webmonkey/programming/

Thau's JavaScript Tutorial

by Thau!

In the last lesson, you learned about where JavaScript goes and how it looks. Now it's time to start learning the language. In this lesson, you'll learn how JavaScript stores information, how it makes decisions based on that information, and how to change images based on user interaction.

Ready? It's time to learn the fundamentals of computer programming. First stop, variables. If you've taken algebra, you've seen variables. If you haven't taken algebra, don't worry about it. Variables are simply the way JavaScript stores information. For example, if you write "x=2," "x" is a variable that holds the value "2." If you then say "y=x+3," "y" will hold the value "5."

Here's an example of JavaScript that uses variables.

View source on the example, and we'll go through it step by step. Here's what you see:

```
<script language="JavaScript">
<!-- hide me
```

These first two lines you've seen before. They're the typical preamble to any JavaScript program.

```
// load up some variables

var secs_per_min = 60;
var mins_per_hour = 60;
var hours_per_day = 24;
var days_per_year = 365;
```

The first line here is a comment. This comment states the obvious, but it's nice to block off chunks of variable declarations.

The next bunch of lines are variable declarations. There are a few things to notice about these lines:

The first time you use a variable, you should declare it with the word "var."
: Although declaring variables with var is not strictly necessary, it's usually a good idea. When we talk about functions two lessons from now, you'll see why.

Variables must start with either a letter or the underscore character.
: After the first character, variables can have numbers. So monkey_23 is a fine variable name.

Variable names are case-sensitive.
: This means that JavaScript will treat Loop and loop as two different variables. Generally, it's a good idea to pick a naming convention and stick to it. I like having all my variables lowercase, with underscores separating words, like secs_per_min in the example above. Other people prefer using internal capitalization, like secsPerMin.

Variables should describe what they are.
: Variables such as x, y, or hack_hack_hack aren't very useful to a person who's trying to figure out your script. Don't make your variables so long that they take forever to type, but make them long enough to be descriptive.

You can give a variable a value when you declare it, or you can wait until later.
: In the example, each variable was given a value the first time it was used. You don't have to do this, and we'll see examples later on where it's nice to declare a variable even though we don't know the value right away.

Statements end with a semicolon.
: Statements are the sentences of JavaScript and semicolons are the end punctuation marks. Spaces

and line breaks are ignored by the JavaScript interpreter, so the layout of the script serves only to make it more legible for people. This entire example could have been written in one really long line if you take out the comments. But that would be impossible to read.

To be complete, I should mention that in some cases a semicolon isn't necessary and you might see some scripts in which people leave them out. However, it's always a good idea to put the semicolon in there. Not only does it make your program more legible, but it also makes it less likely that adding a line later will mess up your program. Always stick a semicolon at the end of a statement — it is the Webmonkey way.

```
// do some calculations

var secs_per_day = secs_per_min * mins_per_hour * hours_per_day;

var secs_per_year = secs_per_day * days_per_year;
```

Here we see some basic math. After JavaScript executes these statements, the variable secs_per_year will contain whatever you get when you multiply 60, 60, 24, and 365. From this point on, whenever JavaScript sees the variable secs_per_year, it will substitute in that huge number.

```
// end hiding -->
</script>
```

Nothing new here, just the normal way to end a piece of JavaScript.

That's all the JavaScript that's in the header of this example. After JavaScript has executed all this code, the above variables will be declared and given values. That's very nice, but we haven't really done anything with the variables yet. That's what happens in the body of the example.

On to the body of the example!

The Body of the First Variable Example

Now that we've defined the variables, let's do something with them. We start JavaScript in the body of a page just like we do in the head:

```
<script language="JavaScript">
<!-- hide me
```

And here's how we can use JavaScript to write variables and HTML to a Web page:

```
// here's how to use JavaScript to write out HTML

document.writeln("<b>The monkey dances ");

document.writeln(secs_per_year);

document.writeln(" seconds per year.</b><p>");
```

Here are the interesting points about these three lines:

`document.writeln()` (as in "write line") writes whatever's between the parentheses to a Web page.

There's a surprising amount of detail involved in what `document.writeln()` really means. For now, just remember that when you're between <script> and </script> tags, you need to use `document.writeln("blah!")` to write HTML to your Web page.

Characters inside quotes are printed; characters not inside quotes are considered variables.

