webmonkey/programming/

Thau's JavaScript Tutorial

by Thau!

Now that you've mastered the basics of computer programming, it's time to refocus on the Document Object Model (DOM). We've already seen that the DOM hierarchy starts with a Window object. Inside each Window object is a Document object. We'll be spending this lesson going through the Document object and seeing how it can be used to get all kinds of information from your users and to dynamically present new information.

We've already had a good look at one of the properties of the Document object, the Images array. If you remember back to Lesson 4, the first image in a document can be modified by changing its src property. For example,

```
window.document.images[0].src='some_new_picture.gif';
```

will change the first image in a document to the GIF called `some_new_picture.gif`. As you might have guessed, each Image in the Image array is itself an object in the DOM. That's why `images[0].src` works. It's saying, get the Image object `image[0]` from the Images array and set its src property. The whole statement above reads in English, "get the document property of this window, get the first image from the document's image array, and change its source property to `some_new_picture.gif`."

The Image object has lots of other interesting properties. For example, you can have your JavaScript check to see if an image has been completely loaded before doing anything. However, these interesting properties will have to wait for later lessons. Gotta keep you coming back somehow, right? Today, we'll be focusing entirely on forms and how to use them in JavaScript.

Introduction to Forms

Forms are part of the HTML 1.0 specification. Many people don't know about them, however, because there's a misconception that they're only useful if you (or someone you know) can do server-side CGI programming. As June points out in her forms tutorial, forms can be fun even if you aren't a CGI programmer. More important for this tutorial, JavaScript can be used to add all sorts of functionality to forms without the help of server-side CGI.

If you don't know how forms work, check out Jay's tutorial and try writing some forms yourself. I'm going to assume that you know the basics.

The first thing you need to know about forms and JavaScript is that forms, like images, are stored in an array of objects. Just like you can refer to the first image on a page by calling it `window.document.images[0]`, you can refer to the first form on a page using `window.document.forms[0]`. Similarly, just like you can name images, you can name forms. For an example if this form ...

Power to the primates!

... is written like this,

```
<form name="first_form">
<input type="text" name="first_text" size="40"
value="Power to the primates!">
</form>
```

You can refer to the form in either of these two ways:

```
var the_form = window.document.forms[0];
var the_same_form = window.document.first_form;
```

Although being able to refer to forms is sometimes useful, more often you want to refer to elements inside of forms,

such as the text field in the last example. Here's an example of manipulating the value of a text field.

Manipulating the Value of a Text Field
Mouse over the links below the text field and see what happens.

Are you happy?

Yes, and I know it. No!

This magic is performed by changing the value of the text field. The form looks like the one in the last example:

```
<form name="first_form">
<input type="text" name="first_text" value="Are you happy?">
</form>
```

The links that change the text field are:

```
<a href="#" onMouseOver="window.document.first_form.first_text.value=
'Clap clap!';">Yes, and I know it.</a>
<a href="#" onMouseOver="window.document.first_form.first_text.value=
'Sour puss!';">No!</a>
```

These are normal mouseovers; the important part is: `window.document.first_form.first_text.value='Clap clap!'`. This says, "find the form called `first_form` in the document, find the form element called `first_text`, and set its value to 'Clap clap!'." The second line works similarly. This is very much like changing the src of an image. Instead of having a src, text fields have values.

Other types of form elements can be affected by messing with their values — for example, textareas:

```
Mouse over below to see the first verse of
The Webmonkey song, adapted from
"I Wanna Be Like You" (The Monkey Song)
from Walt Disney's <CITE>The Jungle
Book</cite>
written by Richard M. Sherman and Robert B.
Sherman
```

Part 1 Part 2

The forms and links here are very similar to those above. The form is:

```
<form name="form_two">
<textarea name="the_textarea" rows=10 cols=60>
Mouse over below to see the first verse of
The Webmonkey song, adapted from
"I Wanna Be Like You" (The Monkey Song)
from Walt Disney's The Jungle Book
written by Richard M. Sherman and Robert B. Sherman
</textarea>
</form>
```

Notice that the form has a name, `form_two`, and the textarea has a name as well, `the_textarea`.

