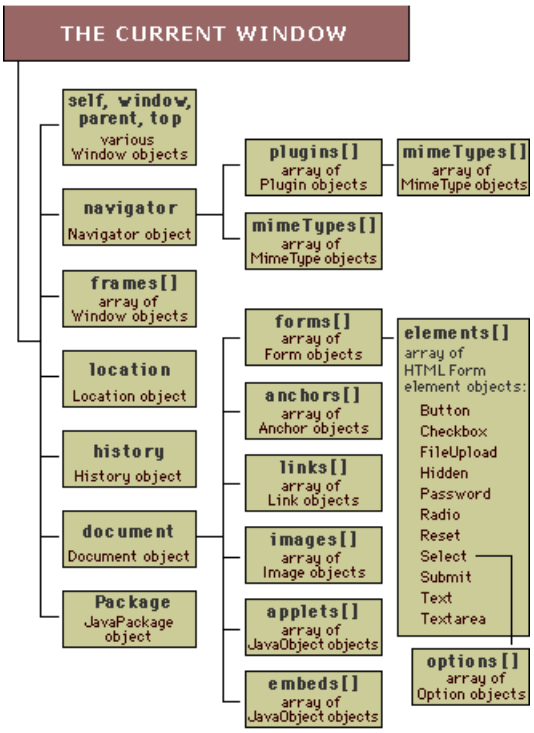
So far we've learned that JavaScript includes default objects, like Windows. We've learned that objects have properties that describe them, and methods that describe what the objects know how to do. Now it's time to delve a little deeper.

One neat thing about objects is that the properties of an object can be objects too. For example, windows have a property called document that refers to the actual HTML document in the window. This document property is itself an object that has properties and methods of its own. We saw an example of this when we talked about image swapping. Harkening back to the last lesson, we learned that you can do an image swap like this:

```
<a href="#" onMouseOver=
"window.document.the_image.src='button_d.gif';">
change</a>
```

That long string, window.document.the_image.src='button_d.gif', translates into: "Find the document property of the window, find the_image property of the document, find the src property of the_image, and set it to button_d.gif." Quite a mouthful, eh? It all works because windows are objects, documents inside windows are objects, and images inside the documents are objects too.

It may seem like a lot of detail to keep track of, but actually it's not too bad. The JavaScript Document Object Model describes a small hierarchy of objects. Here it is:



The top box of the diagram represents your browser window. Following the line from that box down, you'll see it connects to seven more boxes. These are the properties of the browser window. The sixth box there, "document," represents the contents of your window. If you follow the little line leading out of the document box, you'll see it connects to six more boxes. These are the properties of the document object. Notice that the fourth box is "images." This is the list of all the images in your Web page. Because the images are properties of the "document," which is a property of the "window," the precise way to describe an image to JavaScript is to tell it to look at the "window," find the window's "document," and in the document, look for the "image."

We're going to be visiting the various properties of the document object in lesson 5. However, before you can extract the full potential of that object, you should know how to mess with the DOM in other windows.

Messing with the DOM in other Windows

As we've seen, an image swap can be performed with a line like this:

```
window.document.the_image.src="button_d.gif";
```

This works by telling JavaScript to look at a window, find its document, and then find the thing called `the_image` inside that document. Once JavaScript locates the image, it can change its src to whatever GIF we want.

Sometimes it's useful to have a link in one window change an image in another window. Imagine a slide show in which one window displays the images and another little window contains thumbnails of each slide show image. Clicking on a thumbnail in the little window changes the image in the big window. To give you an idea of what I'm talking about, here's an example of a remote slide show control.

There are two windows in this example: the remote control window, and the slide show display window. Clicking on the link above opens the remote control. The remote control then automatically opens the display window by using this line between the <head> tags:

```
var display_window =
window.open("slide_show_main.html","display_window");
```

This opens a new window and assigns the variable `display_window` to that window. So now whenever we want to use JavaScript to refer to that window, we use the variable `display_window`.

When the display window opens, there's a chance it'll open right on top of the remote. To ensure that the remote stays visible, we add this line, which executes after the display window opens:

```
window.focus();
```

Now that we've opened both the display window and the remote, we have to figure out how to make the links in the remote window change the image in the display window. If you view source on the display window, you'll see that it's very simple: It contains just one image, which I've named `main_image`:

```
<img src="sky.gif"
name="main_image" height="400" width="400">
```

To have a link in the remote window change the image in the display window, we have to tell JavaScript to look at the display window (which we do with the `display_window` variable we assigned earlier), find its document, and locate the image called `main_image` inside that document. In JavaScriptese:

```
display_window.document.main_image
```

And here's how you change the src of that image from the remote window:

```
display_window.document.main_image.src = 'sun.gif';
```

If you view source on the remote, you'll see that I've stuck the above line into a link, like so:

```
<a href="#" onClick=
"display_window.document.main_image.src='sky.gif';return false;"><img
src="sky.gif"></a>
```

Now that I've given you a taste of what you can do with the properties of the document object, let's talk about a property of the window object: frames.