Notice that in the first and third lines, quotes surround the thing we want printed, while there are no quotes around secs_per_year. Because there are no quotes around secs_per_year, JavaScript thinks it's a variable and swaps in the value of the variable. Luckily, back in the header we defined secs_per_year to be some huge number, so that's what gets printed. If we hadn't defined secs_per_year, JavaScript would probably report an error.

Anything between quotes is called a string and won't get interpreted by JavaScript. This example uses double quotes ("), but it could also use single quotes ('). The two are interchangeable. If the second line were `document.writeln("secs_per_year")`, the JavaScript would write secs_per_year to the page, instead of `31,536,000`.

This distinction between variables and strings is very important, so make sure you understand what I'm saying before you go on.

You can write HTML with the `document.writeln()` command.

Note the and tags in the first and third lines.

That's the rundown on this example. One question that often comes up is, "What JavaScript goes in the head of a page and what goes in the body?"

Usually it doesn't matter, but it's a good idea to put most of your JavaScript in the head of a page. This is because the head gets read before the body, so any variables that appear in the body (like `secs_per_min`) will already have been declared in the head by the time they're needed. If for some reason `secs_per_min` were defined after JavaScript tried to do the `document.writeln(secs_per_min)` command, you'd get a JavaScript error.

OK, we're almost ready to do an exercise using variables, but first, a short note about strings.

The Magic of Strings

As mentioned on the previous page, any group of characters between quotes is called a string. Either single or double quotes will do. Just as variables can hold numbers, variables can hold strings. So it's legal to say:

```
var nice_monkey = "The monkey smiles at you and recites Shakespeare.";

var bad_monkey = "The monkey scowls at you and burps.";
```

Sticking these statements into a JavaScript declares the variables `nice_monkey` and `bad_monkey` and makes them equivalent to these strings. Once you've done this, you can write ...

```
document.writeln(nice_monkey);
```

... whenever you want JavaScript to write out the long message about what nice monkeys do. Here's an example of what you can do with strings. View source and we'll go through the new and interesting parts of this script.

The script starts with something new:

```
var monkey = prompt("What's the monkey's name?", "The monkey");
```

Here we're calling the prompt method to get input from a user. When the prompt method is called, it brings up a dialog box that asks for a user's input. When the user hits OK, the prompt returns whatever is in the input box. In the above line, this returned value gets put into the monkey variable.

Notice that the prompt method takes two parameters, both of which are strings. The first parameter is what gets printed above the input field in the dialog box. In this case it's, "What's the monkey's name?" The second parameter, "The monkey" in this example, is set as the default value of the input box. If you don't want a default value, just put two quote marks as the second parameter. Like this:

```
var monkey = prompt("What's the monkey's name?", "");
```

The next lines are a few straightforward variable assignments, like we've seen before. After these lines we see:

```
var techy_monkey = monkey + demanding + tech;
```

This line introduces a string operator: the plus sign. When the plus sign appears between two strings, or two variables that contain strings as above, it means "concatenate." So the above line creates a new variable called `techy_monkey` that contains a string made of whatever the three variables contain. In this case, it's `"The monkey" + "demands, no, insists upon receiving" + "a computer that won't crash, and a homemade browser!"` In other words ...

```
var techy_monkey = monkey + demanding + tech;
```

... is the same as saying ...

```
var techy_monkey = "The monkey demands, no, insists upon receiving a computer
that won't crash, and a homemade browser!";
```

The next bunch of lines show some more tricks you can do with strings. All the tricks work in similar ways, so we'll just look at three of them:

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

```
var shouting_hippy= hippy_monkey.toUpperCase();

var red_bold_tech = bold_tech.fontcolor('red');
```

The first of these lines says, "Take the string that's contained in the variable hippy_monkey and put it in italics tags." It's actually the same thing as saying

```
var italic_hippy = "<i>" + hippy_monkey + "</i>";
```

But it looks much nicer. In either case, when you later write document.writeln(italic_hippy) in your JavaScript, you get the string in hippy_monkey printed in italics.

The next line, which declares shouting_hippy, shows a string trick that you can't do with HTML. It takes whatever's in hippy_monkey and makes all the letters uppercase.

The third line shows an example of changing a property of a string. All strings have color, and you can change the color of a string using the string.fontcolor('new color'); command. You could also do this:

```
var red_bold_tech = "<font color='red'>" + bold_tech + "</font>";
```

But that's not as easy to read as this:

```
var red_bold_tech = bold_tech.fontcolor('red');
```

You've seen everything else in this example except this line:

```
document.writeln(bold_tech + "<br>");
```

This is just a normal document.writeln(), except instead of just printing a string, we're concatenating two strings and then printing the result. This could have been done in two lines, like this:

```
var broken_bold = bold_tech + "<br>";

document.writeln(broken_bold);
```

But that would involve creating another variable, and writing another line, when it's really unnecessary.