The links are basically the same as what you saw in the text field example:

```
<a href="#" onMouseOver="window.document.form_two.the_textarea.value=
first_part;">Part 1</a>
<a href="#" onMouseOver="window.document.form_two.the_textarea.value=
second_part;">Part 2</a>
```

The only difference is that instead of assigning a string to the value of the textareas, I assigned variables that I defined in the header. View Source to see that they're there. I only did this for the sake of neatness, to get those

long strings out of the HTML. A well-groomed monkey is a happy monkey. Here's one of the strings:

```
var first_part = "Now I'm the king of the swingers\nOh, the jungle VIP\nI've reached the
top and had to stop\nAnd that's what botherin' me";
```

Notice the "\n". This is standard computerese for new lines. In general, because you're working with HTML, the new line isn't important. But if you're writing something in a <pre> tag, or you're writing into a textarea, the "\n" comes in handy.

In addition to changing the values of form elements, JavaScript allows you to detect events that go on inside of them. Here's an example of detecting events inside a text field.

Text Field Events

Text fields understand onBlur, onFocus, and onChange. The onFocus event occurs when someone clicks inside a text field. onBlur happens when somebody moves out of a text field by clicking outside of it, or hitting "tab." onChange happens when somebody changes what's in the text field and then moves outside the text field.

Try doing these things in the text field and see what happens in the textarea below it.

Here's how this works. The text field looks like this:

```
<input type="text" name="first_text"
  onFocus="writeIt('focus');"
  onBlur="writeIt('blur');"
  onChange="writeIt('change');">
```

Each of the event handlers calls the function writeIt(), which I've defined in the header. The header looks like this:

```
<head>
<title>Text Field Events</title>
<script language="JavaScript">
<!-- hide me
function writeIt(the_word)
{
  var word_with_return = the_word + "\n";
  window.document.first_form.the_textarea.value +=
    word_with_return;
}

// show me -->
</script>
</head>
```

This should all look pretty familiar to you. The first few lines are the typical JavaScript preamble and the function definition. The first line in the body of the function ...

```
var word_with_return = the_word + "\n";
```

... initializes a new variable, word_with_return, and sets it equal to the string that was passed into the function concatenated with a "\n". (Remember, the "\n" is standard computerese for "new line.")

The next line ...

```
window.document.first_form.the_textarea.value += word_with_return;
```

... says, "set the value of the textarea to its current value plus the new variable." This shortcut was covered when we learned about loops. It's the same as saying:

```
window.document.first_form.the_textarea.value =
  window.document.first_form.the_textarea.value + word_with_return;
```

It just takes less time. So far, we've seen a property of text fields and textareas (the value) and some events that they can handle. The remaining thing to know about these elements is the methods that they can handle: `blur()`, `focus()`, and `select()`.

Here are some links to show you how `focus()` and `select()` work. Beware that they sometimes stop working after the first time:

Hey, hey, we're the monkeys

Mouseover to focus Mouseover to select

Here are the form and the two links:

```
<form name="method_form">
<input type="text" name="method_text" size=40 value="Hey, hey, we're the monkeys">
</form>

<a href="#" onMouseOver="window.document.method_form.method_text.focus();">Mouseover to focus</a>
<a href="#" onMouseOver="window.document.method_form.method_text.select();">Mouseover to
select</a>
```

Accessing the methods of a text field is just like accessing the methods of any object: `object_name.method()`. The name of a text field is `window.document.form_name.text_field_name`. So, to call the `focus()` method on the text field above, we call:

```
window.document.method_form.method_text.focus();
```

That's pretty much all there is to know about text fields and textareas. We're about ready to try the first exercise for this lesson. First, however, there's one more thing you need to know about form handlers before you can do the exercise.