Getting Framed

All the remaining properties of windows that we'll see in this lesson relate to frames and how to use them with JavaScript. If you don't know how to use frames, check out Jillo's "Frames Are a Picnic." Read Jillo's wisdom, practice building a few framesets, and then come back. After I show you how to play with frames in JavaScript, I'll give you your homework for the day.

In JavaScript, frames are treated just like windows. Just as you can tweak the contents of one window using JavaScript inside a different window, you can change the contents of one frame using JavaScript in another frame.

Here's a straightforward example of using frames with JavaScript. I'll break it down line by line.

The first thing we need to look at is the frameset. Here it is:

```
<frameset rows="25%,*">
<frame src="frames_example_controls.html" name="control_frame">
<frame src="blank.html" name="target_frame">
</frameset>
```

This is just like any ordinary frameset. The important thing to note and remember is that the frames in the frameset are named. The first frame, called `control_frame`, holds the HTML page that contains the JavaScript. The second frame, as you can see from the `src="blank.html"`, contains nothing.

The second thing to look at is the contents of the `control_frame`. To keep things simple, it only has one interesting line:

```
<a href="#" onClick="top.target_frame.document.writeln('Monkey
do!<br>');">Monkey see</a>
```

This line tries to write the words "Monkey do!
" into the bottom frame. But before JavaScript can write to the frame, it has to know what frame we're talking about. The expression `top.target_frame`, which you'll see in the `onClick`, tells JavaScript to look at the top-most window, which is the browser itself, and find the thing called `target_frame` inside that window. Because we called the bottom frame "target_frame" when defining the frameset, JavaScript will know that `target_frame` means the bottom frame.

Once we've told JavaScript which frame we're talking about, we can treat the frame just like a window. To write the word "howdy" to a window called `greeting_window`, we'd do this:

```
greeting_window.document.writeln("howdy");
```

This tells JavaScript to find the window named `greeting_window`, then find the document of that window, and then write the word "howdy" into that document. Yes, if you haven't figured it out yet, `writeln()` is a method of the document object, which is why we've had to write `document.writeln('blah blah)` instead of just `writeln('blah blah')`.

Because frames are treated just like windows, we can do a similar thing:

```
top.target_frame.document.writeln("Monkey do!");
```

Now, let's learn more about windows and frames.

More about Window and Frame Hierarchies

We've seen a couple of instances of built-in variables being used in the last few examples. One of the built-in variables is `window`. The `window` variable refers to whatever window the JavaScript is executing in. If you have some JavaScript in a frame and you write `window.document.writeln()`, the `writeln` will happen in that frame. People sometimes use the built-in variable called `self` instead of `window`. The two (`window` and `self`) are interchangable.

In the last example, you were introduced to the built-in variable called `top`. This will always refer to the top-most browser window. If you want to start at the top of the window hierarchy, use `top`.

Another built-in variable is `parent`, which refers to the window containing the frame you're currently in. If you have a frameset in a window, and one frame contains another frameset, the second frameset can refer to the enclosing frame using the `parent` variable. This gets a little tricky so I'm not going to go into it, but here's an example of using JavaScript with frames inside frames for you to look over. Check it out or just go on to your homework for Lesson 3.

Get Your Hands Dirty

In this homework assignment, you'll use the assignment from Lesson 2 to start building your dream browser. You'll need to know one more thing before doing the assignment. The document object has a property called `bgColor`. You can set the `bgColor` of a document object to a color just like you do in a body tag: Either use a name, like "red," or a hexadecimal value like `"#FF0000"`.

With that important piece of information in mind, try to write the framesets and JavaScript necessary to do Homework Assignment 3.

After you're done with the assignment, be sure to review what we've covered this lesson.

Lesson 3 Review

We covered four main topics today: windows, frames, the Document Object Model, and Object-Oriented programming. By now you should know about

Opening windows
: How to open windows and make them look the way you want

Communication between windows
: How to get JavaScript in one window to affect another window

Communication between frames
: How to get JavaScript in one frame to affect other frames

Objects
: Objects have properties and methods

The Document Object Model
: A browser window is an object that comprises other objects. If you know the hierarchy of the objects in the DOM you know how to affect different aspects of your HTML page.

Although we didn't go into all the details of all the objects in the DOM, we did get an idea of how it works. In the Lesson 4, we'll cover a few aspects of computer programming that haven't been touched on yet. And in Lesson 5, we'll study the DOM further and go into how most of the objects in an HTML document can be accessed and changed.

Thau! is a core troll at bianca's and is the author of The Book of Javascript: A Practical Guide to Interactive Web Pages.