Now that you've learned all about variables and strings, it's time to try an exercise.

Variable Exercise

Try writing out the mad-lib without using any HTML between the <body> and </body> tags.

This JavaScript is non-functional, if you wish to see it in action please refer to the non-print version.
This JavaScript is non-functional, if you wish to see it in action please refer to the non-print version.
THIS JAVASCRIPT IS NON-FUNCTIONAL, IF YOU WISH TO SEE IT IN ACTION PLEASE REFER TO THE NON-PRINT VERSION.
This JavaScript is non-functional, if you wish to see it in action please refer to the non-print version.

Once you've gotten this to work, it's time to learn about if-then clauses.

if-then Branching

What "if-then" statements do is allow your program to behave very differently depending on what a user inputs. For example, you could write a script that would act one way toward you and a different way toward everyone else. Here's the basic form of an if-then statement:

```
if (some condition is true)
{
   do something;
   do something;
   do something;
}
```

The important parts of this structure are:

- It starts with the word "if" (if must be lowercase).

- There is a condition in parentheses that is either true or false.
- There is a set of statements that should be executed if the condition is true. These statements go between curly brackets.

Remember, spacing is only there to make the script more legible. You can put the entire if-then statement on one line if you want, but that would be hard to read.

Here's an example of an if-then statement in action.

Example of a Simple if-then Statement

If you typed yes in the prompt box, you should have received a warm greeting before seeing the rest of the page. If you typed something else, you shouldn't have received any greeting at all.

Here's the the heart of the script:

```
var monkey_love = prompt("Do you love the monkey?","Type yes or no");
if (monkey_love == "yes")
{
alert("Welcome! I'm so glad you came! Please, read on!");
}
```

You've seen the first line before. It just brings up a prompt box and loads the user's response into the variable monkey_love. The second line, however, has something new in it: a condition. This condition says that if the variable monkey_love equals the value "yes," the script should run the statement between the curly brackets. If monkey_love equals something else, the statement will not be run.

Note that the condition is **two** equal signs. This is one of those things that everyone messes up initially. If you put one equal sign instead of two, you're telling JavaScript that monkey_love *should* equal "yes" instead of testing whether or not it actually *does* equal "yes." Luckily, most browsers look for this sort of mistake and will warn you about it when you try to run your script. However, it's best not to make the mistake in the first place.

Other typical conditions are:

```
(variable_1 > variable_2) is true if variable_1 is greater than variable_2
(variable_1 < variable_2) is true if variable_1 is less than variable_2
(variable_2 <= variable_2) is true if variable_1 is less than or equal to
variable_2
(variable_1 != variable_2) is true if variable_1 does not equal variable_2
```

Two ways to make your conditions fancier:

If you want two things to be true before running the statements in the curly brackets, you can do this:

```
if ((age < 21) && (drinking_alcohol == "yes"))
{
document.writeln("Hey! You're too young to drink here!");
}
```

Notice the two ampersands. That's how you say "and" in JavaScript. Notice also that the whole clause, including the two sub-parts and the ampersands must be enclosed in parentheses.

If you want either one of two things to be true before running the statements in the curly brackets, do this:

```
if ((variable_1 == "bananas") || (variable_1 == "JavaScript"))
{
document.writeln("The monkey is happy because it has " + variable_1);
}
```

On to the if-then exercise!

if-then Exercise

This JavaScript is non-functional, if you wish to see it in action please refer to the non-print version.
Once you've gotten this to work, it's time to learn about link events.

Link Events

Whenever a user clicks on a link, or moves her cursor over one, JavaScript is sent a link event. One link

event is called onClick, and it gets sent whenever someone clicks on a link. Another link event is called onMouseOver. This one gets sent when someone moves the cursor over the link.

You can use these events to affect what the user sees on a page. Here's an example of how to use link events. Try it out, view source, and we'll go over it line by line.

The first interesting thing is that there are no <script> tags. That's because anything that appears in the quotes of an onClick or an onMouseOver is automatically interpreted as JavaScript. In fact, because semicolons mark the end of statements allowing you to write entire JavaScripts in one line, you can fit an entire JavaScript program between the quotes of an onClick. It'd be ugly, but you could do it.