Form Handlers

Forms are objects; they have their own methods, properties, and event handlers. One event handler that you should know about is `onSubmit`.

`onSubmit` gets called in two situations: if a user clicks on a Submit button, or if a user hits Return in a text field. If these actions are not handled in some way, your JavaScript may behave differently than expected. Try clicking on Submit below and see what happens.

Submit

In Netscape, clicking on an unhandled Submit button generally leads to reloading the page. In general, you won't want this. In order to block this behavior, you need to do this:

```
<form onSubmit="return false;">
<input type="submit" value="Submit">
</form>
```

Try it:

Click on me

Generally, `return false` is a way that JavaScript stops your browser from doing what it would otherwise do. Another example of this is stopping an `href` from going to the URL assigned to it. For example, the link ...

```
<a href="http://www.webmonkey.com/"
onClick="return false;">mattmarg!</a>
```

... won't go anywhere because you've returned false on the onClick. Click on this dead link, if you don't believe me.

This may seem like a weird thing to do, but actually it's quite common, especially in the case of forms. Here's an

example of using a form to get input from a user. Type something into this text field and hit Return:

Who does the monkey love:

Here's the form:

```
<form name="text_entry_form" onSubmit="monkeyLove(); return false;">
<b>Who does the monkey love: </b>
<input type="text" name="monkey_love" size="30">
</form>
```

When you hit Return in the text field, the `onSubmit` handler gets called. It executes the function `monkeyLove()`, which changes the value in the text field.

If the `return false` wasn't in the `onSubmit` handler, the function `monkeyLove()` would execute, changing the text field, but then the page would reload, changing the text field back to what it was. To stop this from happening, you need to have that `return false` in the `onSubmit`.

Here's the `monkeyLove()` function for your perusal:

```
function monkeyLove()
{
  var who_it_is = window.document.text_entry_form.monkey_love.value;
  who_it_is = 'The monkey loves ' + who_it_is;
  window.document.text_entry_form.monkey_love.value = who_it_is;
}
```

And here's an example of the same form without the return false, just so you can see what happens:

Who does the monkey love:

Once you've convinced yourself of the merits of `return false`, it's time to try a text-field exercise.

Text Field Exercise

This exercise is an extension of one we did in Lesson 4. The exercise is to get this textarea to behave like a browser's Location box. If you type `ip://www.webmonkey.com/` or just `www.webmonkey.com`, it'll spawn a window that displays the Mattmarg.com Web site:

Try getting this to work before viewing source to see a solution. When you're done, it's time to move on to checkboxes and radio buttons.

Checkboxes

Text fields and textareas are just two types of form elements. Others that we'll look at in this lesson are checkboxes, radio buttons, and selects. Let's discuss checkboxes first.

Checkboxes have one main property of interest: checked.

- [x] This checkbox is checked
- [ ] This checkbox is not checked

If you have a form named `the_form` and a checkbox named `the_checkbox` you can see if the checkbox has been checked like this:

```
var is_checked = window.document.the_form.the_checkbox.checked;
if (is_checked == true)
{
  alert("Yup, it's checked!");
} else {
  alert("Nope, it's not checked.");
}
```

As you can see, if a checkbox is checked, the checked property will be `true`. `true` is a built-in JavaScript datatype, so you don't have to put it in quotes when checking it. If the checkbox is not checked, the checked property will be false (also a built-in datatype).