Here's the first line of interest:

```
<a href="#" onClick="alert('Ooo, do it again!');">Click on me!</a>
```

This is just like a normal anchor tag, but it has the magic onClick="" element, which says, "When someone clicks on this link, run the little bit of JavaScript between my quotes." Notice, there's even a terminating semicolon at the end of the alert.

Also note that there's a # sign between the quotes in the href="#". This makes it so the browser doesn't try to load another Web page when you click on the link. If you put http://www.webmonkey.com/ in the href instead of the #, the browser would run the JavaScript in the onClick and the load up this here Webmonkey site.

Unfortunately, sometimes using the # sign causes the browser to jump to the top of the page when the link is clicked. To stop that from happening, change the link to this:

```
<a href="#"onClick="alert('Ooo, do it again!');return false;">Click on me!</a>
```

Notice the return false; following the alert inside the onClick. This tells JavaScript to stop the browser from even looking at what's in the href. Because some very old browsers don't understand the return false, people usually do both things: Put the # inside the href and put return false inside the onClick. That makes sure all browsers do the right thing.

Here's the next line:

```
<a href="#" onMouseOver="alert('Hee hee!');">Mouse over me!</a>
```

This is just like the first line, but it uses an onMouseOver instead of an onClick.

Now that we've learned about link events, we can go into the wonderful land of image swaps.

Image Swapping

One of the most commonly used features of JavaScript is the ability to change images on a mouseover. Unfortunately, Microsoft Internet Explorer 3.0 does not support image replacement. So if you're using IE 3.0, it might be time to upgrade to IE 4.0 or switch to Netscape.

Here's a quick and dirty example of a basic image swap.

Let's go through the example step by step. The first line of interest is:

```
<img src="button_r.gif" name="the_image">
```

This is just like a standard <img src= > tag except this one has been given a name: the_image. This name could be anything: my_image, a_box, whatever — but it can't have any spaces in it.

The next line of interest is:

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

This is where the image swap happens. It's just like the onMouseOver you saw before. The active piece of JavaScript, which appears in the quotes of the onMouseOver is this:

```
document.the_image.src='button_d.gif';
```

This statement says, "find the image called 'the_image' and change its src to button_d.gif." Note that there are double quotes around the whole statement, and 'button_d.gif' takes single quotes. Although quotes are interchangable, if you have one set of quotes inside another set of quotes, the sets have to

be of different kinds. So you could either do " 'something' " or ' "something" ' but not " 'something" ' or " "something" ". Got it?

Just as there was a lot of detail in what makes `document.writeln()` work, I'm not telling you exactly how this image swap is working. You'll learn the details of both when we look at object-oriented programming and the Document Object Model in the next lesson.

An important caveat about image swapping is that the image you're switching to should be the same size as the original. If it's not, it'll get smashed or stretched to fit the original's size.

For now, you're ready for today's homework.

Exercise 2

Browser Configuration

Welcome This JavaScript is non-functional, if you wish to see it in action please refer to the non-print version.

Brand your browser by moving your mouse over the image below until you find one you like. Then click on the image.

Your homework is to write a JavaScript program that behaves like this. It's a bit tricky, so give yourself some time. After you're through, view source on this page to see how I did it and then go to the review of Lesson 2.

Review

To review, here are the things we covered today. If you feel like you didn't learn one of these things, go back and hunt for it.

Variables
: Variables can hold numbers and strings. There are a few restrictions and rules of thumb to keep in mind when naming variables.

Statements
: Statments end in a semicolon.

Strings
: Strings are sequences of characters in quotation marks. You can use either type of quote, single or double. There are lots of neat things you can do to manipulate the way a string will print out. And you can use + to concatenate two strings.

`document.writeln()`
: Use `document.writeln()` to write text and HTML to a Web page.

`prompt`
: You can use `prompt` to get input from users.

`if-then-else`
: You use `if-then-else` clauses to make your JavaScript behave differently depending on user actions.

Link events
: `onClick` and `onMouseOver` inside an `href` can be used to run bits of JavaScript that react to user actions.

image swaps
: By naming images, we can use JavaScript to change what image is displayed.

If you made it through all that stuff, Congratulations! It was a lot to learn. In Lesson 3 we'll be

burrowing into the heart of JavaScript: the Document Object Model. We'll also learn about how to open and manipulate windows and frames, and we'll begin building our new browser.

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.

Feedback | Help | About Us | Jobs | Advertise | Privacy Statement | Terms of Service

Copyright © 1994-2004 Wired Digital Inc., a Lycos Network site. All rights reserved.