Just like you can read the checked property, you can set the checked property. Here's an example of checking and setting the checked property of a checkbox:

Checkbox 1
Click to check Checkbox 1
Click to uncheck Checkbox 1
Click to see the value of Checkbox 1

Here's the code for the above:

```
<form name="form_1">
<input type="checkbox" name="check_1">Checkbox 1
</form>

<a href="#" onClick="window.document.form_1.check_1.checked=true;
return false;">Click to check Checkbox 1</a>

<a href="#" onClick="window.document.form_1.check_1.checked=false;
return false;">Click to uncheck Checkbox 1</a>

<a href="#" onClick="alert(window.document.form_1.check_1.checked);
return false;">Click to see the value of Checkbox 1</a>
```

Notice that I'm sticking the `return false` in the `href`. It stops the page from bouncing around when you click on a link.

Checkboxes have one interesting event handler: `onClick`. When someone clicks on a checkbox, the `onClick` event handler goes into action. Here's an example of it in use.

The Light Switch

This example introduces a few new things. First, there's form that introduces the `onClick` checkbox handler:

```
<form name="form_2">
<input type="checkbox" name ="check_1"
onClick="switchLight();">The Light Switch

</form>
```

When someone clicks on the checkbox, the `onClick` handler gets triggered and executes the `switchLight()` function. Here's the `switchLight()` function (it's in the header of this page):

```
function switchLight()
{
  var the_box = window.document.form_2.check_1;
  var the_switch = "";
  if (the_box.checked == false) {
    alert("Hey! Turn that back on!");
    document.bgColor='black';
  } else {
    alert("Thanks!");
    document.bgColor='white';
  }
}
```

The interesting thing here is the first line:

```
var the_box = window.document.form_2.check_1;
```

This line assigns the checkbox object to a variable. This a handy way to cut down on your typing. It's also a good way to pass objects as parameters to functions. We'll see a lot more of that in later lessons.

That's most of what you have to know about checkboxes. Now let's learn about radio buttons.

Radio Buttons

Happily, radio buttons are almost exactly like checkboxes with respect to JavaScript. The only real difference is in the HTML. Checkboxes are on/off devices. If a checkbox is checked, you can uncheck it. If it's unchecked, you can check it. Radio buttons are different. Once a radio button is on, it stays on until another one is selected. Then the first one goes off. Here's a typical radio button set:

Larry
Moe

○ Curly

As you can see, you can't simply unselect a radio button, you have to choose a new one. With that in mind we can re-do the light switch example with two radio buttons instead of one checkbox:

○ Light off
◉ Light on

This example looks very much like the checkbox example. The form is:

```
<form name="form_1">
<input type="radio" name ="radio_1" onClick="offButton();">Light off
<input type="radio" name ="radio_2" onClick="onButton();" checked>Light on
</form>
```

When the first radio button is clicked, the `offButton()` function is called. That function is:

```
function offButton()
{
  var the_box = window.document.form_1.radio_1;
  if (the_box.checked == true)
  {
    window.document.form_1.radio_2.checked = false;
    document.bgColor='black';
    alert("Hey! Turn that back on!");
  }

}
```

This is pretty much just like the checkbox example earlier. The main difference is this line:

```
window.document.form_1.radio_2.checked = false;
```

This tells JavaScript to turn off the other button when this button has been clicked. The function that is run when the other button is clicked is similar to this one:

```
function onButton()
{
  var the_box = window.document.form_1.radio_2;
  if (the_box.checked == true) {
    window.document.form_1.radio_1.checked = false;
    document.bgColor='white';
    alert("Thanks!");
  }
}
```

There are a few more details about checkboxes and radio buttons, but those can wait until the next set of tutorials. Right now, let's look at the last type of form element, selects.

Selects

Selects are the strangest of the form elements we'll cover in this lesson. There are two primary kinds, pulldown selects and list selects. Here are examples of each:

Pulldown:

probiscus

List:

probiscus
spider
lemur

The thing that makes select objects strange is that the whole select is named, but none of the select's options is. For example, the HTML for the first select looks like this:

```
<select name="pulldown_1" size=1>
<option>probiscus
<option>spider
<option>lemur
<option>chimp
```

```
<option>gorilla
<option>orangutan
</select>
```

Notice that the whole select is named `pulldown_1`, but the individual options aren't named. This makes it difficult to access individual options.

Luckily, through the magic of arrays and objects, there's a way to access and change the options of a select. If you wanted to change the second option of the pulldown menu, you'd type:

```
window.document.form_1.pulldown_1.options[1].text = 'new_text';
```

This works because the select element has an options property that is an array of all the select's options. You can grab one of the options in the array and change its text property. Try it — change the select and then scroll up to see that the pulldown menu actually changed. The second option in the pulldown select should now be `*thau*`.

In addition to the options property, selects have a property called `selectedIndex`. When one option in a select has been chosen, the `selectedIndex` property of the select will be the array number of the selected option. To test this, select one of the options in the list select (the second one) and then check the index. Remember that the first option in the array is option `0`. The line I used to check this was:

```
<a href="#" onClick=
"alert('index is: ' + window.document.form_1.list_1.selectedIndex);
return false;">check the index.</a>
```

The form is named `form_1`, the list select is named `list_1`. To get the `selectedIndex`, I looked at `window.document.form_1.list_1.selectedIndex`. If you want, you can set the `selectedIndex` like this ...

```
window.document.form_1.list_1.selectedIndex = 1;
```

... and highlight the second option in the `list_1` select.

Once you know the index number of the selected option, you can find out what it is:

```
var the_select = window.document.form_1.list_1;
var the_index = the_select.selectedIndex;
var the_selected = the_select.options[the_index].text;
```

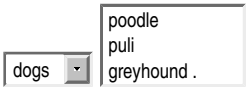
The `selectedIndex` property is great when you only have one option selected. What happens when you have more than one option selected? Well, that gets slightly stranger. If you want to know, here's some information about multiple selects.

Just like the other form elements, the select element has a handler: `onChange()`. This handler gets called whenever there's a change to the select. Here's the example from Lesson 1 to show you how onChange works with a select.

onChange in Select Form Elements

Play with this example and then read the blow-by-blow description below it:

My favorite animal is ...



This is a fairly complicated JavaScript, so let's go through it slowly. First, let's look at the form itself (**note** that your browser might wrap that onChange line, but it should be on one line, no spaces, inyour code):

```
<form name="the_form">
<select name="choose_category"
 onChange="swapOptions
 (window.document.the_form.choose_category.
 options[selectedIndex].text);">
<option selected>Dogs
<option>Fish
<option>Birds
</select>

<select name="the_examples" multiple>
<option>poodle
<option>puli
```

```
<option>greyhound     .
</select>
</form>
```

This form has two elements, a pulldown select and a list select. The pulldown select has an onChange handler that calls a function called `swapOptions()`. `swapOptions()`, which is defined in the header, has one parameter - the selected animal type.

Now let's check out the header. The first thing that happens is I define a few arrays:

```
var dogs = new Array("poodle","puli","greyhound");
var fish = new Array("trout", "mackerel", "bass");
var birds = new Array("robin", "hummingbird", "crow");
```

Notice that the arrays are named the same thing as the animals in the pulldown select. You'll see why soon. Now let's look at the function that gets called when the pulldown select is changed:

```
function swapOptions(the_array_name)
{
 var numbers_select = window.document.the_form.the_examples;
 var the_array = eval(the_array_name);
 setOptionText(window.document.the_form.the_examples, the_array);
}
```

The function definition includes one parameter: `the_array_name`. If you pulled the pulldown select to "Fish," `the_array_name` will equal the string "Fish."

The first line in the body of the function sets a variable to refer to the second form element, the list select.

The second line introduces something new: `eval()`. `eval()` is a little weird and better left for later lessons. The end result of the second line is that the variable `the_array` will equal one of the arrays defined earlier. If `the_array_name` is "Fish," `the_array` will equal the Fish array. If you want a description of how this happens, study up on eval.

The third line of the function calls another function called `setOptionText()`. `setOptionText()` does the work of setting the values of the list select to the contents of `the_array`. Here it is:

```
function setOptionText(the_select, the_array)
{
 for (loop=0; loop < the_select.options.length; loop++)
 {
  the_select.options[loop].text = the_array[loop];
 }
}
```

This function takes two parameters, a reference to a select element and an array. The first line of the function sets up a for loop, which loops through all the options in the select element. Remember that the options property of a select element is an array of all that select's options. Because it's an array, you can find out how many options are in a select using the length property of arrays. That's how the loop works.

The first time you hit the loop, the loop variable equals 0. The body of the loop then reads:

```
the_select.options[0].text = the_array[0];
```

If you chose "Fish" in the pulldown, `the_array[0]` will be "trout," so this line will change the first option in the list select to "trout." The next time through the loop, loop will equal 1, and the second option in the list select will equal "mackerel."

There's one caveat to all this. When you change the text of an option, the size of the select won't change. So if you change an option's text to something really long, it'll probably get cut off.

One way to get around this is to stretch out the select when you first go to it. For example, I did this:

```
<option>greyhound     .
```

Note the period I used to stretch out my select box. It's sort of cheesy, but it works.

OK, that was a mouthful. If you understand how this example works, you understand a lot of JavaScript, and you'll

have no problem doing your homework assignment for this lesson. Once you've done the example and viewed source to see how I did it, go on to this lesson's review.

Review of Lesson 5

Today we focused on forms and what you can do with them. We learned:

Forms and their elements can be named
: You can refer to a form element like this: `window.document.form_name.element_name`

Text field and textarea form elements have value properties
: Refer to the value property like this: `window.document.form_name.element_name.value`; you can set the value by equating it to something, or get the value

Text field and textarea elements understand `focus()`, `blur()`, and `change()` event handlers
: When you click on a text field or textarea, the focus event handler triggers; clicking out of one of these elements triggers the blur event handler; if you hit Tab or Return after changing an element, `onChange()` gets triggered

Forms have an `onSubmit()` event handler
: To stop a form from reloading the page when you click on a Submit button, put `onSubmit="return false;"` in the `<form>` tag

Checkboxes and radio buttons have a checked property
: You can check and uncheck checkboxes and radio buttons by changing their checked property

Checkboxes and radio buttons have an `onClick` event handler
: Just like links, you can trigger functions when a user clicks on a checkbox or radio button

Selects have an options property that is an array of that select's options
: Just like any array, you can find the length of the options array by typing `window.document.form_name.select_name.options.length`

You can change the text of an option by messing with its text property
: To change the text of the first element of a select you'd type `window.document.form_name.select_name.options[0].text='new text';`

You can find out which option was selected using the `selectedIndex` property of the select element
: `window.document.form_name.select_name.selectedIndex` returns the array number, you can then type `window.document.form_name.select_name.options[that_number].text` to find out what was actually selected

Selects have `onChange` event handlers
: If someone changes the thing highlighted in a select, the `onChange` handler gets triggered

You can change the URL of a window using `window.location.replace('new url');`
: This was in the homework

You can create a new option for a select element using new Option
: Since each option in the options array of a select is an object, you can use new to create a new Option; then you can stick this option into the options array

Today was packed, and it concludes this set of five lessons. We covered a lot of territory, from the basics of computer programming to a major portion of the JavaScript DOM. Not surprising, there's still much more to learn. We only covered part of the DOM, and there are more than a few interesting JavaScript objects to study. We also didn't talk too much about making our own objects, nor did we cover all the interesting properties of the Windows object.

Obviously this is a plug for the next set of lessons. So after you give yourself a minute to catch your breath, dive right into my Advanced JavaScript Tutorial.

Thau! is a core troll at bianca's and is the author of The Book of Javascript: A Practical Guide to Interactive Web Pages. He has a large collection of exotic chickens and futurists.